Contents

1 Principles of Numerical Calculations **1**

1.1 Introduction . . . 1
1.2 Common Ideas and Concepts . . . 2
1.2.1 Fixed-Point Iteration . . . 2
1.2.2 Linearization and Extrapolation . . . 5
1.2.3 Finite Difference Approximations . . . 9
Review Questions . . . 12
Problems and Computer Exercises . . . 13
1.3 Some Numerical Algorithms . . . 14
1.3.1 Recurrence Relations . . . 14
1.3.2 Divide and Conquer Strategy . . . 16
1.3.3 Approximation of Functions . . . 18
1.3.4 The Principle of Least Squares . . . 20
Review Questions . . . 22
Problems and Computer Exercises . . . 22
1.4 Matrix Computations . . . 24
1.4.1 Matrix Multiplication . . . 25
1.4.2 Solving Triangular Systems . . . 26
1.4.3 Gaussian Elimination . . . 28
1.4.4 Sparse Matrices and Iterative Methods . . . 34
1.4.5 Software for Matrix Computations . . . 36
Review Questions . . . 37
Problems and Computer Exercises . . . 38
1.5 Numerical Solution of Differential Equations . . . 39
1.5.1 Euler's Method . . . 39
1.5.2 An Introductory Example . . . 39
1.5.3 A Second Order Accurate Method . . . 43
Review Questions . . . 47
Problems and Computer Exercises . . . 48
1.6 Monte Carlo Methods . . . 49
1.6.1 Origin of Monte Carlo Methods . . . 49
1.6.2 Random and Pseudo-Random Numbers . . . 51
1.6.3 Testing Pseudo-Random Number Generators . . . 55
1.6.4 Random Deviates for Other Distributions. . . . 58

1.6.5 Reduction of Variance. 61
Review Questions 66
Problems and Computer Exercises 66

Bibliography **69**

Index **72**

Chapter 1

Principles of Numerical Calculations

1.1 Introduction

Although mathematics has been used for centuries in one form or another within many areas of science and industry, modern scientific computing using electronic computers has its origin in research and developments during the second world war. In the late forties and early fifties the foundation of numerical analysis was laid as a separate discipline of mathematics. The new capabilities of performing millions of operations led to new classes of algorithms, which needed a careful analysis to ensure their accuracy and stability.

Recent modern development has increased enormously the scope for using numerical methods. Not only has this been caused by the continuing advent of faster computers with larger memories. Gain in problem solving capabilities through better mathematical algorithms have in many cases played an equally important role! This has meant that today one can treat much more complex and less simplified problems through massive amounts of numerical calculations. This development has caused the always close interaction between mathematics on the one hand and science and technology on the other to increase tremendously during the last decades. Advanced mathematical models and methods are now used more and more also in areas like medicine, economics and social sciences. It is fair to say that today experiment and theory, the two classical elements of scientific method, in many fields of science and engineering are supplemented in many areas by computations as an equally important component.

As a rule, applications lead to mathematical problems which in their complete form cannot be conveniently solved with exact formulas unless one restricts oneself to special cases or simplified models which can be exactly analyzed. In many cases, one thereby reduces the problem to a linear problem—for example, a linear system of equations or a linear differential equation. Such an approach can quite often lead to concepts and points of view which can, at least qualitatively, be used even in the unreduced problems.

1.2 Common Ideas and Concepts

In most numerical methods one applies a small number of general and relatively simple ideas. These are then combined in an inventive way with one another and with such knowledge of the given problem as one can obtain in other ways—for example, with the methods of mathematical analysis. Some knowledge of the background of the problem is also of value; among other things, one should take into account the order of magnitude of certain numerical data of the problem.

In this chapter we shall illustrate the use of some general ideas behind numerical methods on some simple problems which may occur as subproblems or computational details of larger problems, though as a rule they occur in a less pure form and on a larger scale than they do here. When we present and analyze numerical methods, we use to some degree the same approach which was described first above: we study in detail special cases and simplified situations, with the aim of uncovering more generally applicable concepts and points of view which can guide in more difficult problems.

It is important to have in mind that the success of the methods presented depends on the smoothness properties of the functions involved. In this first survey we shall tacitly assume that the functions have as many well-behaved derivatives as is needed.

1.2.1 Fixed-Point Iteration

One of the most frequently recurring ideas in many contexts is **iteration** (from the Latin *iteratio*, "repetition") or **successive approximation**. Taken generally, iteration means the repetition of a pattern of action or process. Iteration in this sense occurs, for example, in the repeated application of a numerical process—perhaps very complicated and itself containing many instances of the use of iteration in the somewhat narrower sense to be described below—in order to improve previous results. To illustrate a more specific use of the idea of iteration, we consider the problem of solving a nonlinear equation of the form

$$x = F(x), \tag{1.2.1}$$

where F is assumed to be a differentiable function whose value can be computed for any given value of a real variable x, within a certain interval. Using the method of iteration, one starts with an initial approximation x_0, and computes the sequence

$$x_1 = F(x_0), \qquad x_2 = F(x_1), \qquad x_3 = F(x_2), \ldots \tag{1.2.2}$$

Each computation of the type $x_{n+1} = F(x_n)$ is called an iteration. If the sequence $\{x_n\}$ converges to a limiting value α then we have

$$\alpha = \lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} F(x_n) = F(\alpha),$$

so $x = \alpha$ satisfies the equation $x = F(x)$. As n grows, we would like the numbers x_n to be better and better estimates of the desired root. One then stops the iterations when sufficient accuracy has been attained.

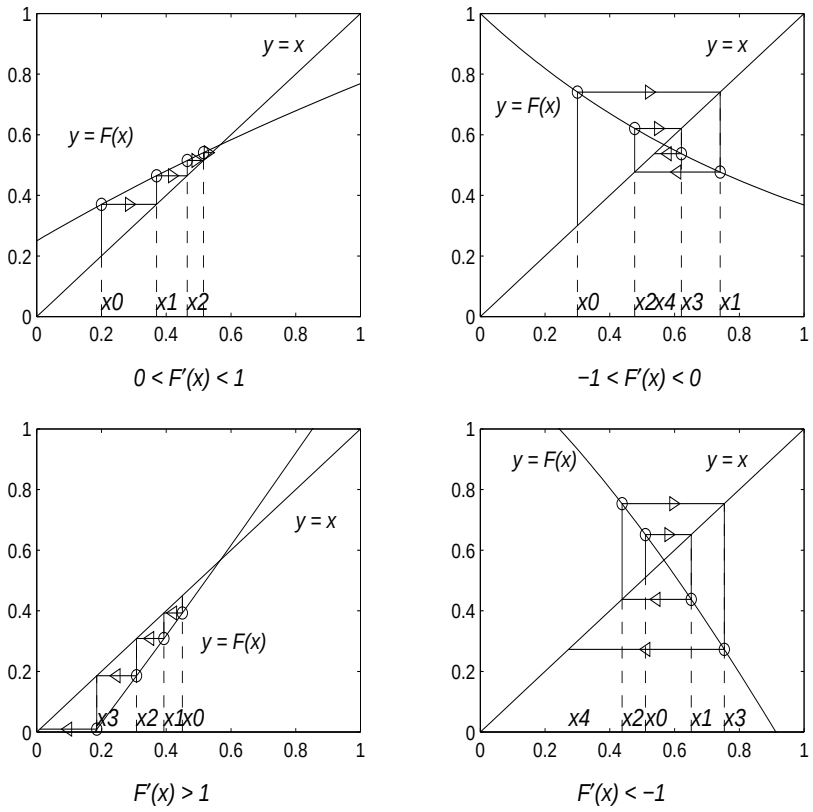


Figure 1.2.1. (a)–(d) *Geometric interpretation of iteration* $x_{n+1} = F(x_n)$.

A geometric interpretation is shown in Fig. 1.2.1. A root of Equation (1.2.1) is given by the abscissa (and ordinate) of an intersecting point of the curve $y = F(x)$ and the line $y = x$. Using iteration and starting from x_0 we have $x_1 = F(x_0)$. The point x_1 on the x-axis is obtained by first drawing a horizontal line from the point $(x_0, F(x_0)) = (x_0, x_1)$ until it intersects the line $y = x$ in the point (x_1, x_1) and from there drawing a vertical line to $(x_1, F(x_1)) = (x_1, x_2)$ and so on in a "staircase" pattern. In Fig. 1.2.1a it is obvious that $\{x_n\}$ converges monotonically to α. Fig. 1.2.1b shows a case where F is a decreasing function. There we also have convergence but not monotone convergence; the successive iterates x_n are alternately to the right and to the left of the root α.

But there are also divergent cases, exemplified by Figs. 1.2.1c and 1.2.1d. One can see geometrically that the quantity which determines the rate of convergence (or divergence) is the slope of the curve $y = F(x)$ in the neighborhood of the root. Indeed, from the mean value theorem we have

$$\frac{x_{n+1} - \alpha}{x_n - \alpha} = \frac{F(x_n) - F(\alpha)}{x_n - \alpha} = F'(\xi_n),$$

where ξ_n lies between x_n and α. We see that, if x_0 is chosen sufficiently close to the root, (yet $x_0 \neq \alpha$), the iteration will diverge if $|F'(\alpha)| > 1$ and converge if $|F'(\alpha)| < 1$. In these cases the root is called, respectively, repulsive and attractive. We also see that the convergence is faster the smaller $|F'(\alpha)|$ is.

Example 1.2.1.

A classical fast method for calculating square roots:

The equation $x^2 = c$ $(c > 0)$ can be written in the form $x = F(x)$, where $F(x) = \frac{1}{2}(x + c/x)$. If we set

$$x_0 > 0, \quad x_{n+1} = \tfrac{1}{2}(x_n + c/x_n),$$

then the $\alpha = \lim_{n\to\infty} x_n = \sqrt{c}$ (see Fig. 1.2.2)

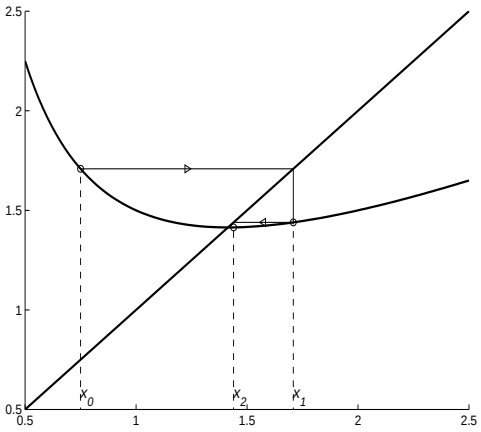


Figure 1.2.2. *The fix-point iteration* $x_n = (x_n + c/x_n)/2$, $c = 2$, $x_0 = 0.75$.

For $c = 2$, and $x_0 = 1.5$, we get $x_1 = \frac{1}{2}(1.5 + 2/1.5) = 1\frac{5}{12} = 1.4166666\ldots$, and

$$x_2 = 1.414215\,686274, \qquad x_3 = 1.414213\,562375,$$

which can be compared with $\sqrt{2} = 1.414213\,562373\ldots$ (correct to digits shown). As can be seen from Fig. 1.2.2 a rough value for x_0 suffices. The rapid convergence is due to the fact that for $\alpha = \sqrt{c}$ we have

$$F'(\alpha) = (1 - c/\alpha^2)/2 = 0.$$

One can in fact show that if x_n has t correct digits, then x_{n+1} will have at least $2t - 1$ correct digits; see Example 6.3.3 and the following exercise. The above iteration method is used quite generally on both pocket calculators and computers for calculating square roots. The computation converges for any $x_0 > 0$.

Iteration is one of the most important aids for the practical as well as theoretical treatment of both linear and nonlinear problems. One very common application

of iteration is to the solution of *systems of equations*. In this case $\{x_n\}$ is a sequence of vectors, and F is a vector-valued function. When iteration is applied to *differential equations* $\{x_n\}$ means a sequence of functions, and $F(x)$ means an expression in which integration or other operations on functions may be involved. A number of other variations on the very general idea of iteration will be given in later chapters.

The form of equation (1.2.1) is frequently called the **fixed point form**, since the root α is a fixed point of the mapping F. An equation may not be given originally in this form. One has a certain amount of choice in the rewriting of equation $f(x) = 0$ in fixed point form, and the rate of convergence depends very much on this choice. The equation $x^2 = c$ can also be written, for example, as $x = c/x$. The iteration formula $x_{n+1} = c/x_n$, however, gives a sequence which alternates between x_0 (for even n) and c/x_0 (for odd n)—the sequence does not even converge!

Let an equation be given in the form $f(x) = 0$, and for any $k \neq 0$, set

$$F(x) = x + kf(x).$$

Then the equation $x = F(x)$ is equivalent to the equation $f(x) = 0$. Since $F'(\alpha) = 1 + kf'(\alpha)$, we obtain the fastest convergence for $k = -1/f'(\alpha)$. Because α is not known, this cannot be applied literally. However, if we use x_n as an approximation this leads to the choice $F(x) = x - f(x)/f'(x)$, or the iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{1.2.3}$$

This is the celebrated **Newton's method**.[1] (Occasionally this method is referred to as the Newton–Raphson method.) We shall derive it in another way below.

Example 1.2.2.

The equation $x^2 = c$ can be written in the form $f(x) = x^2 - c = 0$. Newton's method for this equation becomes

$$x_{n+1} = x_n - \frac{x_n^2 - c}{2x_n} = \frac{1}{2}\Big(x_n + \frac{c}{x_n}\Big),$$

which is the fast method in Example 1.2.1.

1.2.2 Linearization and Extrapolation

Another often recurring idea is that of **linearization**. This means that one *locally*, i.e. in a small neighborhood of a point, *approximates a more complicated function with a linear function*. We shall first illustrate the use of this idea in the solution of the equation $f(x) = 0$. Geometrically, this means that we are seeking the intersection point between the x-axis and the curve $y = f(x)$; see Fig. 1.2.3. Assume that

[1] Isaac Newton (1642–1727), English mathematician, astronomer and physicist, invented, independently of the German mathematician and philosopher Gottfried W. von Leibniz (1646–1716), the infinitesimal calculus. Newton, the Greek mathematician Archimedes (287–212 B.C.) and the German mathematician Carl Friedrich Gauss (1777–1883) gave pioneering contributions to numerical mathematics and to other sciences.

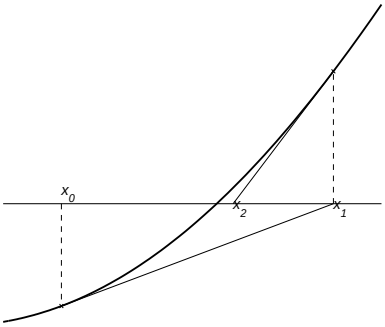


Figure 1.2.3. *Newton's method.*

we have an approximating value x_0 to the root. We then approximate the curve with its *tangent* at the point $(x_0, f(x_0))$. Let x_1 be the abscissa of the point of intersection between the x-axis and the tangent. Since the equation for the tangent reads

$$y - f(x_0) = f'(x_0)(x - x_0),$$

we obtain by setting $y = 0$, the approximation

$$x_1 = x_0 - f(x_0)/f'(x_0).$$

In many cases x_1 will have about twice as many correct digits as x_0. However, if x_0 is a poor approximation and $f(x)$ far from linear, then it is possible that x_1 will be a worse approximation than x_0.

If we combine the ideas of iteration and linearization, that is, we substitute x_n for x_0 and x_{n+1} for x_1, we rediscover Newton's method mentioned earlier. If x_0 is close enough to α the iterations will converge rapidly; see Fig. 1.2.3, but there are also cases of divergence.

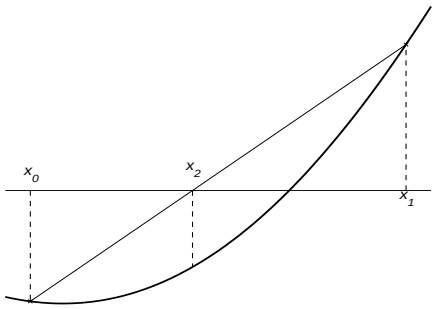


Figure 1.2.4. *The secant method.*

Another way, instead of drawing the tangent, to approximate a curve locally with a linear function is to choose two neighboring points on the curve and to approximate the curve with the *secant* which joins the two points; see Fig. 1.2.4. The

secant method for the solution of nonlinear equations is based on this approximation. This method, which preceded Newton's method, is discussed more closely in Sec. 6.4.1.

Newton's method can easily be generalized to solve a *system of nonlinear equations*

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad i = 1 : n.$$

or $f(x) = 0$, where f and x now are vectors in $\mathbf{R}^n$. Then x_{n+1} is determined by the *system of linear equations*

$$f'(x_n)(x_{n+1} - x_n) = f(x_n), \tag{1.2.4}$$

where

$$f'(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \in \mathbf{R}^{n \times n}, \tag{1.2.5}$$

is the matrix of partial derivatives of f with respect to x. This matrix is called the **Jacobian** of f and often denoted by $J(x)$. System of nonlinear equations arise in many different contexts in scientific computing, e.g., in the solution of differential equations and optimization problems. We shall several times, in later chapters, return to this fundamental concept.

The secant approximation is useful in many other contexts. It is, for instance, generally used when one "reads between the lines" or interpolates in a table of numerical values. In this case the secant approximation is called **linear interpolation**. When the secant approximation is used in **numerical integration**, that is in the approximate calculation of a definite integral,

$$I = \int_a^b y(x)\, dx, \tag{1.2.6}$$

(see Fig. 1.2.5) it is called the **trapezoidal rule**. With this method, the area between the curve $y = y(x)$ and the x-axis is approximated with the sum $T(h)$ of the areas of a series of parallel trapezoids.

Using the notation of Fig. 1.2.5, we have

$$T(h) = h\frac{1}{2}\sum_{i=0}^{n-1}(y_i + y_{i+1}), \quad h = \frac{b-a}{n}. \tag{1.2.7}$$

(In the figure, $n = 4$.) We shall show in a later chapter that the error is very nearly proportional to h^2 when h is small. One can then, in principle, attain arbitrary high accuracy by choosing h sufficiently small. However, the computational work involved is roughly proportional to the number of points where $y(x)$ must be computed, and thus inversely proportional to h. Thus the computational work grows rapidly as one demands higher accuracy (smaller h).

Numerical integration is a fairly common problem because in fact it is quite seldom that the "primitive" function can be analytically calculated in a finite expression containing only elementary functions. It is not possible, for example, for

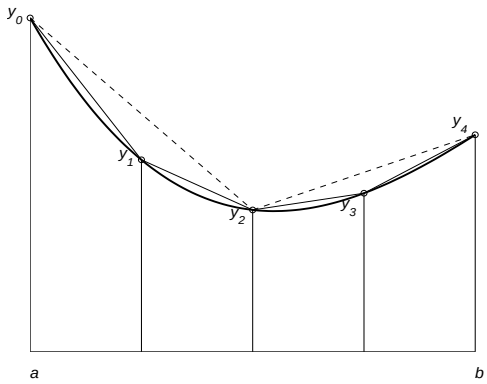


Figure 1.2.5. *Numerical integration by the trapezoidal rule ($n = 4$).*

such simple functions as e^{x^2} or $(\sin x)/x$. In order to obtain *higher accuracy* with significant less work than the trapezoidal rule requires, one can use one of the following two important ideas:

(a) **Local approximation** of the integrand with a polynomial of higher degree, or with a function of some other class, for which one knows the primitive function.

(b) Computation with the trapezoidal rule for several values of h and then extrapolation to $h = 0$, so-called **Richardson extrapolation**[2] or **the deferred approach to the limit**, with the use of general results concerning the dependence of the error on h.

The technical details for the various ways of approximating a function with a polynomial, among others Taylor expansions, interpolation, and the method of least squares, are treated in later chapters.

The extrapolation to the limit can easily be applied to numerical integration with the trapezoidal rule. As was mentioned previously, the trapezoidal approximation (1.2.7) to the integral has an error approximately proportional to the square of the step size. Thus, using two step sizes, h and $2h$, one has:

$$T(h) - I \approx kh^2, \qquad T(2h) - I \approx k(2h)^2,$$

and hence $4(T(h) - I) \approx T(2h) - I$, from which it follows that

$$I \approx \tfrac{1}{3}(4T(h) - T(2h)) = T(h) + \tfrac{1}{3}(T(h) - T(2h)).$$

Thus, by adding the corrective term $\frac{1}{3}(T(h) - T(2h))$ to $T(h)$, one should get an estimate of I which typically is far more accurate than $T(h)$. In Sec. 3.6 we shall see

[2]Lewis Fry Richardson (1881–1953) studied mathematics, physics, chemistry, botany and zoology. He graduated from King's College, Cambridge 1903. He was the first (1922) to attempt to apply the method of finite differences to weather prediction, long before the computer age!

that the improvements is in most cases quite striking. The result of the Richardson extrapolation is in this case equivalent to the classical **Simpson's rule**[3] for numerical integration, which we shall encounter many times in this volume. It can be derived in several different ways. Sec. 3.6 also contains application of extrapolation to other problems than numerical integration, as well as a further development of the extrapolation idea, namely **repeated Richardson extrapolation**. In numerical integration this is also known as **Romberg's method**.

Knowledge of the behavior of the error can, together with the idea of extrapolation, lead to a powerful method for improving results. Such a line of reasoning is useful not only for the common problem of numerical integration, but also in many other types of problems.

Example 1.2.3.

The integral $\int_{10}^{12} f(x)\,dx$ is computed for $f(x) = x^3$ by the trapezoidal method. With $h = 1$ we obtain

$$T(h) = 2,695, \qquad T(2h) = 2,728,$$

and extrapolation gives $T = 2.684$, equal to the exact result. Similarly, for $f(x) = x^4$ we obtain

$$T(h) = 30,009, \qquad T(2h) = 30,736,$$

and with extrapolation $T = 29,766.7$ (exact $29,766.4$).

1.2.3 Finite Difference Approximations

The local approximation of a complicated function by a linear function leads to another frequently encountered idea in the construction of numerical methods, namely the approximation of a derivative by a difference quotient. Fig. 1.2.6 shows the graph of a function $y(x)$ in the interval $[x_{n-1}, x_{n+1}]$ where $x_{n+1} - x_n = x_n - x_{n-1} = h$; h is called the step size. If we set $y_i = y(x_i)$, $i = n-1, n, n+1$, then the derivative at x_n can be approximated by a **forward difference quotient**,

$$y'(x_n) \approx \frac{y_{n+1} - y_n}{h}, \tag{1.2.8}$$

or a similar backward difference quotient involving y_n and y_{n-1}. The error in the approximation is called a **discretization error**.

However, it is conceivable that the **centered difference approximation**

$$y'(x_n) \approx \frac{y_{n+1} - y_{n-1}}{2h} \tag{1.2.9}$$

will usually be more accurate. It is in fact easy to motivate this. By Taylor's formula,

$$y(x+h) - y(x) = y'(x)h + y''(x)h^2/2 + y'''(x)h^3/6 + \dots \tag{1.2.10}$$

$$-y(x-h) + y(x) = y'(x)h - y''(x)h^2/2 + y'''(x)h^3/6 - \dots \tag{1.2.11}$$

[3] Thomas Simpson (1710–1761), English mathematician best remembered for his work on interpolation and numerical methods of integration. He taught mathematics privately in the London coffee–houses and from 1737 began to write texts on mathematics.

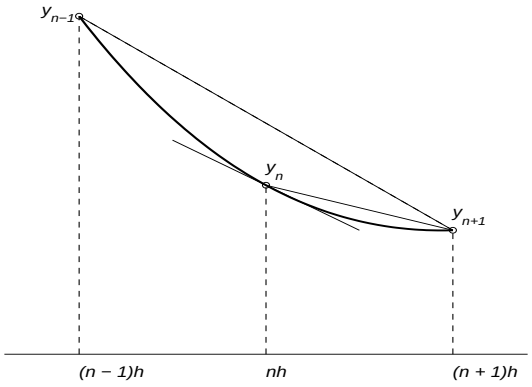


Figure 1.2.6. *Finite difference quotients.*

Set $x = x_n$. Then, by the first of these equations,

$$y'(x_n) = \frac{y_{n+1} - y_n}{h} - \frac{h}{2}y''(x_n) + \dots$$

Next, add the two Taylor expansions and divide by $2h$. Then the first error term cancels and we have

$$y'(x_n) = \frac{y_{n+1} - y_{n-1}}{2h} + \frac{h^2}{6}y'''(x_n) + \dots$$

We shall in the sequel call a formula (or a method), where a step size parameter h is involved, **accurate of order** p, if its error is approximately proportional to h^p. Since $y''(x)$ vanishes for all x if and only if y is a linear function of x, and similarly, $y'''(x)$ vanishes for all x if and only if y is a quadratic function, we have established the following important result:

Lemma 1.2.1. *The forward difference approximation* (1.2.8) *is exact only for a linear function, and it is only first order accurate in the general case. The centered difference approximation* (1.2.9) *is exact also for a quadratic function, and is second order accurate in the general case.*

For the above reason the approximation (1.2.9) is, in most situations, preferable to (1.2.8). However, there are situations when these formulas are applied to the approximate solution of differential equations where the forward difference approximation suffices, but where the centered difference quotient is entirely unusable, for reasons which have to do with how errors are propagated to later stages in the calculation. We shall not discuss it more closely here, but mention it only to intimate some of the surprising and fascinating mathematical questions which can arise in the study of numerical methods.

Higher derivatives are approximated with **higher differences**, that is, differ-

ences of differences, another central concept in numerical calculations. We define:

$$\begin{aligned}(\Delta y)_n &= y_{n+1} - y_n;\\ (\Delta^2 y)_n = (\Delta(\Delta y))_n &= (y_{n+2} - y_{n+1}) - (y_{n+1} - y_n)\\ &= y_{n+2} - 2y_{n+1} + y_n;\\ (\Delta^3 y)_n = (\Delta(\Delta^2 y))_n &= y_{n+3} - 3y_{n+2} + 3y_{n+1} - y_n;\end{aligned}$$

etc. For simplicity one often omits the parentheses and writes, for example, $\Delta^2 y_5$ instead of $(\Delta^2 y)_5$. The coefficients that appear here in the expressions for the higher differences are, by the way, the binomial coefficients. In addition, if we denote the step length by Δx instead of by h, we get the following formulas, which are easily remembered:

$$\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x}, \qquad \frac{d^2y}{dx^2} \approx \frac{\Delta^2 y}{(\Delta x)^2}, \tag{1.2.12}$$

etc. Each of these approximations is second order accurate for the value of the derivative at an x which equals the *mean value* of the largest and smallest x for which the corresponding value of y is used in the computation of the difference. (The formulas are only first order accurate when regarded as approximations to derivatives at other points between these bounds.) These statements can be established by arguments similar to the motivation for the formulas (1.2.8) and (1.2.9).

Taking the difference of the Taylor expansions (1.2.10)–(1.2.11) with one more term in each, and dividing by h^2 we obtain the following important formula

$$y''(x_n) = \frac{y_{n+1} - 2y_n + y_{n-1}}{h^2} - \frac{h^2}{12}y^{iv}(x_n) + \cdots,$$

Introducing the **central difference operator**

$$\delta y_n = \left(x_n + \tfrac{1}{2}h\right) - y\left(x_n - \tfrac{1}{2}h\right), \tag{1.2.13}$$

and neglecting higher order terms we get

$$y''(x_n) \approx \frac{1}{h^2}\delta^2 y_n - \frac{h^2}{12}y^{iv}(x_n). \tag{1.2.14}$$

The approximation of equation (1.2.9) can be interpreted as an application of (1.2.12) with $\Delta x = 2h$, or else as the mean of the estimates which one gets according to equation (1.2.12) for $y'((n + \frac{1}{2})h)$ and $y'((n - \frac{1}{2})h)$.

When the values of the function have errors, for example, when they are rounded numbers, the difference quotients become more and more uncertain the less h is. Thus if one wishes to compute the derivatives of a function given by a table, one should as a rule use a step length which is greater than the table step.

Example 1.2.4.

For $y = \cos x$ one has, using function values correct to six decimal digits:

This arrangement of the numbers is called a **difference scheme**. Note that the differences are expressed in units of 10^{-6}. Using (1.2.9) and (1.2.12) one gets

$$\begin{aligned}y'(0.60) &\approx (0.819648 - 0.830941)/0.02 = -0.56465,\\ y''(0.60) &\approx -83 \cdot 10^{-6}/(0.01)^2 = -0.83.\end{aligned}$$

x	y	Δy	$\Delta^2 y$
0.59	0.830941		
		-5605	
0.60	0.825336		-83
		-5688	
0.61	0.819648		

The correct results are, with six decimals,

$$y'(0.60) = -0.564642, \quad y''(0.60) = -0.825336.$$

In y'' we only got two correct decimal digits. This is due to **cancellation**, which is an important cause of loss of accuracy; see further Sec. 2.2.3. Better accuracy can be achieved by *increasing* the step h; see Problem 5 at the end of this section.

Finite difference approximations are useful for partial derivatives too. Suppose that the values $u_{i,j} = u(x_i, y_j)$ of a function $u(x, y)$ are given on a square grid with grid size h, i.e. $x_i = x_0 + ih$, $y_j = y_0 + jh$, $0 \le i \le M$, $0 \le j \le N$ that covers a rectangle. A very important equation of Mathematical Physics is **Poisson's equation**:[4]

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \tag{1.2.15}$$

where $f(x, y)$ is a given function. Under certain conditions, gravitational, electric, magnetic, and velocity potentials satisfy **Laplace equation**[5], which is (1.2.15) with $f(x, y) = 0$. By (1.2.14), a second order accurate approximation of Poisson's equation is given by

$$\begin{aligned}\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} &+ \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \\ &= \frac{1}{h^2}\big(u_{i,j+1} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} - 4u_{i,j}\big) = f_{i,j}.\end{aligned}$$

This corresponds to the "computational molecule"

$$\begin{bmatrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{bmatrix}$$

Review Questions

1. Make lists of the concepts and ideas which have been introduced. Review their use in the various types of problems mentioned.

[4] Siméon Denis Poisson (1781–1840).
[5] Pierre Simon, Marquis de Laplace (1749–1827).

2. Discuss the convergence condition and the rate of convergence of the method of iteration for solving $x = F(x)$.

3. What is the trapezoidal rule? What is said about the dependence of its error on the step length?

Problems and Computer Exercises

1. Calculate $\sqrt{10}$ to seven decimal places using the method in Example 1.2.1. Begin with $x_0 = 2$.

2. Consider $f(x) = x^3 - 2x - 5$. The cubic equation $f(x) = 0$ has been a standard test problem, since Newton used it in 1669 to demonstrate his method. By computing (say) $f(x)$ for $x = 1, 2, 3$, we see that $x = 2$ probably is a rather good initial guess. Iterate then by Newton's method until you trust that the result is correct to six decimal places.

3. The equation $x^3 - x = 0$ has three roots, $-1, 0, 1$. We shall study the behaviour of Newton's method on this equation, with the notations used in §1.2.2 and Fig. 1.2.3.

(a) What happens if $x_0 = 1/\sqrt{3}$? Show that x_n converges to 1 for any $x_0 > 1/\sqrt{3}$. What is the analogous result for convergence to -1?

(b) What happens if $x_0 = 1/\sqrt{5}$? Show that x_n converges to 0 for any $x_0 \in (-1/\sqrt{5},\ 1/\sqrt{5})$.

Hint: Show first that if $x_0 \in (0,\ 1/\sqrt{5})$ then $x_1 \in (-x_0, 0)$. What can then be said about x_2?

(c) Find, by a drawing (with paper and pencil), $\lim x_n$ if x_0 is a little less than $1/\sqrt{3}$. Find by computation $\lim x_n$ if $x_0 = 0.46$.

*(d) A complete discussion of the question in (c) is rather complicated, but there is an implicit recurrence relation that produces a decreasing sequence $\{a_1 = 1/\sqrt{3},\ a_2,\ a_3, \ldots\}$, by means of which you can easily find $\lim_{n\to\infty} x_n$ for any $x_0 \in (1/\sqrt{5},\ 1/\sqrt{3})$. Try to find this recurrence.

Answer: $a_i - f(a_i)/f'(a_i) = -a_{i-1}$; $\lim_{n\to\infty} x_n = (-1)^i$ if $x_0 \in (a_i, a_{i+1})$; $a_1 = 0.577$, $a_2 = 0.462$, $a_3 = 0.450$, $a_4 \approx \lim_{i\to\infty} a_i = 1/\sqrt{5} = 0.447$.

4. Calculate $\int_0^{1/2} e^x\,dx$

(a) to six decimals using the primitive function.

(b) with the trapezoidal rule, using step length $h = 1/4$.

(c) using Richardson extrapolation to $h = 0$ on the results using step length $h = 1/2$, and $h = 1/4$.

(d) Compute the ratio between the error in the result in (c) to that of (b).

5. In Example 1.2.4 we computed $y''(0.6)$ for $y = \cos(x)$, with step length $h = 0.01$. Make similar calculations using $h = 0.1$, $h = 0.05$ and $h = 0.001$. Which value of h gives the best result, using values of y to six decimal places? Discuss qualitatively the influences of both the rounding errors in the function values

and the error in the approximation of a derivative with a difference quotient on the result for various values of h.

1.3 Some Numerical Algorithms

For a given numerical problem one can consider many different algorithms. These can differ in efficiency and reliability and give approximate answers sometimes with widely varying accuracy. In the following we give a few examples of how algorithms can be developed to solve some typical numerical problems.

1.3.1 Recurrence Relations

One of the most important and interesting parts of the preparation of a problem for a computer is to *find a recursive description of the task*. Often an enormous amount of computation can be described by a small set of recurrence relations. Euler's method for the step-by-step solution of ordinary differential equations is an example. Other examples will be given in this section; see also problems at the end of this section.

A common computational task is the evaluation of a polynomial, at a given point x where, say,

$$p(x) = a_0x^3 + a_1x^2 + a_2x + a_3 = ((a_0x + a_1)x + a_2)x + a_3.$$

We set $b_0 = a_0$, and compute

$$b_1 = b_0x + a_1, \quad b_2 = b_1x + a_2, \quad p(x) = b_3 = b_2x + a_3.$$

This illustrates, for $n = 3$, **Horner's rule** for evaluating a polynomial of degree n,

$$p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n,$$

This algorithm can be described by the recurrence relation:

$$b_0 = a_0, \quad b_i = b_{i-1}x + a_i, \quad i = 1 : n, \tag{1.3.1}$$

where $p(x) = b_n$.

The quantities b_i in (1.3.1) are of intrinsic interest because of the following result, often called **synthetic division**:

$$\frac{p(x) - p(z)}{x - z} = \sum_{i=0}^{n-1} b_i x^{n-1-i}, \tag{1.3.2}$$

where the b_i are defined by (1.3.1). The proof of this result is left as an exercise. Synthetic division is used, for instance, in the solution of algebraic equations, when already computed roots are successively eliminated. After each elimination, one can deal with an equation of lower degree. This process is called **deflation**see Sec. 6.5.5. . (As shown in Sec. 6.6.4, some care is necessary in the numerical application of this idea.)

The proof of the following useful relation is left as an exercise to the reader:

Lemma 1.3.1.

Let the b_i be defined by (1.3.1) and

$$c_0 = b_0, \qquad c_i = b_i + zc_{i-1}, \quad i = 1 : n-1. \tag{1.3.3}$$

Then $p'(z) = c_{n-1}$.

Recurrence relations are among the most valuable aids in numerical calculation. Very extensive calculations can be specified in relatively short computer programs with the help of such formulas. However, unless used in the right way errors can grow exponentially and completely ruin the results.

Example 1.3.1.

To compute the integrals $I_n = \int_0^1 \frac{x^n}{x+5}\,dx$, $i = 1 : N$ one can use the recurrence relation

$$I_n + 5I_{n-1} = 1/n, \tag{1.3.4}$$

which follows from

$$I_n + 5I_{n-1} = \int_0^1 \frac{x^n + 5x^{n-1}}{x+5}\,dx = \int_0^1 x^{n-1}\,dx = \frac{1}{n}.$$

Below we use this formula to compute I_8, using six decimals throughout. For $n = 0$ we have

$$I_0 = [\ln(x+5)]_0^1 = \ln 6 - \ln 5 = 0.182322.$$

Using the recurrence relation we get

$$\begin{aligned}
I_1 &= 1 - 5I_0 = 1 - 0.911610 = 0.088390,\\
I_2 &= 1/2 - 5I_1 = 0.500000 - 0.441950 = 0.058050,\\
I_3 &= 1/3 - 5I_2 = 0.333333 - 0.290250 = 0.043083,\\
I_4 &= 1/4 - 5I_3 = 0.250000 - 0.215415 = 0.034585,\\
I_5 &= 1/5 - 5I_4 = 0.200000 - 0.172925 = 0.027075,\\
I_6 &= 1/6 - 5I_5 = 0.166667 - 0.135375 = 0.031292,\\
I_7 &= 1/7 - 5I_6 = 0.142857 - 0.156460 = -0.013603.
\end{aligned}$$

It is strange that $I_6 > I_5$, and obviously absurd that $I_7 < 0$! The reason for the absurd result is that the round-off error ϵ in $I_0 = 0.18232156\ldots$, whose magnitude is about $0.44 \cdot 10^{-6}$ is *multiplied* by (-5) in the calculation of I_1, which then has an error of -5ϵ. That error produces an error in I_2 of $5^2\epsilon$, etc. Thus the magnitude of the error in I_7 is $5^7\epsilon = 0.0391$, which is larger than the true value of I_7. On top of this comes the round-off errors committed in the various steps of the calculation. These can be shown in this case to be relatively unimportant.

If one uses higher precision, the absurd result will show up at a later stage. For example, a computer that works with a precision corresponding to about 16

decimal places, gave a negative value to I_{22} although I_0 had full accuracy. The above algorithm is an example of a disagreeable phenomenon, called **numerical instability**.

We now show how, in this case, one can avoid numerical instability by choosing a more suitable algorithm.

Example 1.3.2.

We shall here use the recurrence relation in the other direction,

$$I_{n-1} = (1/n - I_n)/5. \tag{1.3.5}$$

Now the errors will be *divided* by -5 in each step. But we need a starting value. We can directly see from the definition that I_n decreases as n increases. One can also surmise that I_n decreases slowly when n is large (the reader is recommended to motivate this). Thus we try setting $I_{12} = I_{11}$. It then follows that

$$I_{11} + 5I_{11} \approx 1/12, \qquad I_{11} \approx 1/72 \approx 0.013889.$$

(show that $0 < I_{12} < 1/72 < I_{11}$). Using the recurrence relation we get

$$I_{10} = (1/11 - 0.013889)/5 = 0.015404, \qquad I_9 = (1/10 - 0.015404)/5 = 0.016919,$$

and further

$$\begin{aligned} &I_8 = 0.018838, \quad I_7 = 0.021232, \quad I_6 = 0.024325, \quad I_5 = 0.028468, \\ &I_4 = 0.034306, \quad I_3 = 0.043139, \quad I_2 = 0.058039, \quad I_1 = 0.088392, \end{aligned}$$

and finally $I_0 = 0.182322$. Correct!

If we instead simply take as starting value $I_{12} = 0$, one gets $I_{11} = 0.016667$, $I_{10} = 0.018889$, $I_9 = 0,016222$, $I_8 = 0.018978$, $I_7 = 0.021204$, $I_6 = 0.024331$, and $I_5, \ldots, I_0$ have the same values as above. The difference in the values for I_{11} is 0.002778. The subsequent values of $I_{10}, I_9, \ldots, I_0$ are quite close *because the error is divided by -5 in each step.* The results for I_n obtained above have errors which are less than 10^{-3} for $n \leq 8$.

The reader is warned, however, not to draw erroneous conclusions from the above example. The use of a recurrence relation "backwards" is not a universal recipe as will be seen later on! Compare also Problems 6 and 7 at the end of this section.

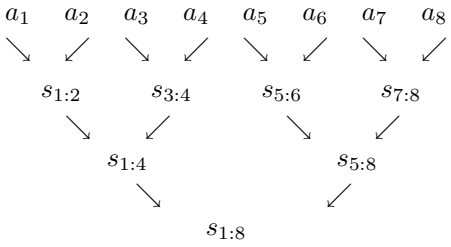
1.3.2 Divide and Conquer Strategy

A powerful strategy for solving large scale problems is the **divide and conquer** strategy. The idea is to split a high dimensional problem into problems of lower dimension. Each of these are then again split into smaller subproblems, etc., until a number of sufficiently small problems are obtained. The solution of the initial problem is then obtained by combining the solution of the subproblems working backwards in the hierarchy.

We illustrate the idea on the computation of the sum $s = \sum_{i=1}^{n} a_i$. The usual way to proceed is to use the recursion

$$s_0 = 0, \qquad s_i = s_{i-1} + a_i, \quad i = 1 : n.$$

Another order of summation is as illustrated below for $n = 2^3 = 8$:



where $s_{i,j} = a_i + \cdots + a_j$. In this table each new entry is obtained by adding its two neighbors in the row above. Clearly this can be generalized to compute an arbitrary sum of $n = 2^k$ terms in k steps. In the first step we perform $n/2$ sums of two terms, then $n/4$ partial sums each of 4 terms, etc., until in the kth step we compute the final sum.

This summation algorithm uses the same number of additions as the first one. However, it has the advantage that it splits the task in *several subtasks that can be performed in parallel*. For large values of n this summation order can also be much more accurate than the conventional order (see Problem 2.3.5, Chapter 2). Espelid [9] gives an interesting discussion of such summation algorithms.

The algorithm can also be described in another way. Consider the following definition of a summation algorithm for computing the $s(i, j) = a_i + \cdots + a_j$, $j > i$:

$sum = s(i, j)$;
if $j = i + 1$ **then** $sum = a_i + a_j$;
  else $k = \lfloor (i + j)/2 \rfloor$; $sum = s(i, k) + s(k + 1, j)$;
end

This function defines $s(i, j)$ in a recursive way; if the sum consists of only two terms then we add them and return with the answer. Otherwise we split the sum in two and use the function again to evaluate the corresponding two partial sums. This approach is aptly called the divide and conquer strategy. The function above is an example of a **recursive algorithm**—it calls itself. Many computer languages (e.g., MATLAB) allow the definition of such recursive algorithms. The divide and conquer is a **top down** description of the algorithm in contrast to the **bottom up** description we gave first.

There are many other less trivial examples of the power of the divide and conquer approach. It underlies the Fast Fourier Transform and leads to efficient implementations of, for example, matrix multiplication, Cholesky factorization, and other matrix factorizations. Interest in such implementations have increased lately since it has been realized that they achieve very efficient automatic parallelization of many tasks.

1.3.3 Approximation of Functions

Many important function in applied mathematics cannot be expressed in finite terms of elementary functions, and must be approximated by numerical methods. Examples from statistics are the normal probability function, the chi-square distribution function, the exponential integral, and the Poisson distribution. These can, by simple transformations, be brought to particular cases of the **incomplete gamma function**

$$\gamma(a,z) = \int_0^z e^{-t}t^{a-1}dt, \quad \Re a > 0, \tag{1.3.6}$$

A collection of formulas that can be used to evaluate this function is found in Abramowitz and Stegun [1, Sec. 6.5]. Codes and some theoretical background are given in Numerical Recipes [34, Sec. 6.2–6.3].

Example 1.3.3.

As a simple example we consider evaluating the **error function** defined by

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt, \tag{1.3.7}$$

for $x \in [0,1]$. This function is encountered in computing the distribution function of a normal deviate. It takes the values $\operatorname{erf}(0) = 0$, $\operatorname{erf}(\infty) = 1$, and is related to the incomplete gamma functions by $\operatorname{erf}(x) = \gamma(1/2, x^2)$.

In order to compute $\operatorname{erf}(x)$ for $x \in [0,1]$ with a relative error less than 10^{-8} with a small number of arithmetic operations, the function can be approximated by a power series. Setting $z = -t^2$ in the well known Maclaurin series for e^z, truncating after $n+1$ terms and integrating term by term we obtain the approximation

$$erf(x) \approx \frac{2}{\sqrt{\pi}} \int_0^x \sum_{j=0}^{n} (-1)^j \frac{t^{2j}}{j!}\, dt = \frac{2}{\sqrt{\pi}} \sum_{j=0}^{n} a_j x^{2j+1}, \tag{1.3.8}$$

where

$$a_0 = 1, \quad a_j = \frac{(-1)^j}{j!(2j+1)}.$$

(Note that $\operatorname{erf}(x)$ is a odd function of x.) This series converges for all x, but is suitable for numerical computations only for values of x which are not too large. To evaluate the series we note that the coefficients a_j satisfies the recurrence relation

$$a_j = -a_{j-1}\frac{(2j-1)}{j(2j+1)}, \quad j > 0.$$

This recursion shows that for $x \in [0,1]$ the absolute values of the terms $t_j = a_j x^{2j+1}$ decrease monotonically. This implies that the absolute error in a partial sum is bounded by the absolute value of the first neglected term. (Why? For an answer see Theorem 3.1.5 in Chapter 3.)

A possible algorithm for evaluating the sum in (1.3.8) is then:

Set $s_0 = t_0 = x$; for $j = 1, 2, \ldots$ compute

$$t_j = -t_{j-1}\frac{(2j-1)}{j(2j+1)}x^2, \quad s_j = s_{j-1} + t_j, \quad \text{until} \quad |t_j| \leq \cdot 10^{-8} s_j.$$

Here we have estimated the error by the last term added in the series. Since we have to compute this term for the error estimate we might as well use it! Note also that in this case, where the number of terms is fixed in advance, Horner's scheme is not suitable for the evaluation. Fig. 1.3.1 shows the graph of the relative error

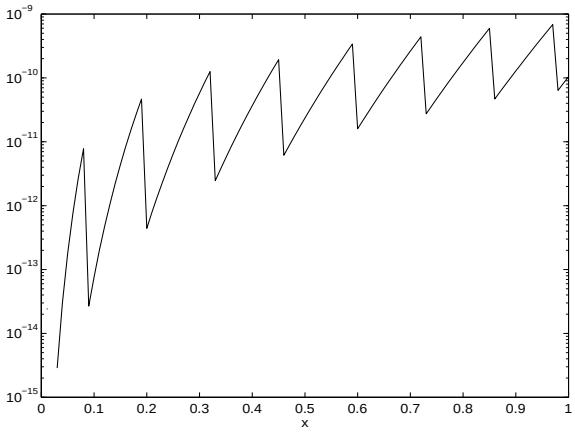


Figure 1.3.1. *Relative error* $e(x) = |p_{2n+1}(x) - \text{erf}(x)|/\text{erf}(x)$.

in the computed approximation $p_{2n+1}(x)$. At most twelve terms in the series were needed.

In the above example there are no errors in measurement, but the "model" of approximating the error function with a polynomial is not exact, since the function demonstrably is not a polynomial. There is a **truncation error**[6] from truncating the series, which can in this case be made as small as one wants by choosing the degree of the polynomial sufficiently large (e.g., by taking more terms in the Maclaurin series).

The use of power series and rational approximations will be studied in depth in Chapter 3, where also other more efficient methods than the Maclaurin series for approximation by polynomials will be treated.

A different approximation problem, which occurs in many variants, is to approximate a function f by a member f^* of a class of functions which is easy to work with mathematically (e.g., polynomials, rational functions, or trigonometric polynomials), where each particular function in the class is specified by the numerical values of a number of parameters.

[6] In general the error due to replacing an infinite process by a finite is referred to as a truncation error.

In computer aided design (CAD) curves and surfaces have to be represented mathematically, so that they can be manipulated and visualized easily. Important applications occur in aircraft and automotive industries. For this purpose **spline functions** are now used extensively. The name **spline** comes from a very old technique in drawing smooth curves, in which a thin strip of wood, called a draftsman's spline, is bent so that it passes trough a given set of points. The points of interpolation are called **knots** and the spline is secured at the knots by means of lead weights called **ducks**. Before the computer age splines were used in ship building and other engineering designs.

Bézier curves, which can also be used for these purposes, were developed in 1962 by Bézier and de Casteljau, when working for the French car companies Renault and Citroën,

1.3.4 The Principle of Least Squares

In many applications a linear mathematical model is to be fitted to given observations. For example, consider a model described by a scalar function $y(t) = f(x,t)$, where $x \in \mathbf{R}^n$ is a parameter vector to be determined from measurements (y_i, t_i), $i = 1 : m$. There are two types of shortcomings to take into account: errors in the input data, and shortcomings in the particular model (class of functions, form), which one intends to adopt to the input data. For ease in discussion. We shall call these **measurement errors** and **errors in the model**, respectively.

In order to reduce the influence of measurement errors in the observations one would like to use a greater number of measurements than the number of unknown parameters in the model. If $f(x,t)$ be *linear* in x and of the form

$$f(x,t) = \sum_{j=1}^{n} x_j \phi_j(t).$$

Then the equations

$$y_i = \sum_{j=1}^{n} x_j \phi_j(t_i), \quad i = 1 : m,$$

form an overdetermined linear system $Ax = b$, where $a_{ij} = \phi_j(t_i)$ and $b_i = y_i$. The resulting problem is then to "solve" an **overdetermined** linear system of equations $Ax = b$. where $b \in \mathbf{R}^m$, $A \in \mathbf{R}^{m\times n}$ $(m > n)$. Thus we want to find a vector $x \in \mathbf{R}^n$ such that Ax is the "best" approximation to b. We refer in the following to $r = b - Ax$ as the **residual vector**.

There are many possible ways of defining the "best" solution. A choice which can often be motivated for statistical reasons and which also leads to a simple computational problem is to take as solution a vector x, which minimizes the sum of the squared residuals, i.e.

$$\min_{x\in\mathbf{R}^n} \sum_{i=1}^{m} r_i^2, \tag{1.3.9}$$

The principle of least squares for solving an overdetermined linear system was first used by Gauss, who in 1801 used it to successively predicted the orbit of the as-

teroid Ceres. It can shown that the **least squares solution** satisfies the **normal equations**

$$A^T Ax = A^T b. \tag{1.3.10}$$

The matrix $A^T A$ is symmetric and can be shown to be nonsingular if A has linearly independent columns, in which case $Ax = b$ has a *unique* least squares solution.

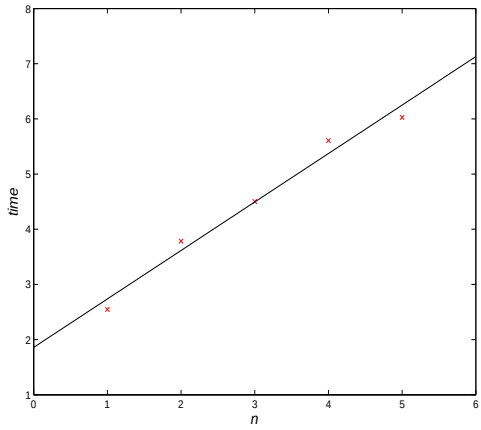


Figure 1.3.2. *Fitting a linear relation to observations.*

Example 1.3.4.

The points in Fig. 1.3.2 show for $n = 1 : 5$, the time t_n, for the nth passage of a swinging pendulum through its point of equilibrium. The condition of the experiment were such that a linear relation of the form $t = a + b\,n$ can be assumed to be valid. Random errors in measurement are the dominant cause of the deviation from linearity shown in Fig. 1.3.2. This deviation causes the values of the parameters a and b to be uncertain. The least squares fit to the model, shown by the straight line in Fig 1.3.2, minimizes the sum of squares of the deviations $\sum_{n=1}^{5}(a+b\,n-t_n)^2$.

Example 1.3.5.

The recently discovered comet 1968 Tentax is supposed to move within the solar system. The following observations of its position in a certain polar coordinate system have been made

r	2.70	2.00	1.61	1.20	1.02
ϕ	48°	67°	83°	108°	126°

By Kepler's first law the comet should move in a plane orbit of elliptic or hyperbolic form, if the perturbations from planets are neglected. Then the coordinates satisfy

$$r = p/(1 - e\cos\phi),$$

where p is a parameter and e the eccentricity. We want to estimate p and e by the method of least squares from the given observations.

We first note that if the relationship is rewritten as

$$1/p - (e/p)\cos\phi = 1/r,$$

it becomes linear in the parameters $x_1 = 1/p$ and $X_2 = e/p$. We then get the linear system $Ax = b$, where

$$A = \begin{pmatrix} 1.0000 & -0.6691 \\ 1.0000 & -0.3907 \\ 1.0000 & -0.1219 \\ 1.0000 & 0.3090 \\ 1.0000 & 0.5878 \end{pmatrix}, \qquad b = \begin{pmatrix} 0.3704 \\ 0.5000 \\ 0.6211 \\ 0.8333 \\ 0.9804 \end{pmatrix}.$$

The least squares solution is $x = (\,0.6886 \quad 0.4839\,)^T$ giving $p = 1/x_1 = 1.4522$ and finally $e = px_2 = 0.7027$.

In practice, both the measurements and the model are as a rule insufficient. One can also see approximation problems as analogous to the task of a communication engineer, to filter away *noise* from the *signal*. These questions are connected with both Mathematical Statistics and the mathematical discipline Approximation Theory.

Review Questions

1. Describe Horner's rule and synthetic division.
2. Give a concise explanation why the algorithm in Example 1.3.1 did not work and why that in Example 1.3.2 did work.
3. Describe the idea behind the divide and conquer strategy. What is a main advantage of this strategy? How do you apply it to the task of summing n numbers?
4. Describe the least squares principle for solving an overdetermined linear system.

Problems and Computer Exercises

1. (a) Use Horner's scheme to compute $p(2)$ where

$$p(x) = x^4 + 2x^3 - 3x^2 + 2.$$

(b) Count the number of multiplications and additions required for the evaluation of a polynomial $p(z)$ of degree n by Horner's rule. Compare with the work needed when the powers are calculated recursively by $x^i = x \cdot x^{i-1}$ and subsequently multiplied by a_{n-i}.

2. Show how repeated synthetic division can be used to move the origin of a polynomial, i.e., given $a_1,\ a_2, \ldots, a_n$ and z, find $c_1,\ c_2, \ldots, c_n$ so that

$$p_n(x) = \sum_{j=1}^{n} a_j x^{j-1} \equiv \sum_{j=1}^{n} c_j (x-z)^{j-1}.$$

Write a program for synthetic division (with this ordering of the coefficients), and apply it to this algorithm.

Hint: Apply synthetic division to $p_n(x)$, $p_{n-1}(x) = (p_n(x) - p_n(z))/(x-z)$, etc.

3. (a) Show that the transformation made in Problem 2 can also be expressed by means of the matrix-vector equation,

$$c = \mathrm{diag}(z^{1-i})\, P\, \mathrm{diag}(z^{j-1})\, a,$$

where $a = [a_1, a_2, \ldots a_n]^T$, $c = [c_1, c_2, \ldots c_n]^T$, and $\mathrm{diag}(z^{j-1})$ is a diagonal matrix with the elements z^{j-1}, $j = 1:n$. The matrix $P \in \mathbf{R}^{n\times n}$ has elements $p_{i,j} = \binom{j-1}{i-1}$, if $j \geq i$, else $p_{i,j} = 0$. By convention, $\binom{0}{0} = 1$ here.

(b) Note the relation of P to the Pascal triangle, and show how P can be generated by a simple recursion formula. Also show how each element of P^{-1} can be expressed in terms of the corresponding element of P. How is the origin of the polynomial $p_n(x)$ moved, if you replace P by P^{-1} in the matrix-vector equation that defines c?

(c) If you reverse the order of the elements of the vectors $a,\ c$—this may sometimes be a more convenient ordering—how is the matrix P changed?

Comment: With a terminology to be used much in this book (see Sec. 4.1.2), we can look upon a and c as different coordinate vectors for the same element in the n-dimensional linear space $\mathcal{P}_n$ of polynomials of degree *less than* n. The matrix P gives the coordinate transformation.

4. Derive recurrence relations and write a program for computing the coefficients of the *product* r of two polynomials p and q,

$$r(x) = p(x)q(x) = \left(\sum_{i=1}^{m} a_i x^{i-1}\right)\left(\sum_{j=1}^{n} b_j x^{j-1}\right) = \sum_{k=1}^{m+n-1} c_k x^{k-1}.$$

5. Let x, y be nonnegative integers, with $y \neq 0$. The division x/y yields the quotient q and the remainder r. Show that if x and y have a common factor, then that number is a divisor of r as well. Use this remark to design an algorithm for the determination of the greatest common divisor of x and y (*Euclid's algorithm*).

6. Derive a forward and a backward recurrence relation for calculating the integrals

$$I_n = \int_0^1 \frac{x^n}{4x+1}\, dx.$$

Why is in this case the forward recurrence stable and the backward recurrence unstable?

7. (a) Solve Example 1.3.1 on a computer, with the following changes: Start the recursion (1.3.4) with $I_0 = \ln 1.2$, and compute and print the sequence $\{I_n\}$ until I_n for the first time becomes negative.

(b) Start the recursion (1.3.5) first with the condition $I_{19} = I_{20}$, then with $I_{29} = I_{30}$. Compare the results you obtain and assess their approximate accuracy. Compare also with the results of 7 (a).

***8.** (a) Write a program (or study some library program) for finding the quotient $Q(x)$ and the remainder $R(x)$ of two polynomials $A(x), B(x)$, i.e., $A(x) = Q(x)B(x) + R(x)$, $\deg R(x) < \deg B(x)$.

(b) Write a program (or study some library program) for finding the coefficients of a polynomial with given roots.

***9.** (a) Write a program (or study some library program) for finding the greatest common divisor of two *polynomials*. Test it on a number of polynomials of your own choice. Choose also some polynomials of a rather high degree, and do not only choose polynomials with small integer coefficients. Even if you have constructed the polynomials so that they should have a common divisor, rounding errors may disturb this, and some tolerance is needed in the decision whether a remainder is zero or not. One way of finding a suitable size of the tolerance is to make one or several runs where the coefficients are subject to some small random perturbations, and find out how much the results are changed.

(b) Apply the programs mentioned in the last two problems for finding and eliminating multiple zeros of a polynomial.

Hint: A multiple zero of a polynomial is a common zero of the polynomial and its derivative.

10. It is well known that $\operatorname{erf}(x) \to 1$ as $x \to \infty$. If $x \gg 1$ the relative accuracy of the complement $1 - \operatorname{erf}(x)$ is of interest. However, the series expansion used in Example 1.3.3 for $x \in [0, 1]$ is not suitable for large values of x. Why?

Hint: Derive an approximate expression for the largest term.

1.4 Matrix Computations

Matrix computations are ubiquitous in Scientific Computing. A survey of basic notations and concepts in matrix computations and linear vector spaces is given in Appendix A. This is needed for several topics treated in later chapters of this first volume. A fuller treatment of this topic will be given in Vol. II.

In this section we focus on some important developments since the 1950s in the solution of linear systems. One is the systematic use of matrix notations and the interpretation of Gaussian elimination as matrix factorization. This **decompositional approach** has several advantages, e.g, a computed factorization can often be used with great saving to solve new problems involving the original matrix. Another is the rapid developments of sophisticated iterative methods, which are becoming increasingly important as the size of systems increase.

1.4.1 Matrix Multiplication

A **matrix** A is a collection of $m \times n$ numbers ordered in m rows and n columns

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}.$$

We write $A \in \mathbf{R}^{m\times n}$, where $\mathbf{R}^{m\times n}$ denotes the set of all real $m \times n$ matrices. If $m = n$, then the matrix A is said to be square and of order n. If $m \neq n$, then A is said to be rectangular.

The **product** of two matrices A and B is defined if and only if the number of columns in A equals the number of rows in B. If $A \in \mathbf{R}^{m\times n}$ and $B \in \mathbf{R}^{n\times p}$ then

$$C = AB \in \mathbf{R}^{m\times p}, \qquad c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}. \tag{1.4.1}$$

It is often useful to think of a matrix as being built up of blocks of lower dimensions. The great convenience of this lies in the fact that the operations of addition and multiplication can be performed by treating the blocks as *non-commuting scalars* and applying the definition (1.4.1). Of course the dimensions of the blocks must correspond in such a way that the operations can be performed.

Example 1.4.1.

Assume that the two $n \times n$ matrices are partitioned into 2×2 block form

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

where A_{11} and B_{11} are square matrices of the same dimension. Then the product $C = AB$ equals

$$C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}. \tag{1.4.2}$$

Be careful to note that since matrix multiplication is not commutative the *order of the factors in the products cannot be changed*! In the special case of block upper triangular matrices this reduces to

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} = \begin{pmatrix} R_{11}S_{11} & R_{11}S_{12} + R_{12}S_{22} \\ 0 & R_{22}S_{22} \end{pmatrix}. \tag{1.4.3}$$

Note that the product is again block upper triangular and its block diagonal simply equals the products of the diagonal blocks of the factors.

It is important to know roughly how much work is required by different matrix algorithms. By inspection of (1.4.1) it is seen that computing the mp elements c_{ij} requires mnp additions and multiplications.

In matrix computations the number of multiplicative operations $(\times, /)$ is usually about the same as the number of additive operations $(+, -)$. Therefore, in older literature, a **flop** was defined to mean roughly the amount of work associated with the computation

$$s := s + a_{ik}b_{kj},$$

i.e., one addition *and* one multiplication (or division). In more recent textbooks (e.g., Golub and Van Loan [14, 1996]) a flop is defined as one floating point operation doubling the older flop counts.[7] Hence, multiplication $C = AB$ of two two square matrices of order n requires $2n^3$ flops. The matrix-vector multiplication $y = Ax$, where $x \in \mathbf{R}^{n\times 1}$ requires $2mn$ flops.

Operation counts are meant only as a rough appraisal of the work and one should not assign too much meaning to their precise value. On modern computer architectures the rate of transfer of data between different levels of memory often limits the actual performance. Also ignored here is the fact that on current computers division usually is 5–10 times slower than a multiply.

However, an operation count still provides useful information, and can serve as an initial basis of comparison of different algorithms. For example, it tells us that the running time for multiplying two square matrices on a computer roughly will increase cubically with the dimension n. Thus, doubling n will approximately increase the work by a factor of eight; cf. (1.4.2).

An intriguing question is whether it is possible to multiply two matrices $A, B \in \mathbf{R}^{n\times n}$ (or solve a linear system of order n) in less than n^3 (scalar) multiplications. The answer is yes! Strassen [38] developed a fast algorithm for matrix multiplication, which, if used recursively to multiply two square matrices of dimension $n = 2^k$, reduces the number of multiplications from n^3 to $n^{\log_2 7} = n^{2.807\ldots}$.

1.4.2 Solving Triangular Systems

The solution of **linear systems of equations** is one of the most frequently encountered problems in scientific computing. One important source of linear systems is discrete approximations of continuous differential and integral equations.

A linear system can be written in matrix-vector form as

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}, \tag{1.4.4}$$

where a_{ij} and b_i, $1 \le i \le m$, $1 \le j \le n$ be the known input data and the task is to compute the unknown variables x_j, $1 \le j \le n$. More compactly $Ax = b$, where $A \in \mathbf{R}^{m\times n}$ is a matrix and $x \in \mathbf{R}^n$ and $b \in \mathbf{R}^m$ are column vectors. If A is square and nonsingular there is an inverse matrix A^{-1} such that $A^{-1}A = AA^{-1} = I$, the identity matrix. The solution to (1.4.4) can then be written as $x = A^{-1}b$, but *in almost all cases one should avoid computing the inverse* A^{-1}.

[7] Stewart [p. 96][36] uses **flam** (floating point addition and multiplication) to denote an "old" flop.

Linear systems which (possibly after a permutation of rows and columns of A) are of triangular form are particularly simple to solve. Consider a square **upper triangular** linear system ($m = n$)

$$\begin{pmatrix} u_{11} & \cdots & u_{1,n-1} & u_{1n} \\ & \ddots & \vdots & \vdots \\ & & u_{n-1,n-1} & u_{n-1,n} \\ & & & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_{n-1} \\ b_n \end{pmatrix}.$$

The matrix U is nonsingular if and only if

$$\det(U) = u_{11} \cdots u_{n-1,n-1} u_{nn} \neq 0.$$

If this is the case the unknowns can be computed by the following recursion

$$x_n = b_n/u_{nn}, \qquad x_i = \Big(b_i - \sum_{k=i+1}^{n} u_{ik} x_k\Big)/u_{ii}, \quad i = n-1, \ldots, 1. \tag{1.4.5}$$

It follows that the solution of a triangular system of order n can be computed in about n^2 flops. Note that this is the same amount of work as required for *multiplying* a vector by a triangular matrix.

Since the unknowns are solved for in *backward* order, this is called **back-substitution**. Similarly, a square linear system of **lower triangular** form $Lx = b$,

$$\begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

where L is nonsingular, can be solved by **forward-substitution**

$$x_1 = b_1/l_{11}, \qquad x_i = \Big(b_i - \sum_{k=1}^{i-1} l_{ik} x_k\Big)/l_{ii}, \quad i = 2 : n. \tag{1.4.6}$$

(Note that by reversing the order of the rows and columns an upper triangular system is transformed into a lower triangular and vice versa.)

When implementing a matrix algorithm on a computer, the *order of operations* in matrix algorithms may be important. One reason for this is the economizing of storage, since even matrices of moderate dimensions have a large number of elements. When the initial data is not needed for future use, computed quantities may overwrite data. To resolve such ambiguities in the description of matrix algorithms it is important to be able to describe computations like those in equations (1.4.5) in a more precise form. For this purpose we will use an informal programming language, which is sufficiently precise for our purpose but allows the suppression of cumbersome details. We illustrate these concepts on the back-substitution algorithm given above. In the following back-substitution algorithm the solution x overwrites the data b.

Algorithm 1.4.1 Back-substitution

Given a nonsingular upper triangular matrix $U \in \mathbf{R}^{n\times n}$ and a vector $b \in \mathbf{R}^n$, the following algorithm computes $x \in \mathbf{R}^n$ such that $Ux = b$:

$$\begin{aligned}
&\textbf{for } i = n : (-1) : 1 \\
&\quad s := \sum_{j=i+1}^{n} u_{ik} b_k; \\
&\quad b_i := (b_i - s)/u_{ii}; \\
&\textbf{end}
\end{aligned}$$

Here $x := y$ means that the value of y is evaluated and assigned to x. We use the convention that when the upper limit in a sum is smaller than the lower limit the sum is set to zero.

Another possible sequencing of the operations in Algorithm 1.3.1 is the following:

$$\begin{aligned}
&\textbf{for } k = n : (-1) : 1 \\
&\quad b_k := b_k/u_{kk}; \\
&\quad \textbf{for } i = k-1 : (-1) : 1 \\
&\qquad b_i := b_i - u_{ik} b_k; \\
&\quad \textbf{end} \\
&\textbf{end}
\end{aligned}$$

Here the elements in U are accessed column-wise instead of row-wise as in the previous algorithm. Such differences can influence the efficiency of the implementation depending on how the elements in the matrix U are stored.

1.4.3 Gaussian Elimination

Gaussian elimination[8] is taught already in elementary courses in linear algebra. However, although the theory is deceptively simple the practical solution of large linear systems is far from trivial. In the beginning of the computer age in 1940s there was a mood of pessimism about the possibility of accurately solving systems even of modest order, say $n = 100$. Today there is a much deeper understanding of how Gaussian elimination performs in finite precision arithmetic and linear systems with hundred of thousands unknowns are routinely solved in scientific computing!

Clearly the following elementary operation can be performed on the system without changing the set of solutions:

- Interchanging two equations
- Multiplying an equation by a nonzero scalar α.

[8] Named after Carl Friedrich Gauss (1777–1855), but known already in China as early as in the first century BC.

- Adding a multiple α of the ith equation to the jth equation.

These operations correspond in an obvious way to row operations carried out on the augmented matrix (A, b). By performing a sequence of such elementary operations one can always transform the system $Ax = b$ into a simpler system, which can be trivially solved.

In the most important direct method Gaussian elimination the unknowns are eliminated in a systematic way, so that at the end an equivalent triangular system is produced, which can be solved by substitution. Consider the system (1.4.4) with $m = n$ and assume that $a_{11} \neq 0$. Then we can eliminate x_1 from the last $(n-1)$ equations as follows. Subtracting from the ith equation the multiple

$$l_{i1} = a_{i1}/a_{11}, \quad i = 2 : n,$$

of the first equation, the last $(n-1)$ equations become

$$\begin{pmatrix} a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \ddots & \vdots \\ a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix} \begin{pmatrix} x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix},$$

where the new elements are given by

$$a_{ij}^{(2)} = a_{ij} - l_{i1}a_{1j}, \qquad b_i^{(2)} = b_i - l_{i1}b_1, \quad i = 2 : n.$$

This is a system of $(n-1)$ equations in the $(n-1)$ unknowns $x_2, \ldots, x_n$. If $a_{22}^{(2)} \neq 0$, we can proceed and in the next step eliminate x_2 from the last $(n-2)$ of these equations. This gives a system of equations containing only the unknowns $x_3, \ldots, x_n$. We take

$$l_{i2} = a_{i2}^{(2)}/a_{22}^{(2)}, \quad i = 3 : n,$$

and the elements of the new system are given by

$$a_{ij}^{(3)} = a_{ij}^{(2)} - l_{i2}a_{2j}^{(2)}, \qquad b_i^{(3)} = b_i^{(2)} - l_{i2}b_2^{(2)}, \quad i = 3 : n.$$

The diagonal elements $a_{11}, a_{22}^{(2)}, a_{33}^{(3)}, \ldots$, which appear during the elimination are called **pivotal elements**. As long as these are nonzero, the elimination can be continued. After $(n-1)$ steps we get the single equation

$$a_{nn}^{(n)} x_n = b_n^{(n)}.$$

Collecting the first equation from each step we get

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(n)} \end{pmatrix}, \tag{1.4.7}$$

where we have introduced the notations $a_{ij}^{(1)} = a_{ij}$, $b_i^{(1)} = b_i$ for the coefficients in the original system. Thus, we have reduced (1.4.4) to an equivalent nonsingular, upper triangular system (1.4.7), which can be solved by back-substitution. In passing we remark that the determinant of a matrix A, defined in (A.2.4), does not change under row operations we have from (1.4.7)

$$\det(A) = a_{11}^{(1)} a_{22}^{(2)} \cdots a_{nn}^{(n)} \tag{1.4.8}$$

Gaussian elimination is indeed in general the most efficient method for **computing determinants**!

Algorithm 1.4.2 Gaussian Elimination (without row interchanges)

Given a matrix $A = A^{(1)} \in \mathbf{R}^{n\times n}$ and a vector $b = b^{(1)} \in \mathbf{R}^n$, the following algorithm computes the elements of the reduced system of upper triangular form (1.4.7). It is assumed that $a_{kk}^{(k)} \neq 0$, $k = 1 : n$:

for $k = 1 : n-1$
  for $i = k+1 : n$
    $l_{ik} := a_{ik}^{(k)} / a_{kk}^{(k)}$; $\quad a_{ik}^{(k+1)} := 0$;
    for $j = k+1 : n$
      $a_{ij}^{(k+1)} := a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$;
    end
    $b_i^{(k+1)} := b_i^{(k)} - l_{ik} b_k^{(k)}$;
  end
end

We remark that no extra memory space is needed to store the multipliers. When $l_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$ is computed the element $a_{ik}^{(k+1)}$ becomes equal to zero, so the multipliers can be stored in the lower triangular part of the matrix. Note also that if the multipliers l_{ik} are saved, then the operations on the vector b can be carried out at a later stage. This observation is important in that it shows that *when solving a sequence of linear systems*

$$Ax_i = b_i, \quad i = 1 : p,$$

with the same matrix A but different right hand sides the operations on A only have to be carried out once.

If we form the matrices

$$L = \begin{pmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \dots & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n)} \end{pmatrix} \tag{1.4.9}$$

then it can be shown that we have $A = LU$. Hence Gaussian elimination provides *a factorization of the matrix A into a lower triangular matrix L and an upper triangular matrix U*. This interpretation of Gaussian elimination has turned out to be extremely fruitful. For example, it immediately follows that the inverse of A (if it exists) has the factorization

$$A^{-1} = (LU)^{-1} = U^{-1}L^{-1}.$$

This shows that the solution of linear system $Ax = b$,

$$x = A^{-1}b = U^{-1}(L^{-1}b),$$

can be computed by solving the two triangular systems $Ly = b$, $Ux = y$. Indeed it has been said (G. E. Forsythe and C. B. Moler [12]) that

"*almost anything you can do with A^{-1} can be done without it*"

Several other important matrix factorizations will be studied at length in Volume II.

From Algorithm 1.3.2 it follows that $(n-k)$ divisions and $(n-k)^2$ multiplications and additions are used in step k to transform the elements of A. A further $(n-k)$ multiplications and additions are used to transform the elements of b. Summing over k and neglecting low order terms we find that the total number of flops required for the reduction of $Ax = b$ to a triangular system by Gaussian elimination is

$$\sum_{k=1}^{n-1} 2(n-k)^2 \approx 2n^3/3,$$

for the LU factorization of$A$ and

$$\sum_{k=1}^{n-1} 2(n-k) \approx n^2,$$

for each right hand side vector b. Comparing this with the n^2 flops needed to solve a triangular system we conclude that, except for very small values of n, *the LU factorization of A dominates the work in solving a linear system.* If several linear systems with the same matrix A but different right-hand sides are to be solved, then the factorization needs to be performed only once!

Example 1.4.2. Many applications give rise to linear systems where the matrix A only has a few nonzero elements close to the main diagonal. Such matrices are called **band matrices**. An important example is, banded matrices of the form

$$A = \begin{pmatrix} b_1 & c_1 & & & \\ a_1 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & a_{n-1} & b_n \end{pmatrix}, \tag{1.4.10}$$

which are called **tridiagonal**. Tridiagonal systems of linear equations can be solved by Gaussian elimination with much less work than the general case. The following algorithm solves the tridiagonal system $Ax = g$ by Gaussian elimination without pivoting.

First compute the LU factorization $A = LU$, where

$$L = \begin{pmatrix} 1 & & & & \\ \gamma_1 & 1 & & & \\ & \gamma_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & \gamma_{n-1} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} \beta_1 & c_1 & & & \\ & \beta_2 & c_2 & & \\ & & \ddots & \ddots & \\ & & & \beta_{n-1} & c_{n-1} \\ & & & & \beta_n \end{pmatrix}.$$

The new elements in L nd U are obtained from the recursion: Set $\beta_1 = b_1$, and

$$\gamma_k = a_k/\beta_k, \qquad \beta_{k+1} = b_{k+1} - \gamma_k c_k, \quad k = 1 : n-1. \tag{1.4.11}$$

(Check this by computing the product LU!) The solution to $Ax = L(Ux) = g$ is then obtained in two steps. First a forward substitution to get $y = Ux$

$$y_1 = g_1, \qquad y_{k+1} = g_{k+1} - \gamma_k y_k, \quad k = 1 : n-1, \tag{1.4.12}$$

followed by a backward recursion for x

$$x_n = y_n/\beta_n, \qquad x_k = (y_k - c_k x_{k+1})/\beta_k, \quad k = n-1 : -1 : 1. \tag{1.4.13}$$

In this algorithm the LU factorization requires only about n divisions and n multiplications and additions. The solution of the two triangular systems require about twice as much work.

Consider the case when in step k of Gaussian elimination a zero pivotal element is encountered, i.e. $a_{kk}^{(k)} = 0$. (The equations may have been reordered in previous steps, but we assume that the notations have been changed accordingly.) If A is nonsingular, then in particular its first k columns are linearly independent. This must also be true for the first k columns of the reduced matrix and hence some element $a_{ik}^{(k)}$, $i = k : n$ must be nonzero, say $a_{rk}^{(k)} \neq 0$. By interchanging rows k and r this element can be taken as pivot and it is possible to proceed with the elimination. The important conclusion is that *any nonsingular system of equations can be reduced to triangular form by Gaussian elimination, if appropriate row interchanges are used*.

Note that when rows are interchanged in A the same interchanges must be made in the elements of the right-hand side b. Also the computed factors L and U will be the same as had the the row interchanges first been performed on A and the Gaussian elimination been performed without interchanges.

To ensure the numerical stability in Gaussian elimination it will, except for special classes of linear systems, be necessary to perform row interchanges *not only when a pivotal element is exactly zero*. Usually it suffices to use **partial pivoting**, i.e. to choose the pivotal element in step k as the element of largest magnitude in the unreduced part of the kth column.

Example 1.4.3.

The linear system

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

is nonsingular for any $\epsilon \neq 1$ and has the unique solution $x_1 = -x_2 = -1/(1-\epsilon)$. However, when $a_{11} = \epsilon = 0$ the first step in Gaussian elimination cannot be carried out. The remedy here is obviously to interchange the two equations, which directly gives an upper triangular system.

Suppose that in the system above $\epsilon = 10^{-4}$. Then the exact solution, rounded to four decimals equals $x = (-1.0001, 1.0001)^T$. However, if Gaussian elimination is carried through without interchanges we obtain $l_{21} = 10^4$ and the triangular system

$$\begin{aligned} 0.0001x_1 + x_2 &= 1 \\ (1 - 10^4)x_2 &= -10^4. \end{aligned}$$

Suppose that the computation is performed using arithmetic with three decimal digits. Then in the last equation the coefficient $a_{22}^{(2)}$ will be rounded to -10^4 and the solution computed by back-substitution is $\bar{x}_2 = 1.000$, $\bar{x}_1 = 0$, which is a catastrophic result!

If before performing Gaussian elimination we interchange the two equations then we get $l_{21} = 10^{-4}$ and the reduced system becomes

$$\begin{aligned} x_1 + x_2 &= 0 \\ (1 - 10^{-4})x_2 &= 1. \end{aligned}$$

The coefficient $a_{22}^{(2)}$ is now rounded to 1, and the computed solution becomes $\bar{x}_2 = 1.000$, $\bar{x}_1 = -1.000$, which is correct to the precision carried.

In this simple example it is easy to see what went wrong in the elimination without interchanges. The problem is that *the choice of a small pivotal element gives rise to large elements in the reduced matrix* and the coefficient a_{22} in the original system is lost through rounding. Rounding errors which are small when compared to the large elements in the reduced matrix are unacceptable in terms of the original elements! When the equations are interchanged the multiplier is small and the elements of the reduced matrix of the same size as in the original matrix.

In general an algorithm is said to be **backward stable** if the computed solution $\overline{w}$ always equals *the exact solution* of a problem with "slightly perturbed data". It will be shown in Volume II, Sec. 7.5, that backward stability can almost always be ensured for Gaussian elimination with partial pivoting. The essential condition for stability is that no substantial growth occurs in the elements in L and U. To formulate a basic result of the error analysis we need to introduce some new notations. In the following the absolute values $|A|$ and $|b|$ of a matrix A and vector b should be interpreted componentwise,

$$|A|_{ij} = (|a_{ij}|), \qquad |b|_i = (|b_i|).$$

Similarly the **partial ordering** "$\leq$" for the absolute values of matrices $|A|$, $|B|$ and vectors $|b|$, $|c|$, is to be interpreted component-wise.

Theorem 1.4.1.

Let $\overline{L}$ and $\overline{U}$ denote the LU factors and $\overline{x}$ the solution of the system $Ax = b$, using LU factorization and substitution. Then $\overline{x}$ satisfies exactly the linear system

$$(A + \Delta A)\overline{x} = b, \tag{1.4.14}$$

where δA is a matrix depending on both A and b, such that

$$|\Delta A| \lessapprox 3nu|\overline{L}|\,|\overline{U}|, \tag{1.4.15}$$

where u is a measure of the precision in the arithmetic.

It is important to note that the result that the solution satisfies (1.4.14) with a small $|\Delta A|$ *does not mean that the solution has been computed with a small error.* If the matrix A is ill-conditioned then the solution is very sensitive to perturbations in the data. This is the case, e.g., when the rows (columns) of A are almost linearly dependent. However, this inaccuracy is intrinsic to the problem and cannot be avoided except by using higher precision in the calculations. Condition numbers for linear systems are discussed in Sec. 2.4.4.

1.4.4 Sparse Matrices and Iterative Methods

A matrix A is called a **sparse** if it contains much fewer than the n^2 nonzero elements of a **full matrix** of size $n \times n$. Sparse matrices typically arise in many different applications. In Figure 1.4.1 we show a sparse matrix and its LU factors. In this case the original matrix is of order $n = 479$ and contains 1887 nonzero elements, i.e., less than 0.9% of the elements are nonzero. The LU factors are also sparse and contain together 5904 nonzero elements or about 2.6%.

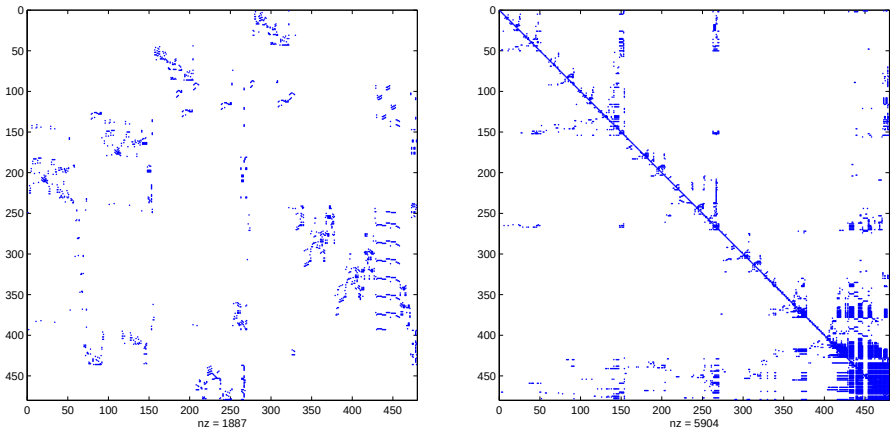


Figure 1.4.1. *Nonzero pattern of a sparse matrix and its LU factors.*

For many classes of sparse linear systems **iterative methods** when a are more efficient o use than **direct methods** such as Gaussian elimination. Typical

examples are those arising when a differential equation in 2D or 3D is discretized. In iterative methods a sequence of approximate solutions is computed, which in the limit converges to the exact solution x. Basic iterative methods work directly with the original matrix A and therefore has the added advantage of requiring only extra storage for a few vectors.

In a classical iterative method due to Richardson [35], a sequence of approximate solutions $x^{(k)}$ is defined by $x^{(0)} = 0$,

$$x^{(k+1)} = x^{(k)} + \omega(b - Ax^{(k)}), \quad k = 0, 1, 2, \ldots, \tag{1.4.16}$$

where $\omega > 0$ is a parameter to be chosen. It follows easily from (1.4.16) that the error in $x^{(k)}$ satisfies $x^{(k+1)} - x = (I - \omega A)(x^{(k)} - x)$, and hence

$$x^{(k)} - x = (I - \omega A)^k (x^{(0)} - x).$$

The convergence of Richardson's method will be studied in Sec. 10.1.4 in Volume II.

Iterative methods are used most often for the solution of very large linear systems, which typically arise in the solution of boundary value problems of partial differential equations by finite difference or finite element methods. The matrices involved can be huge, sometimes involving several million unknowns. The LU factors of matrices arising in such applications typically contain order of magnitudes more nonzero elements than A itself. Hence, because of the storage and number of arithmetic operations required, Gaussian elimination may be far too costly to use.

Example 1.4.4.

In a typical problem for Poisson's equation (1.2.15) the function is to be determined in a plane domain D, when the values of u are given on the boundary ∂D. Such **boundary value problems** occur in the study of steady states in most branches of Physics, such as electricity, elasticity, heat flow, fluid mechanics (including meteorology). Let D be the a square grid with grid size h, i.e. $x_i = x_0 + ih$, $y_j = y_0 + jh$, $0 \le i \le N+1$, $0 \le j \le N+1$. Then the difference approximation yields

$$u_{i,j+1} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} - 4u_{i,j} = h^2 f(x_i, y_j),$$

$(1 \le i \le M,\ 1 \le j \le N)$. This is a huge system of linear algebraic equations; one equation for each interior gridpoint, altogether N^2 unknown and equations. (Note that $u_{i,0},\ u_{i,N+1},\ u_{0,j},\ u_{N+1,j}$ are known boundary values.) To write the equations in matrix-vector form we order the unknowns in a vector

$$u = (u_{1,1}, \ldots, u_{1,N}, u_{2,1}, \ldots, u_{2,N-1}, u_{N,1}, \ldots, u_{N,N}).$$

If the equations are ordered in the same order we get a system $Au = b$ where A is symmetric with all nonzero elements located in five diagonals; see Figure 1.3.3 (left).

In principle Gaussian elimination can be used to solve such systems. However, even taking symmetry and the banded structure into account this would require $\frac{1}{2}N^4$ multiplications, since in the LU factors the zero elements inside the outer diagonals will fill-in during the elimination as shown in Figure 1.4.2 (right).

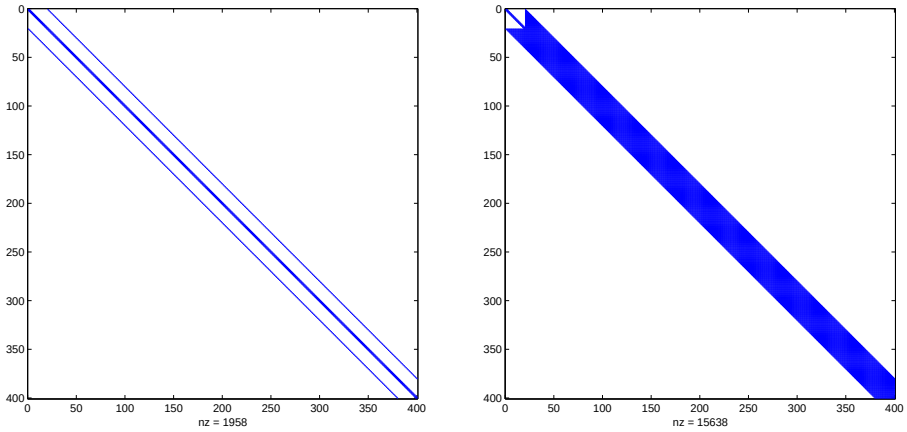


Figure 1.4.2. *Structure of A (left) and $L+U$ (right) for the Poisson problem, $N = 20$. (Row-wise ordering of the unknowns)*

The linear system arising from Poisson's equation has several features common to boundary value problems for all linear partial differential equations. One of these is that there are at most 5 nonzero elements in each row of A, i.e. only a tiny fraction of the elements are nonzero. Therefore one iteration in Richardson's method requires only about $5{\cdot}N^2$ multiplications or equivalently five multiplications per unknown. Using iterative methods which take advantage of the sparsity and other features does allow the efficient solution of such systems. This becomes even more essential for three-dimensional problems!

1.4.5 Software for Matrix Computations

In most computers in use today the key to high efficiency is to avoid as much as possible data transfers between memory, registers and functional units, since these can be more costly than arithmetic operations on the data. This means that the operations have to be carefully structured. One observation is that Gaussian elimination consists of three nested loops, which can be ordered in $3 \cdot 2 \cdot 1 = 6$ ways. Disregarding the right hand side vector b, each version does the operations

$$a_{ij}^{(k+1)} := a_{ij}^{(k)} - a_{kj}^{(k)} a_{ik}^{(k)} / a_{kk}^{(k)},$$

and only the ordering in which they are done differs. The version given above uses row operations and may be called the "kij" variant, where k refers to step number, i to row index, and j to column index. This version is not suitable for programming languages like Fortran 77, in which matrix elements are stored sequentially by columns. In such a language the form "kji" should be preferred, which is the column oriented back-substitution rather than Algorithm 1.3.1 might be preferred.

An important tool for structuring linear algebra computations are the Basic Linear Algebra Subprograms (BLAS). These are now commonly used to formulate

matrix algorithms and have become an aid to clarity, portability and modularity in modern software. The original set of BLAS identified frequently occurring vector operations in matrix computation such as scalar product, adding of a multiple of one vector to another. For example, the operation

$$y := \alpha x + y$$

in Single precision is named SAXPY. These BLAS were adopted in early Fortran programs and by carefully optimizing them for each specific computer the performance was enhanced without sacrificing portability.

For modern computers is is important to avoid excessive data movements between different parts of memory hierarchy. To achieve this so called level 3 BLAS have been introduced in the 1990s. These work on blocks of the full matrix and perform, e.g., the operations

$$C := \alpha AB + \beta C, \qquad C := \alpha A^T B + \beta C, \qquad C := \alpha AB^T + \beta C,$$

Since level 3 BLAS use $O(n^2)$ data but perform $O(n^3)$ arithmetic operations and gives a surface-to-volume effect for the ratio of data movement to operations. LAPACK [2], is a linear algebra package initially released in 1992, which forms the backbone of the interactive matrix computing system MATLAB . LAPACK achieves close to optimal performance on a large variety of computer architectures by expressing as much as possible of the algorithm as calls to level 3 BLAS.

Example 1.4.5.

In 1974 the authors wrote in [8, Sec. 8.5.3] that "a full $1,000 \times 1,000$ system of equations is near the limit at what can be solved at a reasonable cost". Today systems of this size can easily be handled by the a personal computer. The benchmark problem for Japanese Earth Simulator, one of the worlds fastest computers in 2004, was the solution of a system of size $1,041,216$ on which a speed of 35.6×10^{12} operations per second was measured. This is a striking illustration of the progress in high speed matrix computing that has occurred in these 30 years!

Review Questions

1. How many operations are needed (approximately) for
   (a) The multiplication of two square matrices?
   (b) The LU factorization of a square matrix?
   (b) The solution of $Ax = b$, when the triangular factorization of A is known?
2. Show that if the kth diagonal entry of an upper triangular matrix is zero, then its first k columns are linearly dependent.
3. What is the LU-decomposition of an n by n matrix A, and how is it related to Gaussian elimination? Does it always exist? If not, give sufficient conditions for its existence.

4. (a)For what type of linear systems are iterative methods to be preferred to Gaussian elimination?

(b) Describe Richardson's method for solving $Ax = b$. What can you say about the error in successive iterations?

5. What does the acronym BLAS stand for? What is meant by level 3 BLAS and why are they used in current linear algebra software??

Problems and Computer Exercises

1. (a) Let A and B be square upper triangular matrices of order n. Show that the product matrix $C = AB$ is also upper triangular. Determine how many multiplications are needed to compute C.

(b) Show that if R is an upper triangular matrix with zero diagonal elements, then $R^n = 0$.

2. Show that there cannot exist a factorization

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}.$$

Hint: Equate the $(1,1)$-elements and deduce that either the first row or the first column in LU must be zero.

3. (a) Consider the special upper triangular matrix of order n,

$$U_n(a) = \begin{pmatrix} 1 & a & a & \cdots & a \\ & 1 & a & \cdots & a \\ & & 1 & \cdots & a \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix},$$

Determine the solution x to the triangular system $U_n(a)x = e_n$, where $e_n = (0, 0, \ldots, 0, 1)^T$ is the nth unit vector.

(b) Show that the inverse of an upper triangular matrix is also upper triangular. Determine for $n = 3$ the inverse of of $U_n(a)$. Try also to determine $U_n(a)^{-1}$ for an arbitrary n.

Hint: Use the property of the inverse that $UU^{-1} = U^{-1}U = I$, the identity matrix.

4. A matrix H_n of order n such that $h_{ij} = 0$ whenever $i > j + 1$ is called an upper **Hessenberg** matrix. For $n = 5$ it has the structure e.g.,

$$H_5 = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ 0 & h_{32} & h_{33} & h_{34} & h_{35} \\ 0 & 0 & h_{43} & h_{44} & h_{45} \\ 0 & 0 & 0 & h_{54} & h_{55} \end{pmatrix}.$$

(a) Determine the approximate number of operations needed to compute the LU factorization of H_n if no pivoting is needed.

(b) Determine the approximate number of operations needed to solve the system $H_n x = b$, when the factorization in (a) is given.

5. Compute the product $|L|\,|U|$ for the LU factors of the matrix in Example 1.4.3 with and without pivoting.

1.5 Numerical Solution of Differential Equations

1.5.1 Euler's Method

Approximate solution of *differential equations* is a very important task in scientific computing. Nearly all the areas of science and technology contain mathematical models which leads to systems of ordinary (or partial) differential equations. An **initial value problem** for an ordinary differential equation is to find $y(x)$ such that

$$\frac{dy}{dt} = f(t, y), \quad y(0) = c.$$

The differential equation gives, at each point (t, y), the direction of the tangent to the solution curve which passes through the point in question. The direction of the tangent changes continuously from point to point, but the simplest approximation (which was proposed as early as the 18th century by Euler) is that one studies the solution for only certain values of $t = t_n = nh$, $n = 0, 1, 2, \ldots$ (h is called the "step" or "step length") and assumes that dy/dt is constant between the points. In this way the solution is approximated by a polygon segment (Fig. 1.4.1) which joins the points (t_n, y_n), $0, 1, 2, \ldots$, where

$$y_0 = c, \quad \frac{y_{n+1} - y_n}{h} = f(t_n, y_n). \tag{1.5.1}$$

Thus we have a simple *recursion formula*, **Euler's method**:

$$y_0 = c, \quad y_{n+1} = y_n + hf(t_n, y_n), \quad n = 0, 1, 2 \ldots \tag{1.5.2}$$

During the computation, each y_n occurs first on the left-hand side, then *recurs* later on the right-hand side of an equation: hence the name **recursion formula**. (One could also call equation (1.5.2) an iteration formula, but one usually reserves the word "iteration" for the special case where a recursion formula is used solely as a means of calculating a limiting value.)

1.5.2 An Introductory Example

One of the most important techniques in computer applications to science and technology is the **step by step simulation** of a process or the time development of a system. A **mathematical model** is first set up, i.e., **state variables** which describe the essential features of the state of the system are set up. Then the laws are formulated, which govern *the rate of change of the state variables*, and other

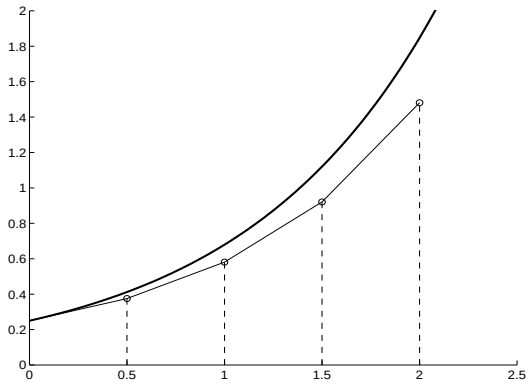


Figure 1.5.1. *Approximate solution of* $dy/dx = y$, $y_0 = 0.25$, *by Euler's method with* $h = 0.5$.

mathematical relations between these variables. Finally, these equations are programmed for a computer to calculate approximately, step by step, the development in time of the system.

The reliability of the results depends primarily on the goodness of the mathematical model and on the size of the time step. The choice of the time step is partly a question of economics. Small time steps may give you good accuracy, but also long computing time. More accurate numerical methods are often a good alternative to the use of small time steps. Such questions will be discussed in depth in Chapter 13 in Volume III.

The construction of a mathematical model is not trivial. Knowledge of numerical methods and programming helps also in that phase of the job, but more important is a good understanding of the fundamental processes in the system, and that is beyond the scope of this text. It is, however, important to realize that if the mathematical model is bad, no sophisticated numerical techniques or powerful computers can stop the results from being unreliable, or even harmful.

A mathematical model can be studied by analytic or computational techniques. Analytic methods do not belong to this text. We want, though, to emphasize that the comparison with results obtained by analytic methods, in the special cases when they can be applied, can be very useful when numerical methods and computer programs are tested. We shall now illustrate these general comments on a particular example.

Example 1.5.1.

Consider the motion of a ball (or a shot) under the influence of gravity and air resistance. It is well known that the trajectory is a parabola, when the air resistance is neglected and the force of gravity is assumed to be constant. We shall still neglect the variation of the force of gravity and the curvature and the rotation of the earth. This means that we forsake serious applications to satellites, etc. We shall, however, take the air resistance into account. We neglect the rotation of the shot around its

own axis. Therefore we can treat the problem a a motion in a plane, but we have to forsake the application to, for example, table tennis or a rotating projectile. Now we have introduced a number of assumptions, which define our **model** of reality.

The state of the ball is described by its position (x, y) and velocity (u, v), each of which has two Cartesian coordinates in the plane of motion. The x-axis is horizontal, and the y-axis is directed upwards. Assume that the air resistance is a force P, such that the direction is opposite to the velocity, and the strength is proportional to the square of the speed and to the square of the radius R of the shot. If we denote by P_x and P_y the components of P along the x and y directions, respectively, we can then write,

$$P_x = -mzu, \quad P_y = -mzv, \quad z = \frac{cR^2}{m}\sqrt{u^2 + v^2}, \tag{1.5.3}$$

where m is the mass of the ball.

For the sake of simplicity we assume that c is a constant. It actually depends on the density and the viscosity of the air. Therefore, we have to forsake the application to cannon shots, where the variation of the density with height is important. If one has access to a good model of the atmosphere, the variation of c would not make the numerical simulation much more difficult. This contrasts to analytic methods, where such a modification is likely to mean a considerable complication. In fact, even with a constant c, a purely analytic treatment offers great difficulties.

Newton's law of motion tells us that,

$$mdu/dt = P_x, \qquad mdv/dt = -mg + P_y, \tag{1.5.4}$$

where the term $-mg$ is the force of gravity. Inserting (1.5.3) into (1.5.4) and dividing by m we get

$$du/dt = -zu, \qquad dv/dt = -g - zv, \tag{1.5.5}$$

By the definition of velocity,

$$dx/dt = u, \qquad dy/dt = v, \tag{1.5.6}$$

Equations (1.5.5) and (1.5.6) constitute a system of four differential equations for the four variables x, y, u, v. The initial state x_0, y_0, and u_0, v_0 at time $t_0 = 0$ is assumed to be given. A fundamental proposition in the theory of differential equations tells that, if initial values of the state variables u, v, x, y are given at some initial time $t = t_0$, then they will be uniquely determined for all $t > t_0$.

The simulation in Example 1.5.1 means that, at a sequence of times, t_n, $n = 0, 1, 2, \ldots$, we determine the approximate values, u_n, v_n, x_n, y_n. We first look at the simplest technique, using Euler's method with a constant time step h. Set therefore $t_n = nh$. We replace the derivative du/dt by the forward difference quotient $(u_{n+1} - u_n)/h$, and similarly for the other variables. Hence after multiplication by h, the differential equations are replaced by the following system of **difference equations**:

$$\begin{aligned} u_{n+1} - u_n &= -hz_n u_n, \\ v_{n+1} - v_n &= -h(g + z_n v_n), \\ x_{n+1} - x_n &= hu_n, \quad y_{n+1} - y_n = hv_n, \end{aligned} \tag{1.5.7}$$

from which u_{n+1}, v_{n+1}, x_{n+1}, y_{n+1}, etc. are solved, step by step, for $n = 0, 1, 2, \ldots$, using the provided initial values u_0, v_0, x_0, y_0. Here z_n is obtained by insertion of $u = u_n$, $v = v_n$ into (1.5.3).

We performed these computations until y_{n+1} became negative for the first time, with $g = 9.81$, $\phi = 60^o$, and the initial values

$$x_0 = 0, \quad y_0 = 0, \quad u_0 = 100 \cos\phi, \quad v_0 = 100 \sin\phi.$$

In Fig. 1.4.2 are shown curves obtained for $h = 0.01$, and $cR^2/m = 0.25i \cdot 10^{-3}$, $i = 0, 1, 2, 3, 4$. There is, in this graphical representation, also an error due to the limited resolution of the plotting device.

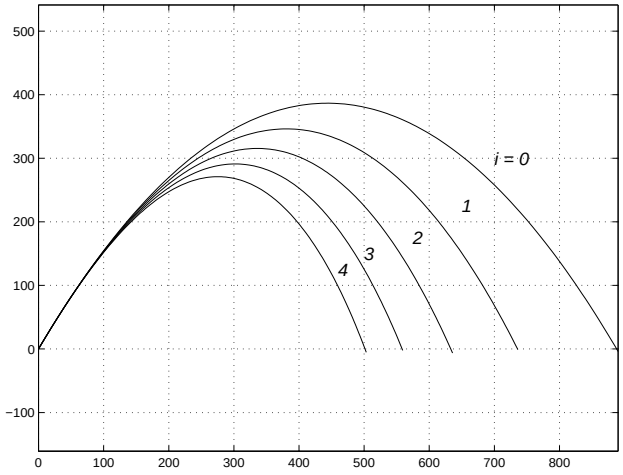


Figure 1.5.2. *Approximate trajectories computed with Euler's method for* $cR^2/m = 0.25i \cdot 10^{-3}$, $i = 0 : 4$, *and* $h = 0.01$.

In Euler's method the state variables are *locally approximated by linear functions* of time, one of the often recurrent ideas in numerical computation. We can use the same idea for computing the coordinate x^* of the point, where the shot hits the ground. Suppose that y_{n+1} becomes negative for the first time when $n = N$. For $x_N \le x \le x_{N+1}$ we then approximate y by a linear function of x, represented by the secant through the points (x_N, y_N) and(x_{N+1}, y_{N+1}) , i.e.,

$$y = y_N + (x - x_N)\frac{y_{N+1} - y_N}{x_{N+1} - x_N}.$$

By setting $y = 0$ we obtain

$$x^* = x_N - y_N \frac{x_{N+1} - x_N}{y_{N+1} - y_N}. \tag{1.5.8}$$

The error from the linear approximation in (1.5.8) used for the computation of x^* is proportional to h^2. It is thus approximately equal to the error committed in *one single step* with Euler's method, and hence of less importance than the other error.

The case without air resistance ($i = 0$) can be solved exactly. In fact it can be shown that $x^* = 2u_0v_0/9.81 = 5000 \cdot \sqrt{3}/9.81 = 882.7986$. The computer produced $x^* = 883.2985$ for $h = 0.01$, and $x^* = 883.7984$ for $h = 0.02$. The error for $h = 0.01$ is therefore 0.4999, and for $h = 0.02$ it is 0.9998. The approximate proportionality to h is thus verified, actually more strikingly than could be expected!

It can be shown that *the error in the results obtained with Euler's method is also proportional to h (not h^2).* Hence a disadvantage of the above method is that the step length h must be chosen quite short if reasonable accuracy is desired. In order to improve the method we can apply another idea mentioned in the previously, namely Richardson extrapolation. The application differs a little from the one you saw there, because now the error is approximately proportional to h, while for the trapezoidal rule it was approximately proportional to h^2. For $i = 4$, the computer produced $x^* = 500.2646$ and $x^* = 500.3845$ for, respectively, $h = 0.01$ and $h = 0.02$. Now let x^* denote the *exact* coordinate of the landing point. Then

$$x^* - 500.2646 \approx 0.01k, \qquad x^* - 500.3845 \approx 0.02k.$$

By elimination of k we obtain

$$x^* \approx 2 \cdot 500.2646 - 500.3845 = 500.1447,$$

which should be a more accurate estimate of the landing point. By a more accurate integration method we obtained 500.1440. So in this case, we gained more than two decimal digits by the use of Richardson extrapolation.

The simulations shown in Fig. 1.4.2 required about 1500 time steps for each curve. This may seem satisfactory, but we must not forget that this is a very small task, compared to most serious applications. So we would like to have a method that allows *much larger time steps* than Euler's method.

1.5.3 A Second Order Accurate Method

In step by step computations we have to distinguish between the **local error**, i.e., the error that is committed at a single step, and the **global error**, i.e., the error of the final results. Recall that we say that a method is accurate of order p, if its global error is approximately proportional to h^p. Euler's method is only first order accurate; we shall below present a method that is second order accurate. To achieve the same accuracy as with Euler's method the number of steps can then be reduced to about the square root of the number of steps in Euler's method, e.g., in the above ball problem to $\sqrt{1500} \approx 40$ steps. Since the amount of work is closely proportional to the number of steps this is an enormous saving!

Another question is how the step size h is to be chosen. It can be shown that even for rather simple examples (see below) it is adequate to use very *different step size* in different parts of the computation. Hence the automatic control of the step size (also called *adaptive* control) is an important issue.

Both requests can be met by an improvement of the Euler method (due to Runge) obtained by the applying the Richardson extrapolation in every second step. This is different from our previous application of the Richardson idea. We

first introduce a better notation by writing a **system of differential equations** and the initial conditions in vector form

$$d\mathbf{y}/dt = \mathbf{f}(t, \mathbf{y}), \quad \mathbf{y}(a) = \mathbf{c}, \tag{1.5.9}$$

where $\mathbf{y}$ is a column vector that contains all the state variables.[9] With this notation methods for large systems of differential equations can be described as easily as methods for a single equation. The change of a system with time can then be thought of as a motion of the state vector in a multidimensional space, where the differential equation defines the **velocity field**. This is our first example of the central role of vectors and matrices in modern computing. We temporarily use *superscripts* for the vector components, because we need subscripts for the same purpose as in the above description of Euler's method.

For the ball example, we have by (1.5.5) and (1.5.6)

$$\mathbf{y} = \begin{pmatrix} y^1 \\ y^2 \\ y^3 \\ y^4 \end{pmatrix} \equiv \begin{pmatrix} x \\ y \\ u \\ v \end{pmatrix}, \quad \mathbf{f}(t, \mathbf{y}) = \begin{pmatrix} y^3 \\ y^4 \\ -zy^3 \\ -g - zy^4 \end{pmatrix}, \quad \mathbf{c} = 10^2 \begin{pmatrix} 0 \\ 0 \\ \cos\phi \\ \sin\phi \end{pmatrix},$$

where

$$z = \frac{cR^2}{m}\sqrt{(y^3)^2 + (y^4)^2}.$$

The computations in the step which leads from t_n to t_{n+1} are then as follows:

i. One Euler step of length h yields the estimate:

$$\mathbf{y}^*_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n).$$

ii. Two Euler steps of length $\frac{1}{2}h$ yield another estimate:

$$\mathbf{y}_{n+\frac{1}{2}} = \mathbf{y}_n + \frac{1}{2}h\mathbf{f}(t_n, y_n); \qquad \mathbf{y}^{**}_{n+1} = \mathbf{y}_{n+\frac{1}{2}} + \frac{1}{2}h\mathbf{f}(t_{n+1/2}, \mathbf{y}_{n+1/2}),$$

where $t_{n+1/2} = t_n + h/2$.

iii. Then $\mathbf{y}_{n+1}$ is obtained by Richardson extrapolation:

$$\mathbf{y}_{n+1} = \mathbf{y}^{**}_{n+1} + (\mathbf{y}^{**}_{n+1} - \mathbf{y}^*_{n+1}).$$

It is conceivable that this yields a 2nd order accurate method. It is left as an exercise (Problem 2) to verify that *this scheme is identical to the following somewhat simpler scheme* known as **Runge's 2nd order method**:

$$\begin{aligned} \mathbf{k}_1 &= h_n\mathbf{f}(t_n, \mathbf{y}_n); \\ \mathbf{k}_2 &= h_n\mathbf{f}(t_n + h_n/2, \mathbf{y}_n + \mathbf{k}_1/2); \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \mathbf{k}_2, \end{aligned} \tag{1.5.10}$$

[9] The boldface notation is temporarily used for vectors *in this section*, not in the rest of the book.

where we have replaced h by h_n in order to include the use of variable step size. Another explanation of the 2nd order accuracy of this method is that the displacement $\mathbf{k}_2$ equals the product of the step size and a sufficiently accurate estimate of the velocity at the *midstep* of the time step. A more detailed analysis of this method comes in Sec. 13.3.2. Sometimes this method is called the improved Euler method or Heun's method, but these names are also used to denote other 2nd order accurate methods.

We shall now describe how the step size can be **adaptively** (or automatically) controlled by means of a tolerance TOL, by which the user tells the program how large error he tolerates in values of variables (relative to the values themselves).[10] Compute

$$\delta = \max_i |k_2^i - k_1^i|/|3y^i|,$$

where δ is related to the *relative* errors of the components of the vector $\mathbf{y}$; see below.

A step size is *accepted* if $\delta \leq \text{TOL}$, and the next step should be

$$h_{next} = h \min\{1.5, \sqrt{\text{TOL}/(1.2\delta)}\},$$

where 1.2 is a safety factor, since the future is never exactly like the past .... The square root occurring here is due to the fact that this method is 2nd order accurate, i.e., the global error is almost proportional to the square of the step size and δ is approximately proportional to h^2.

A step is *rejected*, if $\delta > \text{TOL}$, and *recomputed* with the step size

$$h_{next} = h \max\{0.1, \sqrt{\text{TOL}/(1.2\delta)}\}.$$

The program needs a suggestion for the size of the first step. This can be be a very rough guess, because the step size control described above, will improve it automatically, so that an adequate step size is found after a few steps (or recomputations, if the suggested step was too big). In our experience, a program of this sort can efficiently handle guesses that are wrong by several powers of 10. If $y(a) \neq 0$ and $y'(a) = 0$, you may try the initial step size

$$h = \frac{1}{4} \sum_i |y^i| \Big/ \sum_i |dy^i/dt|$$

evaluated at the initial point $t = a$. When you encounter the cases $y(a) = 0$ or $y'(a) = 0$ for the first time, you are likely to have gained enough experience to suggest something that the program can handle. More professional programs take care of this detail automatically.

The request for a certain *relative* accuracy may cause trouble when some components of y are close to zero. So, already in the first version of your program, you had better *replace* y^i *in the above definition of* δ *by* $\bar{y}^i = \max\{|y^i|, 0.001\}$.

[10] With the terminology that will be introduced in the next chapter, TOL is, with the step size control described here, related to the *global relative errors* . At the time of writing, this contrasts to most codes for the solution of ordinary differential equations, in which the *local* errors per step are controlled by the tolerance.

A more detailed discussion of such matters follows in Sections 13.1 and 13.2 in Volume II (see in particular Computer Exercise 13.1.1). (You may sometimes have to replace the default value 0.001 by something else.)

It is a good habit to make a second run with a predetermined sequence of times (if your program allows this) instead of adaptive control. Suppose that the sequence of times used in the first run is t_0, t_1, t_2, ... Divide each subinterval $[t_n, t_{n+1}]$ into two steps of equal length. So, the second run still has variable step size and twice as many steps as the first run. The errors are therefore expected to be approximately $\frac{1}{4}$ of the errors of the first run. The first run can therefore use a tolerance that is 4 times as large than the error you can tolerate in the final result. Denote the results of the two runs by $y_I(t)$ and $y_{II}(t)$. You can plot $\frac{1}{3}(y_{II}(t) - y_I(t))$ versus t; this is an error curve for $y_{II}(t)$ Alternatively you can add $\frac{1}{3}(y_{II}(t) - y_I(t))$ to $y_{II}(t)$. This is another application of the Richardson extrapolation idea. The cost is only 50% more work than the plain result without an error curve.

If there are no singularities in the differential equation, $\frac{1}{3}(y_{II}(t) - y_I(t))$ *strongly overestimates the error of the extrapolated values*—typically by a factor like $\text{TOL}^{-1/2}$. It is, however, a non-trivial matter to find an error curve that strictly and realistically tells how good the extrapolated results are. There will be more comments about these matters in Sec. 3.3.4 in Volume II (see also Example 13.2.1 in Volume III). The reader is advised to test experimentally how this works on examples where the exact results are known.

An easier, though inferior, alternative is to run a problem with two different tolerances. One reason why it is inferior is that the two runs do not "keep in step". For example, Richardson extrapolation cannot be easily applied.

If you request very high accuracy in your results, or if you are going to simulate a system over a very long time, you will need a method with a higher order of accuracy than two. The reduction of computing time if you replace this method by a higher order method can be large, but the improvements are seldom as drastic as when you replace Euler's method by a second order accurate scheme like this. Runge's 2nd order method is, however, no universal recipe. There are special classes of problems, notably the problems which are called "stiff", which need special methods. These matters are treated in Chapter 13.

One advantage of a second order accurate scheme when requests for accuracy are modest, is that the quality of the computed results is normally not ruined by the use of *linear interpolation* at the graphical output, or at the post-processing of numerical results. (After you have used a more than second order accurate integration method, it may be necessary to use a more sophisticated interpolation at the graphical or numerical treatment of the results.)

Example 1.5.2.

The differential equation $y' = -\frac{1}{2}y^3$, with initial condition $y(1) = 1$, was treated by a program, essentially constructed as described above, with $\text{TOL} = 10^{-4}$ until $t = 10^4$.

In this example we can compare with the exact solution, $y(t) = t^{-1/2}$. It was found that the actual relative error stayed a little less than $1.5\,\text{TOL}$ all the time when $t > 10$. The step size increased almost linearly with t from $h = 0.025$ to

$h = 260$. The number of steps increased almost proportionally to $\log t$; the total number of steps was 374. Only one step had to be recomputed (except for the first step, where the program had to find an appropriate step size).

The computation was repeated with $\text{TOL} = 4 \cdot 10^{-4}$. The experience was the same, except that the steps were about twice as long all the time. This is what can be expected, since the step sizes should be approximately proportional to $\sqrt{\text{TOL}}$, for a second order accurate method. The total number of steps was 194.

Example 1.5.3.

The example of the motion of a ball was treated by Runge's 2nd order method with the constant step size $h = 0.9$. The coordinate of the landing point became $x^* = 500.194$, which is more than twice as accurate than the result obtained by Euler's method (without Richardson extrapolation) with $h = 0.01$, which uses about 90 times as many steps.

We have now seen a variety of ideas and concepts which can be used in the development of numerical methods. A small warning is perhaps warranted here: it is not certain that the methods will work as well in practice as one might expect. This is because approximations and the restriction of numbers to a certain number of digits introduce errors which are propagated to later stages of a calculation. The manner in which errors are propagated is decisive for the practical usefulness of a numerical method. We shall examine such questions in Chapter 2. Later chapters will treat **propagation of errors** in connection with various typical problems.

The risk that error propagation may up-stage the desired result of a numerical process should, however, not dissuade one from the use of numerical methods. It is often wise, though, to experiment with a proposed method on a simplified problem before using it in a larger context. The development of hardware as well as software has created a far better environment for such work than we had a decade ago. In this area too, the famous phrase of the Belgian-American chemist Baekeland holds:

> "*Commit your blunders on a small scale and make your profits on a large scale.*"

Review Questions

1. Explain the difference between the local and global error of a numerical method for solving a differential equation. What is meant by the order of accuracy for a method?
2. Describe how Richardson extrapolation can be used to increase the order of accuracy of Euler's method.

Problems and Computer Exercises

1. Integrate numerically using Euler's method the differential equation $dy/dx = y$, with initial conditions $y(0) = 1$, to $x = 0.4$:

(a) with step length $h = 0.2$ and $h = 0.1$.

(b) Extrapolate to $h = 0$, using the fact that the error is approximately proportional to the step length. Compare the result with the exact solution of the differential equation and determine the ratio of the errors in the results in (a) and (b).

(c) How many steps would have been needed in order to attain, without using extrapolation, the same accuracy as was obtained in (b)?

2. (a) Write a program for the simulation of the motion of the ball, using Euler's method and the same initial values and parameter values as above. Print only x, y at integer values of t and at the last two points (i.e. for $n = N$ and $n = N + 1$) as well as the coordinate of the landing point. Take $h = 0.05$ and $h = 0.1$. As post-processing, improve the estimates of x^* by Richardson extrapolation, and estimate the error by comparison with the results given in the text above.

(b) In Equation (1.5.8) replace in the equations for x_{n+1} and y_{n+1} the right hand sides u_n and v_n by, respectively, u_{n+1} and v_{n+1}. Then proceed as in (a) and compare the accuracy obtained with that obtained in (a).

(c) Choose initial values which correspond to what you think is reasonable for shot put. Make experiments with several values of u_0, v_0 for $c = 0$. How much is x^* influenced by the parameter cR^2/m?

3. Verify that Runge's 2nd order method, as described by equation (1.5.10), is equivalent to the scheme described a few lines earlier (with Euler steps and Richardson extrapolation).

4. Write a program for Runge's 2nd order method with automatic step size control that can be applied to a system of differential equations, *or* use the MATLAB program on the diskette. Store the results so that they can be processed afterwards, e.g., for making table of the results, and/or curves to be drawn showing $y(t)$ versus t, or (for a system) y^2 versus y^1, or some other interesting curves.

Apply the program to Examples 1.4.2 and 1.4.3, and to the circle test, i.e.

$$y_1' = -y_2, \qquad y_2' = y_1,$$

with initial conditions $y_1(0) = 1$, $y_2(0) = 0$. Verify that the exact solution is a uniform motion along the unit circle in the (y_1, y_2)-plane. Stop the computations after 10 revolutions ($t = 20\pi$). Make experiments with different tolerances, and determine how small the tolerance has to be in order that the circle on the screen should not become "thick".

1.6 Monte Carlo Methods

1.6.1 Origin of Monte Carlo Methods

In most of the applications of probability theory one makes a mathematical formulation of a stochastic problem (i.e., a problem where chance plays some part), and then solves the problem by using analytical or numerical methods. In the **Monte Carlo method**, one does the opposite; a mathematical or physical problem is given, and one constructs **numerical game of chance**, the mathematical analysis of which leads to the same equations as the given problem, e.g., for the probability of some event, or for the mean of some random variable in the game. One plays it N times and estimates the relevant quantities by traditional statistical methods. Here N is a large number, because the standard deviation of a statistical estimate typically *decreases only inversely proportional to* $\sqrt{N}$.

The idea behind the Monte Carlo method was used by the Italian physicist Enrico Fermi to study the neutron diffusion in the early 1930s. Fermi used a small mechanical adding machine for this purpose. With the development of computers larger problems could be tackled. At Los Alamos in the late 1940s the use of the method was pioneered by John von Neumann,[11] Ulam[12] and others for many problems in mathematical physics including approximating complicated multidimensional integrals. The picturesque name of the method was coined by Nicholas Metropolis.

The Monte Carlo method is now so popular that the definition is too narrow. For instance, in many of the problems where the Monte Carlo method is successful, there is already an element of chance in the system or process which one wants to study. Thus such games of chance can be considered to be a numerical simulation of the most important aspects. In this wider sense the "Monte Carlo methods also include techniques used by statisticians since around 1900, under names like *experimental* or *artificial sampling*. For example, one used statistical experiments to check the adequacy of certain theoretical probability laws, which the eminent scientist W .S .Gosset, who used the pseudonym "Student" when he wrote on Probability, had derived mathematically.

Monte Carlo methods may be used, when the changes in the system are described with a much more complicated type of equation than a system of ordinary differential equations. Note that there are many ways to combine analytical methods and Monte Carlo methods. An important rule is that *if a part of a problem can be treated with analytical or traditional numerical methods, then one should use such methods.*

[11] John von Neumann was born János Neumann in Budapest 1903, and died in Washington D.C. 1957. He studied under Hilbert in Göttingen during 1926–27, was appointed professor at Princeton University in 1931, and in 1933 joined the newly founded Institute for Advanced Studies in Princeton. He built a framework for quantum mechanics, worked in game theory and was one of the pioneers of computer science.

[12] Stanislaw Marcin Ulam, born in Lemberg, Poland (now Lwow, Ukraine) 1909, died in Santa Fe, New Mexico, USA, 1984. Ulam obtained his Ph.D. in 1933 from the Polytechnic institute of Lwow, where he studied under Banach. He was invited to Harward University by G. D. Birkhoff in 1935, and left Poland permanently in 1939. In 1943 he was asked by von Neumann to come to Los Alamos, where he remained until 1965.

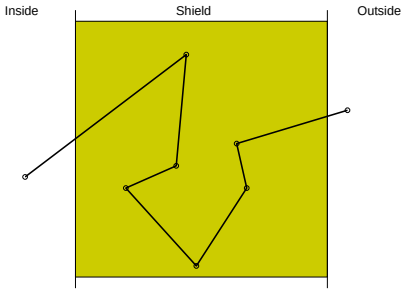


Figure 1.6.1. *Neutron scattering.*

The following are some areas where the Monte Carlo method has been applied:

(a) Problems in reactor physics; for example, a neutron, because it collides with other particles, is forced to make a random journey. In infrequent but important cases the neutron can go through a layer of (say) shielding material (see Fig. 1.5.1).

(b) Technical problems concerning traffic (telecommunication, railway networks, regulation of traffic lights and other problems concerning automobile traffic).

(c) Queuing problems.

(d) Models of conflict.

(e) Approximate computation of multiple integrals.

(f) Stochastic models in financial mathematics.

Monte Carlo methods are often used for the evaluation of high dimensional (10–100) integrals over complicated regions. Such integrals occur in such diverse areas as quantum physics and mathematical finance. The integrand is then evaluated at random points uniformly distributed in the region of integration. The arithmetic mean of these function values is then used to approximate the integral. Such randomization makes multivariate integration computationally feasible. Interestingly choosing the evaluation points uniformly distributed in the region of integration is not the optimal strategy. Instead one should use "quasi-random numbers" designed specifically for that purpose; see Sec. 5.??.

In a simulation, one can study the result of various actions more cheaply, more quickly, and with less risk of organizational problems than if one were to take the corresponding actions on the actual system. In particular, for problems in applied operations research, it is quite common to take a shortcut from the actual system to a computer program for the game of chance, without formulating any mathematical equations. The game is then a model of the system. In order for the term Monte Carlo method to be correctly applied, however, **random choices** should occur in the calculations. This is achieved by using so-called **random numbers**; the values of certain variables are determined by a process comparable to dice throwing.

Simulation is so important that several special programming languages have been developed exclusively for its use.[13]

In the rest of this section we assume that the reader is familiar with some basic concepts, formulas and results from Probability and Statistics, and we make use of them without proofs (which may be found in most texts on these subjects). The terminology of Probability and Statistics is varied, in particular within areas of application. We shall use the following terms for probability distributions in $\mathbf{R}$:

The **distribution function** of a random variable X is denoted by $F(x)$ and defined by

$$F(x) = Pr\{X \leq x\}.$$

Note that $F(x)$ is non-negative and non-decreasing, $F(-\infty) = 0$, $F(\infty) = 1$. If $F(x)$ is differentiable, the (probability) **density function** [14] is $f(x) = F'(x)$. Note that $f(x) \geq 0$, $\int_{\mathbf{R}} f(x)\,dx = 1$, and

$$Pr\{X \in [x, x + dx] = f(x)\,dx + o(dx)\}.$$

The **mean** or the *expectation* is

$$E(X) = \begin{cases} \int_{\mathbf{R}} x f(x)\,dx, & \text{continuous case,} \\ \sum_i p_i x_i, & \text{discrete case,} \end{cases}$$

where the p_i are probabilities that satisfy the conditions $p_i \geq 0$, $\sum_i p_i = 1$, $i = 1 : N$. The **variance** of X equals

$$\text{var}(X) = \begin{cases} E((X - m)^2), & \text{continuous case,} \\ \sum_i p_i (x_i - m)^2, & \text{discrete case,} \end{cases}$$

where $m = E(X)$. The **standard deviation**, $\text{std}(X) = \sqrt{\text{var}(X)}$. Some formulas for the estimation of mean, standard deviation etc., from results of simulation experiments or other statistical data are given in the computer exercises of Sec. 2.3. See also the references to the Matlab Reference Guide in the problems and exercises of the present section.

1.6.2 Random and Pseudo-Random Numbers

In the beginning coins, dice and roulettes were used for creating the randomness, e.g., the sequence of twenty digits

$$11100 \;\; 01001 \;\; 10011 \;\; 01100$$

is a record of twenty tosses of a coin where "heads" are denoted by 1 and "tails" by 0. Such digits are sometimes called (binary) **random digits**, assuming that we

[13] One notable example is the SIMULA programming language designed and built by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center in Oslo 1962–1967, It was originally built as a language for discrete event simulation, but was influential also because it introduced object-oriented programming concepts.

[14] In old literature a density function is often called a frequency function. The term cumulative distribution is also used as a synonym of distribution function. Unfortunately, distribution or probability distribution is sometimes used in the meaning of a density function.

have a perfect coin—i.e., that heads and tails have the same probability of occurring. We also assume that the tosses of the coin are made in a statistically independent way. (Of course, these assumptions cannot be obtained in practice, as shown in theoretical and experimental studies by Persi Diaconis, Stanford University.)

Similarly, decimal random digits could in principle be obtained by using a well-made icosahedral (twenty-sided) dice, and assigning each decimal digit to two of its sides. Such mechanical (or analogous electronical) devices have been used to produce **tables of random sampling digits**; the first one by Tippett was published in 1927 and was to be considered as a sequence of 40000 independent observations of a random variable that equals one of the integer values $0, 1, 2, \ldots, 9$, each with probability 1/10. In the early 1950s the Rand Corporation constructed a million-digit table of random numbers using an electrical "roulette wheel" ([6, 1955]). The wheel had 32 slots, of which 12 were ignored; the others were numbered from 0 to 9 twice. To test the quality of the randomness several tests were applied. Every block of a thousand digits in the tables (and also the table as a whole) were tested.

Random digits from a table can be packed together to give a sequence of equi-distributed integers. For example, the sequence

$$55693 \quad 02945 \quad 81723 \quad 43588 \quad 81350 \quad 76302 \quad \ldots$$

can be considered as six five-digit random numbers, where each element in the sequence has probability of 10^{-5} of taking on the value, 0,1,2,...,99,999. From the same digits one can also construct the sequence

$$0.556935, 0.029455, 0.817235, 0.435885, 0.813505, 0.763025, \ldots, \tag{1.6.1}$$

which can be considered a good approximation to a sequence of independent observations of a variable which is a sequence of uniform deviates in on the interval $[0, 1)$. The 5 in the sixth decimal place is added in order to get the correct mean (without this the mean would be 0.499995 instead of 0.5).

We shall return to this in the next subsection, together with the further development in the computer age, where **arithmetic methods** are used for producing the so-called **pseudo-random numbers** needed for the large-scale simulations that nowadays are demanded, e.g. in the areas applications mentioned below.[15]

In a computer it is usually not appropriate to store a large table of random numbers. One instead computes a sequence of uniform deviates u_0, u_1, $u_2, \ldots, \in [0, 1]$, by a **random number generator**, i.e., some arithmetic algorithm. Sequences obtained in this way are uniquely determined by one or more starting values (**seeds**), to be given by the user (or some default values). The aim of a pseudo-random number generator is to imitate the abstract mathematical concept of mutually independent random variables uniformly distributed over the interval $[0, 1)$. They should be analyzed theoretically and be backed by practical evidence

[15] Several physical devices for random number generation, using for instance electronic or radioactive noise, have been proposed but very few seem to have been inserted in an actual computer system.

from extensive statistical testing. According to a much quoted statement by D. H. Lehmer[16]

> "*A random sequence is a vague notion ... in which each term is unpredictable to the uninitiated and whose digits pass a certain number of tests traditional with statisticians...*"

Because the set of floating point numbers in $[0, 1]$ is finite, although very large, there will eventually appear a number that has appeared before, (say) $u_{i+j} = u_i$ for some positive i, j. The sequence $\{u_n\}$ therefore repeats itself periodically for $n \geq i$; the length of the period is j. A truly random sequence is, of course, never periodic. A sequence generated like this is, for this and for other reasons, called **pseudo-random**. However, the ability to repeat exactly the same sequence of numbers, which is needed for program verification and variance reduction, is a major advantage over generation by physical devices.

There are two popular myths about the making of random number generators:

$$(1)\ \textit{it is impossible}; \qquad (2)\ \textit{it is trivial}\ldots.$$

We have seen that the first myth is correct, unless we add the prefix "pseudo".[17] The second myth, however, is completely false.

In a computer the fundamental concept is not a sequence of decimal random *digits*, but the **uniform random deviates**, i.e., a sequence of *mutually independent observations of a random variable* U with a uniform distribution on $[0, 1)$; the density function of U is thus (with a temporary notation)

$$f_1(u) = \begin{cases} 1, & \text{if } u \in (0, 1); \\ 0, & \text{otherwise.} \end{cases}$$

Random deviates for other distributions, are generated by means of uniform deviates, e.g., the variable $X = a + (b - a)U$ is a *uniform deviate on* (a, b).. Its density function is$f(x) = f_1((x - a)/(b - a))$. If $[a, b] = [0, 1]$ we usually write "uniform deviate" (without mentioning the interval). We often write "deviate" instead of "random deviate", when the meaning is evident from the context.

The most widely generators used for producing pseudo-random numbers are the **multiple recursive generator** based on linear recurrences of order k

$$x_i = a_1 x_{i-1} + \cdots + a_k x_{i-k} + c \mod m, \tag{1.6.2}$$

i.e., x_i is the remainder obtained when the right hand side is divided by the modulus m. Here m is a positive integer and the coefficients $a_1, \ldots, a_k$ belong to the set $\{0, 1, \ldots, m - 1\}$. The state at step i is $s_i = (x_{i-k+1}, \ldots, x_i)$ and the generator is started from a seed $s_{k-1} = (x_0, \ldots, x_{k-1})$. When m is large the output can

[16] Some readers may think that Lehmer's definition is too vague. There have been many deep attempts for more precise formulation. See Knuth [pp. 149–179]knut:97, who catches the flavor of the philosophical discussion of these matters and contributes to it himself.

[17] "Anyone who considers arithmetic methods of producing random numbers is, of course, in a state of sin ", John von Neumann (1951).

be taken as the number $u_i = x_i/m$. When $k = 1$, we obtain the classical **linear congruential generator**.

An important characteristic of a random number generator is its **period**, which is the maximum length of the sequence before it begins to repeat. Note that if the algorithm for computing x_i only depends on x_{i-1}, then the entire sequence repeats once the seed x_0 is duplicated.

A good RNG should have an extremely long period. If m is a prime number and and if the coefficients a_j satisfy certain conditions, then the generated sequence has the maximal period length $m^k - 1$; see Knuth [18].

The linear congruential generator defined by

$$x_i = 16807x_{i-1} \mod (2^{31} - 1), \tag{1.6.3}$$

with period length $(2^{31}-2)$, was proposed originally by Lewis, Goodman, and Miller (1969). It has been widely used in many software libraries for statistics, simulation and optimization. In the survey by Park and Miller [32] this generator was proposed as a "minimal standard" against which other generators should be judged. A similar generator but with the multiplier $7^7 = 823543$ was used in MATLAB 4.

Marsaglia [25] pointed out a theoretical weakness of all linear congruential generators. He showed that if k successive random numbers $(x_{i+1}, \ldots, x_{i+k})$ at a time are generated and used to plot points in k-dimensional space, then they will lie on $(k-1)$-dimensional hyperplanes, and will not fill up the space. More precisely the values will lie on a set of, at most $(k!m)^{1/k} \sim (k/e)m^{1/k}$ equidistant parallel hyperplanes in the k-dimensional hypercube $(0,1)^k$. When the number of hyperplanes is too small, this obviously is a strong limitation to the k-dimensional uniformity. For example, for $m = 2^{31} - 1$ and $k = 3$, this is only about 1600 planes. This clearly may interfere with a simulation problem.

If the constants m, a and c are not very carefully chosen, there will be many fewer hyperplanes than the maximum possible. One such infamous example is the linear congruential generator with $a = 65539, c = 0$ and $m = 2^{31}$ used by IBM mainframe computers for many years.

Another weakness of linear congruential generators is that their low-order digits are much less random than their high-order digits. Therefore when only part of a generated random number is used one should pick the high-order digits.

One approach to better generators is to combine two RNGs. One possibility is to use a second RNG to shuffle the output of a linear congruential generator. In this way it is possible to get rid of some serial correlations in the output; see the generator ran1 described in Press et. al. [34, Chapter 7.1].

Dahlquist in 1962 [7] developed a random number generator to be used for drawing of prizes of Swedish Premium Saving Bonds. This starts with a primary series of random digits, produced by some mechanical device. This primary series has length $n = p_1 + p_2 + \cdots + p_k$, where p_i are prime numbers and $p_i \neq p_j$, $i \neq j$. From this a secondary series of random digits with a period of $p_1 \cdot p_2 \cdots p_n$ is generated by cyclically adding k digits mod 10.

At the time of writing simplistic and unreliable RNGs still abound in some other commercial software products, despite the availability of much better alternatives. L'Ecuyer [22] reports on tests of RNGs used in some popular software

products. Microsoft Excel uses the linear congruential generator

$$u_i = 9821.0u_{n-1} + 0.211327 \mod 1,$$

implemented directly for the u_i in floating point arithmetic. Its period length depends on the precision of the arithmetic and it is not clear what it is. Microsoft Visual Basic uses a linear congruential generator with period 2^{24}, defined by

$$x_i = 1140671485x_{i-1} + 12820163 \mod (2^{24}),$$

and takes $u_i = x_i/2^{24}$. The Unix standard library uses the recurrence

$$x_i = 25214903917x_{i-1} + 12820163 \mod (2^{48}),$$

with period length 2^{48} and sets $u_i = x_i/2^{48}$. The Java standard library uses the same recurrence but construct random deviates u_i from x_{2i} and x_{2i+1}.

One conclusion of recent tests is that when large sample sizes are needed all the above RNGs are unsafe to use and can fail decisively. It has been observed that to avoid misleading results the period length ρ of the RNG needs to be such that generating $\rho^{1/3}$ numbers is not feasible. Thus a period length of 2^{24} or even 2^{48} may not be enough. Linear RNGs are also unsuitable for cryptographical applications, because the output is too predictable. For this reason, nonlinear generators have been developed, but these are in general much slower than the linear generators.

In MATLAB 5 and later versions the previous linear congruential generator has been replaced with a much better generator, based on ideas of G, Marsaglia. This generator has a 35 element state vector and can generate all the floating point numbers in the closed interval $[2^{-53}, 1 - 2^{-53}]$. Theoretically it can generate 2^{1492} values before repeating itself; see Moler [29]. If one generates one million random numbers a second it would take 10^{435} years before it repeats itself!

Some recently developed linear RNGs can generate huge samples of pseudo-random numbers very fast and reliably. The multiple recursive generator MRG32k3a proposed by L'Ecuyer has a period near 2^{191}. The **Mersenne twister** MT19937 by Matsumoto and Nishimura [28], the current "World Champion" among RNGs, has a period length of $2^{19937} - 1$!

1.6.3 Testing Pseudo-Random Number Generators

Many statistical tests have been adapted and extended for the examination of *arithmetic* methods of (pseudo-)random number generation, in use or proposed for digital computers. In these the observed frequencies (a histogram) for some random variable associated with the test, is compared with the theoretical frequencies on the hypothesis that the numbers are independent observations from a true sequence of random digits without bias. This is done by means of the famous χ^2-test of K. Pearson [33][18], which we now describe.

[18] This paper by the English mathematician Karl Pearson (1857–1936) is considered as one of the foundations of modern statistics. In it he gave several examples, e.g., he proved that some runs at roulette he had observed during a visit to Monte Carlo were so far from expectations that the odds against an honest wheel was about 10^{29} to one.

Suppose that the space S of the random variable is divided into a finite number r of non-overlapping parts $S_1, \ldots, S_r$. These parts may be groups into which the sample values have been arranged for tabulation purposes. Let the corresponding group probabilities be $p_i = Pr(S_i)$, $i = 1, \ldots, r$, where $\sum_i p_i = 1$. We now form a measure of the deviation of the observed frequencies $\nu_1, \ldots, \nu_r$, $\sum_i \nu_i = n$, from the expected frequencies

$$\chi^2 = \sum_{i=1}^{r} \frac{(\nu_i - np_i)^2}{np_i} = \sum_{i=1}^{r} \frac{\nu_i^2}{np_i} - n. \tag{1.6.4}$$

It is known that as n tends to infinity the distribution of χ^2 tends to a limit independent of $P(S_i)$, which is the χ^2-distribution with $r-1$ degrees of freedom.

Now let χ_p^2 be a value such that $Pr(\chi^2 > \chi_p^2) = p\%$. Here p is chosen so small that we are practically certain that an event of probability $p\%$ will not occur in a single trial. The hypothesis is rejected if the observed value of χ^2 is larger than χ_p^2. Often a rejection level of 5% or 1% is used.

Example 1.6.1.

In $n = 4040$ throws with a coin, Buffon obtained $\nu = 2048$ heads and hence $n - \nu = 1992$ tails. Is this consistent with the hypothesis that there is a probability of $p = 1/2$ of throwing tails? Here we obtain

$$\chi^2 = \frac{(\nu_i - np)^2}{np} + \frac{(n - \nu - np)^2}{np_i} = 2\frac{(2048 - 2020)^2}{2020} = 0.776.$$

Using a rejection level of 5% we find from a table of the χ^2-distribution with one degree of freedom that $\kappa_5^2 = 3.841$. Hence the hypothesis is accepted at this level.

Some test that have been used for testing RNGs are:

1. **Frequency test** This test is to find out if the generated numbers are equidistributed. One divides the possible outcomes in equal non-overlapping intervals and tallies the amount of numbers in each interval.

2. **Poker test** This test applies to generated digits, which are divided into non-overlapping groups of 5 digits. Within the groups we study some (unordered) combinations of interest in poker. These are given below together with their probabilities.

All different:	abcde	0.3024
One pair:	aabcd	0.5040
Two pairs:	aabbc	0.1080
Three of a kind:	aaabc	0.0720
Full house:	aaabb	0.0090
Four of a kind:	aaaab	0.0045
Five of a kind:	aaaaa	0.0001

3. **Gap test** This test examines the length of "gaps" between occurences of U_j in a certain range. If α and β are two numbers with $0 \le \alpha < \beta \le 1$, we consider the length of consecutive subsequences $U_j, U_{j+1}, \ldots, U_{j+r}$ in which U_{j+r} lies between α and β but $U_j, U_{j+1}, \ldots, U_{j+r-1}$ does not. This subsequence then represents a gap of length r.

   Working with single digits the gap equals the distance between two equal digits. The probability of a gap of length r in this case equals

$$p_r = 0.1(1-0.1)^r = 0.1(0.9)^r, \quad r = 0, 1, 2, \ldots.$$

   Several other tests are decribed in Knuth [18, Sec. 3.3].

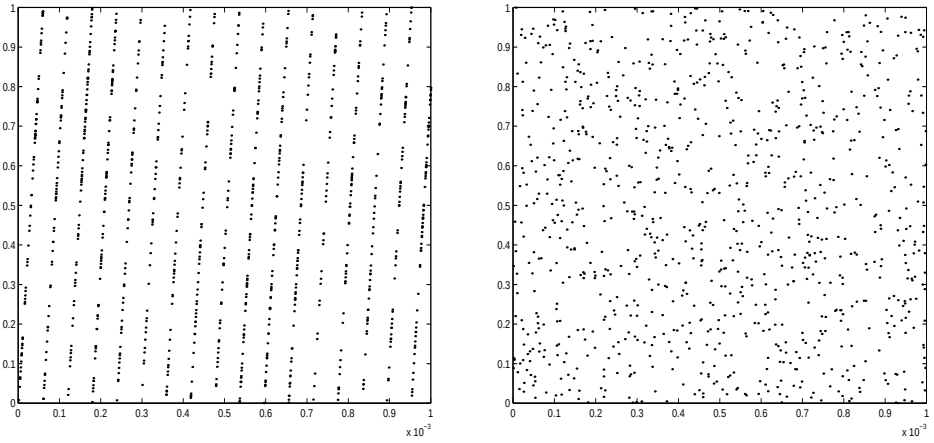


Figure 1.6.2. *Plots of pairs of* 10^6 *random uniform deviates* (U_i, U_{i+1}) *such that* $U_i < 0.0001$. *Left:* MATLAB *4; Right:* MATLAB *5.*

Example 1.6.2.

It is also important to test the serial correlation of the generated numbers. To test the two-dimensional behavior of a random number generator we generated 10^6 pseudo-random numbers U_i. We then placed the numbers each plot (U_i, U_{i+1}) in the unit square. A thin slice of the surface of the square 0.0001 wide by 1.0 high was the cut on its left side and stretched out horizontally. This corresponds to plotting only the pairs (U_i, U_{i+1}) such that $U_i < 0.0001$ (about 1000 points).

In Figure 1.6.2 we show the two plots from the generators in MATLAB 4 and MATLAB 5, respectively. The lattice structure is quite clear in the first plot. With the new generator no lattice structure is visible.

A good generator should have been analyzed theoretically and be supported by practical evidence from extensive statistical and other tests. Knuth [18, Chapter 3], ends his masterly chapter on Random Numbers with the following exercise: *Look at the subroutine library at your computer installation, and replace the random number*

generators by good ones. Try to avoid to be too shocked at what you find. He has in the chapter pointed out both the important ideas, concepts and facts of the topic, and also mentioned some scandalously poor random number generators that were in daily use for decades as standard tools in widely spread computer libraries. Although the generators in daily use have improved, *many are still not satisfactory.* L'Ecuyer [22] writes in 2001:

> *"Unfortunately, despite repeated warnings over the past years about certain classes of generators, and despite the availability of much better alternatives, simplistic and unsafe generators still abound in commercial software."*

1.6.4 Random Deviates for Other Distributions.

We have so far discussed how to generate sequences that behave as if they were random uniform deviates U on $[0, 1)$. By arithmetic operations one can form random numbers with other distributions. A simple example is that $S = a + (b - a)U$ will be uniformly distributed on $[a, b)$. We can also easily generate a random integer between 1 and k; see Example 1.6.1.

Monte Carlo methods often call for other kinds of distributions, for example normal deviates. As we shall see, these can also be generated from a sequence of uniform deviates. Many of the tricks used to do this were originally suggested by John von Neumann in the early 1950s, but have since been improved and refined. We now exemplify, how to use uniform deviates to generate random deviates X for some other distributions.

Discrete Distributions

To make a random choice from a finite number k *equally probable* possibilities is equivalent to generate a random integer X between 1 and k. To do this we take a random deviate U uniformly distributed on $[0, 1)$ multiply by k and take the integer part, and 1, i.e.

$$X = \lceil kU \rceil,$$

where $\lceil x \rceil$ denotes the smallest integer larger than or equal to x.There is a small error because the set of floating point numbers is finite, but this is usually negligible.

In a *more general situation*, we might want to give different probabilities to the values of a variable. Suppose we give the values $X = x_i$, $i = 1 : k$ the probabilities p_i, $i = 1 : k$; note that $\sum p_i = 1$. We can the generate a uniform number U and let

$$X = \begin{cases} x_1, & \text{if } 0 \le U < p_1; \\ x_2, & \text{if } p_1 \le U < p_1 + p_2; \\ \vdots & \\ x_k, & \text{if } p_1 + p_2 + \cdots p_{k-1} \le U < 1. \end{cases}$$

If k is large, and the sequence $\{p_i\}$ is irregular, may require some thought how to find x quickly for a given u. See the analogous question to find a first guess to the root of Equation (1.6.5) below, and the discussion in Knuth [18, Sec. 3.4.1].

A General Transformation from U to X

Suppose we want to generate numbers for a random variable X with a given continuous or discrete distribution function $F(x)$. (In the discrete case, the graph of the distribution function becomes a staircase, see the formulas above.) A general method for this is to solve the equation

$$F(X) = U, \quad \text{or equivalently,} \quad X = F^{-1}(U), \tag{1.6.5}$$

see Figure 1.6.4. Because $F(x)$ is a nondecreasing function, and $Pr\{U \leq u\} = u, \forall u \in [0,1]$, equation (1) is proved by the line

$$Pr\{X \leq x\} = Pr\{F(X) \leq F(x)\} = Pr\{U \leq F(x)\} = F(x).$$

How to solve (1.6.5) fast is often a problem with this method, and for some distributions we shall see better methods below.

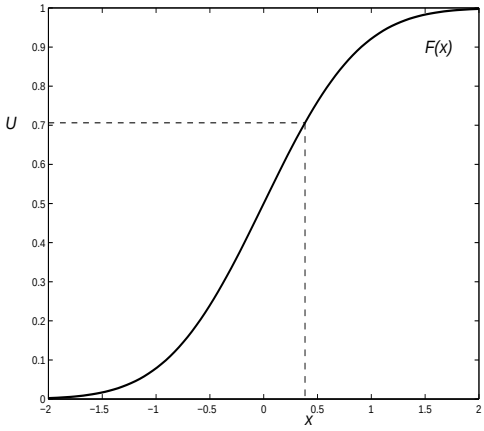


Figure 1.6.3. *Random number with distribution $F(x)$.*

Exponential Deviates.

As an example consider the exponential distribution with parameter $\lambda > 0$. This distribution occurs in queuing problems, e.g., in tele-communication, to model arrival and service times. The important property is that the intervals of time between two successive events are a sequence of exponential deviates. The exponential distribution with mean $1/\lambda$ has density function $f(t) = \lambda e^{-\lambda t}$, $t > 0$, and distribution function

$$F(x) = \int_0^x \lambda e^{-\lambda t}\, dt = 1 - e^{-\lambda x}. \tag{1.6.6}$$

Using the general rule given above, exponentially distributed random numbers X can be generated as follows: Let U be a uniformly distributed random number in $[0,1]$. Solving the equation $1 - e^{-\lambda X} = U$, we obtain

$$X = -\lambda^{-1} \ln(1 - U).$$

A drawback of this method is that the evaluation of the logarithm is relatively slow.

One important use of exponentially distributed random numbers is in the generation of so-called **Poisson processes**. Such processes are often fundamental in models of telecommunications systems and other service systems. A Poisson process with frequency parameter λ is a sequence of events characterized by the property that the probability of occurrence of an event in a short time interval $(t, t+\Delta t)$ is equal to $\lambda\cdot\Delta t + o(\Delta t)$, independent of the sequence of events previous to time t. In applications an "event" can mean a call on a telephone line, the arrival of a customer in a store, etc. For simulating a Poisson process one can use the important property that the intervals of time between two successive events are independent exponentially distributed random numbers.

Normal Deviates.

A normal deviate N is a random variable with zero mean and unit standard deviation, and has the density function

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}, \quad (m = 0,\ \sigma = 1).$$

A normal deviate with mean m and standard deviation σ is $m + \sigma N$; the density function is $(1/\sigma)f((x-m)/\sigma)$. The normal distribution function

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt = \frac{1}{2}\Big(1 + \operatorname{erf}\Big(\frac{x}{\sqrt{2}}\Big)\Big)$$

is related to the error function $\operatorname{erf}(x)$ introduced in Sec. 1.3.4 and is not an elementary function, In this case solving the equation (1.6.5) is time consuming.

Fortunately random normal deviates can be obtained in easier ways. In the **polar algorithm** a random point in the unit circle is generated as follows. Let U_1, U_2 be two independent, uniformly distributed random numbers on $[0, 1]$. Then the point (V_1, V_2), where $V_i = 2U_i - 1$, $i = 1, 2$, is uniformly distributed in the square $[-1, 1] \times [-1, 1]$. We compute $S = V_1^2 + V_2^2$ and reject the point if it outside the unit circle, i.e. if $S > 1$. The remaining points are uniformly distributed on the unit circle.

For each accepted point we form

$$N_1 = V_1\sqrt{\frac{-2\ln S}{S}}, \qquad N_2 = V_2\sqrt{\frac{-2\ln S}{S}}. \tag{1.6.7}$$

It can be proved that N_1, N_2 are two independent, normally distributed random numbers with mean zero and standard deviation 1. We point out that N_1, N_2 can be considered to be rectangular coordinates of a point whose polar coordinates (r, ϕ) are determined by the equations

$$r^2 = N_1^2 + N_2^2 = -2\ln S, \quad \cos\phi = U_1/\sqrt{S}, \quad \sin\phi = U_2/\sqrt{S}.$$

Thus the problem is to show that the distribution function for a pair of independent, normally distributed random variables is rotationally symmetric (uniformly

distributed angle) and that their sum of squares is exponentially distributed with mean 2; see Knuth [18, p. 123].

The polar algorithm, which was used for MATLAB 4, is moderately expensive. First, about $(1 - \pi/4) = 21.5\%$ of the uniform numbers are rejected because the generated point falls outside the unit circle. Further, the calculation of the logarithm contributes significantly to the cost. From MATLAB 5 on a more efficient table look-up algorithm developed by Marsaglia and Tsang [27] is used. This is called the "ziggurat" algorithm after the name of ancient Mesopotamian terraced temples mounds, that look like two-dimensional step functions. A popular description of the ziggurat algorithm is given by Moler [30]; see also [17].

Chi-square Distribution

The chi-square distribution function $P(\chi^2, n)$ is related to the incomplete gamma function (see Sec. 3.?? by

$$P(\chi^2, n) = (n/2, \chi^2/2). \tag{1.6.8}$$

Its complement $Q(\chi^2, n) = 1 - P(\chi^2, n)$ is the probability that the observed chi-square will exceed the value χ^2 even for a correct model. Subroutines for evaluating the χ^2-distribution function as well as other important statistical distribution functions are given in [34, Sec. 6.2–6.3].

Numbers belonging to the **chi-square distribution** can also be obtained by using the definition of the distribution. If $N_1, N_2, \ldots, N_n$ are normal deviates with mean 0 and variance 1, the number

$$Y_n = N_1^2 + N_2^2 + \cdots + N_n^2$$

is distributed as χ^2 with n degrees of freedom.

Other Distributions

Methods to generate random deviates with, e.g., Poisson, gamma and binomial distribution, are described in Knuth [18, Sec. 3.4]) and Press et al. [34, Chapter 7.3]. A general method, introduced by G. Marsaglia [24], is the **rectangle-wedge-tail method**. It been further developed and applied by Marsaglia and coauthors, see references in Knuth [18, Sec. 3.4]). The **rejection method** is based on ideas of von Neumann.Several authors, notably G. Marsaglia, have developed powerful combinations of rejection methods and the rectangle-wedge-tail method.

1.6.5 Reduction of Variance.

From statistics, we know that if one makes n independent observations of a quantity whose standard deviation is σ, then the standard deviation of the mean is $\sigma/\sqrt{n}$. Hence to increase the accuracy by a factor of 10 (say) we have to increase the number of experiments n by a factor 100.

Example 1.6.3.

In 1777 Buffon[19] carried out a probability experiment by throwing sticks over his shoulder onto a tiled floor and counting the number of times the sticks fell across the lines between the tiles. He stated that the favourable cases correspond "to the area of part of the cycloid whose generating circle has diameter equal to the length of the needle". To simulate Buffon's experiment we suppose a board is ruled with equidistant parallel lines and that a needle fine enough to be considered a segment of length l not longer than the distance d between consecutive lines is thrown on the board. The probability is then $2l/(\pi d)$ that it will hit one of the lines.

The Monte Carlo method and this game can be used to approximate the value of π. Take the distance δ between the center of the needle and the lines and the angle ϕ between the needle and the lines to be random numbers. By symmetry we can choose these to be rectangularly distributed on $[0, d/2]$ and $[0, \pi/2]$, respectively. Then the needle hits the line if $\delta < (l/2)\sin\phi$.

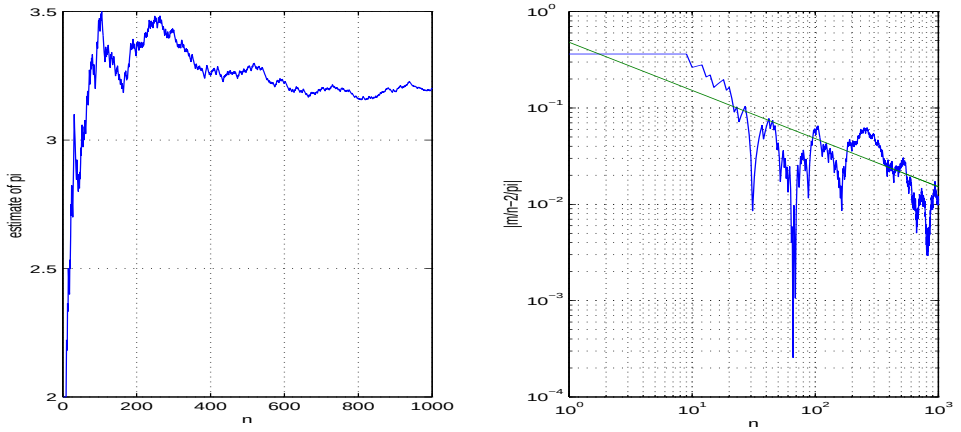


Figure 1.6.4. *The left part shows how the estimate of π varies with the number of throws. The right part compares $|m/n - 2/\pi|$ with the standard deviation of m/n. The latter is inversely proportional to $n^{1/2}$, and is therefore a straight line in the figure.*

We took $l = d$. Let m be the number of hits in the first n throws in a Monte Carlo simulation with 1000 throws. The expected value of m/n is therefore $2/\pi$, and so $2n/m$ is an estimate of π after n throws. In the left part of Fig. 1.5.3 we see, how $2n/m$ varies with n in one simulation. The right part compares $|m/n - 2/\pi|$ with the standard deviation of m/n, which equals $\sqrt{2/\pi(1-2/\pi)/n}$ and is, in the log-log-diagram, represented by a straight line, the slope of which is $-1/2$. This can be taken as a test that the random number generator in MATLAB is behaving correctly! (The spikes, directed downwards in the figure, typically indicate where

[19] Compte de Buffon (1707–1788), French natural scientist that contributed to the understanding of probability. He also computed the probability that the sun would continue to rise after having been observed to rise on n consecutive days.

$m/n - 2/\pi$ changes sign.)

A more efficient way than increasing the number of samples, often is to instead try to decrease the value of σ by redesigning the experiment in various ways. Assume that one has two ways (which require the same amount of work) of carrying out an experiment, and these experiments have standard deviations σ_1 and σ_2 associated with them. If one repeats the experiments n_1 and n_2 times (respectively), the same precision will be obtained if $\sigma_1/\sqrt{n_1} = \sigma_2/\sqrt{n_2}$, or

$$n_1/n_2 = \sigma_1^2/\sigma_2^2. \tag{1.6.9}$$

Thus if a variance reduction by a factor k can be achieved, then the number of experiments needed is also reduced by the same factor k.

An important means of reducing the variance of estimates obtained from the Monte Carlo method is to use **antithetic sequences**. For example, if U_i is a series of random uniform deviates on $[0, 1]$ then $U_i' = 1 - U_i$ are also uniformly distributed on $[0, 1]$. For example, from the sequence in (1.6.1) we get the sequence

$$0.443065, 0.970545, 0.182765, 0.564115, 0.186495, 0.236975, \ldots, \tag{1.6.10}$$

which is the antithetic sequence derived from (1.6.1). Antithetic sequences of normally distributed numbers are obtained simply by reversing the sign of the original sequence.

Roughly speaking, since the influence of chance has opposing effects in the two antithetic experiments, one can presume that the effect of chance on the *means* is much less than the effect of chance in the original experiments. In the following example we show how to make a quantitative estimate of the reduction of variance accomplished with the use of antithetic experiments.

Example 1.6.4.

Suppose the numbers x_i are the results of statistically independent measurements of a quantity with expected value m, and standard deviation σ. Set

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

Then $\bar{x}$ is an estimate of m, and $s/\sqrt{n-1}$ is an estimate of σ.

In ten simulation and their antithetic experiments of a service system the following values were obtained for the treatment time:

$$685 \quad 1,045 \quad 718 \quad 615 \quad 1,021 \quad 735 \quad 675 \quad 635 \quad 616 \quad 889\,.$$

From this experiment the mean for the treatment time is estimated as $763,4$, and the standard deviation 51.5, which we write 763 ± 52. Using an anthitetic series, one got the following values:

$$731 \quad 521 \quad 585 \quad 710 \quad 527 \quad 574 \quad 607 \quad 698 \quad 761 \quad 532\,.$$

Table 1.6.1. *Simulation of patients at a polyclinic.*

	$k=1$					$k=2$	
P_{no}	P_{arr}	T_{beg}	R	T_{time}	T_{end}	P_{arr}	T_{end}
1	0*	0	211	106	106	0*	106
2	50	106	3	2	108	0	108
3	100	108	53	26	134	50	134
4	150*	150	159	80	230	100	214
5	200	230	24	12	242	150	226
6	250*	250	35	18	268	200	244
7	300*	300	54	27	327	250*	277
8	350*	350	39	20	370	300*	320
9	400*	400	44	22	422	350*	372
10	450*	450	13	6	456	400*	406
Σ	2,250			319	2,663	1,800	2,407

The series means is thus

$$708 \quad 783 \quad 652 \quad 662 \quad 774 \quad 654 \quad 641 \quad 666 \quad 688 \quad 710\,,$$

from which one gets the estimate 694 ± 16.

When one instead supplemented the first sequence with ten values using independent random numbers, the estimate 704 ± 36 using all twenty values was obtained. These results indicate that, in this example, using antithetical sequence produces the desired accuracy with $(16/36)^2 \approx 1/5$ of the work required if completely independent random numbers are used. This rough estimate of the work saved is uncertain, but indicates that it is profitable to use the technique of anthitetic series.

Example 1.6.5.

Monte Carlo methods have been successfully used to study queuing problems. A well known example is a study by Bailey [3] to determine how to give appointment times to patients at a polyclinic. The aim is to find a suitable balance between the mean waiting times of both patients and doctors. This problem was in fact solved analytically—much later—after Bailey already had gotten the results that he wanted; this situation is not uncommon when numerically methods (and especially Monte Carlo methods) have been used.

Suppose that k patients have been booked at the time $t = 0$ (when the clinic opens), and that the rest of the patients (altogether 10) are booked at intervals of 50 time units thereafter. The time of treatment is assumed to be exponentially distributed with mean 50. (Bailey used a distribution function which was based on empirical data.) Three alternatives, $k = 1, 2, 3$, are to be simulated. *By using the same random numbers for each k (hence the same treatment times) one gets a* **reduced variance** *in the estimate of the change in waiting times as k varies.*

The computations are shown in the Table 1.5.1. The following abbreviations are used: P = patient, D = doctor, T = treatment. An asterisk indicates that the patient did not need to wait. In the table P_{arr} follows from the rule for booking patients given previously. The treatment time T_{time} equals $50R/100$ where R are exponentially distributed numbers with mean 100 taken from a table. T_{beg} equals the larger of the number P_{arr} (on the same row) and T_{end} (in the row just above), where $T_{end} = T_{beg} + T_{treat}$.

From the table we find that for $k = 1$ the doctor waited the time $D = 456 - 319 = 137$; the total waiting time for patients was $P = 2,663 - 2,250 - 319 = 94$. For $k = 2$ the corresponding waiting times were $D = 406 - 319 = 87$ and $P = 2,407 - 1,800 - 319 = 288$. Similar calculations for $k = 3$ gave $D = 28$ and $P = 553$ (see Fig. 1.5.5). For $k \geq 4$ the doctor never needs to wait.

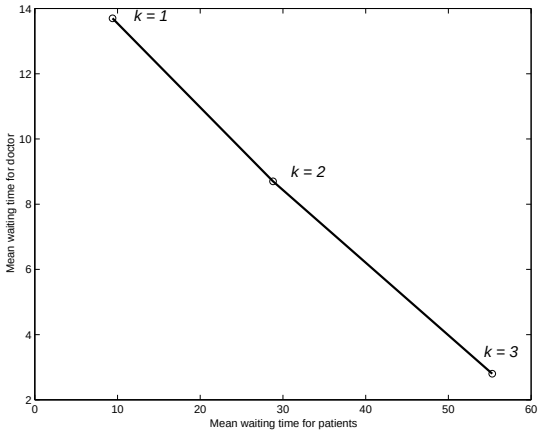


Figure 1.6.5. *Mean waiting times for doctor/patients at polyclinic.*

One cannot, of course, draw any tenable conclusions from one experiment. More experiments should be made in order to put the conclusions on statistically solid ground. Even isolated experiments, however, can give valuable suggestions for the planning of subsequent experiments, or perhaps suggestions of appropriate approximations to be made in the analytic treatment of the problem. The large-scale use of Monte Carlo methods requires careful planning to avoid drowning in in enormous quantities of unintelligible results.

Two methods for **reduction of variance** have here been introduced: *antithetic sequence of random numbers* and the technique of using *the same random numbers in corresponding situations*. The latter technique is used when studying the changes in behavior of a system when a certain parameter is changed (e.g., the parameter k in Exercise 4). (Note that we just have restart the RNG using the same seed.)

Many effective methods have been developed for reducing variance, e.g., **importance sampling** and **splitting techniques** (see Hammersley and Handscomb [15]).

Review Questions

1. What is a uniformly distributed random number?
2. Describe a general method for obtaining random numbers with a given discrete or continuous distribution. Give examples of their use.
3. What are the most important properties of a Poisson process? How can one generate a Poisson process with the help of random numbers?
4. What is the mixed congruential method for generating pseudo-random numbers? What important difference is there between the numbers generated by this method and "genuine" random numbers?
5. Give three methods for reduction of variance in estimates made with the Monte Carlo method, and explain what is meant by this term. Give a quantitative connection between reducing variance and decreasing the amount of computation needed in a given problem?

Problems and Computer Exercises

1. (C. Moler) Consider the toy random number generator, $x_i = ax_i \bmod m$, with $a = 13$, $m = 31$ and start with $x_0 = 1$. Show that this generates a sequence consisting of a permutation of all integers from 1 to 30, and then repeats itself. Thus this generator has the period equal to $m-1 = 30$, equal to the maximum possible.
2. Simulate (say) 360 throws with two usual dices. Denote the sum of the number of dots on the two dice in the n'th throw by Y_n, $2 \le Y_n \le 12$. Tabulate or draw a histogram, i.e., the (absolute) frequency of the occurrence of j dots versus j, $j = 2 : 12$. Make a conjecture about the true value of $P(Y_n = j)$. Try to confirm it by repeating the experiment with fresh uniform random numbers. When you have found the right conjecture, it is not hard to prove it.
3. (a) Let X, Y be independent uniform random numbers on the interval $[0, 1]$. Show that $P(X^2 + Y^2 \le 1) = \pi/4$, and estimate this probability by a Monte Carlo experiment with (say) 1000 pairs of random numbers. For example, make graphical output like in the Buffon needle problem.

   (b) Make an antithetic experiment, and take the average of the two results. Is the average better than one can expect if the second experiment had been independent of the first one.

   (c) Estimate similarly the volume of the four-dimensional unit ball. If you have enough time, use more random numbers. (The exact volume of the unit ball is $\pi^2/2$.)
4. A famous result by P. Diaconis asserts that it takes approximately $\frac{3}{2}\log_2 52 \approx 8.55$ riffle shuffles to randomize a deck of 52 cards, and that randomization occurs abruptly according to a "cutoff phenomenon". (For example, after six shuffles the deck is still far from random.)

The following defintion can be used for simulating a riffle shuffle. The deck of cards is first cut roughly in half according to a binomial distribution, i.e. the probabiity that ν cards are cut is $\frac{n}{\nu}/2^n$. The two halfs are then riffled to gether by dropping cards roughly alternately from each half onto a pile, with the probability of a card being dropped from each half being proportional to the number of cards in it.

Write a program that uses uniform random numbers (and perhaps uses the the formula $X = \lceil kR \rceil$ for several values of k) to simulate a random "shuffle" of a deck of 52 cards according to the above precise definition. This is for a *numerical* game; do not spend time on drawing beautiful hearts, clubs etc.

5. Brownian motion is the irregular motion of dust particles suspended in a fluid, being bombarded by molecules in a random way. Generate two sequences of random normal deviates a_i and b_i, and use these to simulate Brownian motion by generating a path defined by the points (x_i, y_j), where $x_0 = y_0 = 0$, $x_i = x_{i-1} + a_i$, $y_i = y_{i-1} + b_i$. Plot each point and connect the points with a straight line to visualize the path.

6. Repeat the simulation in the queuing problem in Example 1.6.5 for $k = 1$ and $k = 2$ using the sequence of exponentially distributed numbers R

$$13 \quad 365 \quad 88 \quad 23 \quad 154 \quad 122 \quad 87 \quad 112 \quad 104 \quad 213\,,$$

antithetic to that used in Example 1.6.5. Compute the mean of the waiting times for the doctor and for all patients for this and the previous experiment.

7. A target with depth $2b$ and very large width is to be shot at with a cannon. (The assumption that the target is very wide makes the problem one-dimensional.) The distance to the center of the target is unknown, but estimated to be D. The difference between the actual distance and D is assumed to be a normally distributed variable X with zero mean and standard deviation σ_1.

One shoots at the target with a salvo of three shots, which are expected to travel a distance $D - a$, D and $D + a$, respectively. The difference between the actual and the expected distance traveled is assumed to be a normally distributed random variable with zero mean and standard deviation σ_2; the resulting error component in the three shots is denoted by Y_{-1}, Y_0, Y_1. We further assume that these three variables are stochastically independent of each other and X.

One wants to know how the probability of at least one "hit" in a given salvo depends on a and b. Use normally distributed pseudo-random numbers to shoot ten salvos and determine for each salvo, the least value of b for which there is at least one "hit" in the salvo. Show that this is equal to

$$\min_k |X - (Y_k + ka)|, \quad k = -1, 0, 1.$$

Fire an "antithetic salvo" for each salvo.

Graph using $\sigma_1 = 3$, $\sigma_2 = 1$, for both $a = 1$ and $a = 2$ using the same random numbers curves, which give the probability of a hit as a function of the depth of the target.

Notes and References

A good paper explaining to a mathematical audience problems inherent in numerical computations is Forsythe [10, 1970]. Fox [13, 1971] gives numerous examples in which incorrect answers are obtained from plausible numerical methods.

Many of the methods and problems introduced in this introductory chapter will be studied in more detail in later chapters and volumes. In particular, methods for solving a single nonlinear equation are found in Chapter 6, and corresponding methods for nonlinear systems of equations are treated in Chapter 12, Volume II. Gaussian elimination and iterative methods for linear systems will be covered in Volume II, Chapters 7 and 10, respectively. The numerical solution of ordinary differential equations is treated in depth in Volume III, Chapter 13.

An good source of information on random numbers is Knuth [18]. Another comprehensive reference is the monograph by Niederreiter [31, 1992]. An overview more suited for applications is found in Press et al. [34, Chapter 7]. Guidelines for choosing a good RNG are given in Marsaglia [26] and L'Ecuyer [19]. Recent progress in random number generation and testing is found in the two authoritative papers by L'Ecuyer [20, 21]. Hellekalek [16] explains the art to access random number generators for practitioners.

Computational aspects of numerical linear algebra will be treated in Volume II. For an elementary introduction to Linear Algebra we refer to one of several good textbooks, e.g., Leon [23], Strang [37].

The historical developments of Numerical Analysis in the 20th Century is surveyed in [5]. An eloquent essay on the foundations of computational mathematics is found in [4].

Bibliography

[1] Milton Abramowitz and Irene A. Stegun (eds.). *Handbook of Mathematical Functions*. Dover, New York, NY, 1965.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, editors. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[3] N. T. J. Bailey. A study of queues and appointment systems in hospital outpatient departments, with special reference to waiting times. *J. Roy. Stat. Soc.*, 3:14:185ff, 1951.

[4] B. J. C. Baxter and Arieh Iserles. On the foundations of computational mathematics. In P. G. Ciarlet and F. Cucker, editors, *Handbook of Numerical Analysis*, pages 3–34. North Holland Elsevier, Amsterdam, 2002.

[5] Claude Brezinski and Luc Wuytack. Numerical analysis in the twentieth century. In Claude Brezinski and L. Wuytack, editors, *Numerical Analysis: Historical Developments in the 20th Century*, pages 1–40. North Holland Elsevier, Amsterdam, 2001.

[6] RAND Corporation. *A Million Random Digits and 100,000 Normal Deviates*. Free Press, Glencoe, IL, 1955.

[7] Germund Dahlquist. Preliminär rapport om premieobligationsdragning med datamaskin. (in swedish), Riksgäldskontoret, Stockholm, 1962.

[8] Germund Dahlquist and Åke Björck. *Numerical Methods*. Dover, Mineola, NY, 2004.

[9] Terje O. Espelid. On floating-point summation. *SIAM Review*, 37:603–607, 1995.

[10] George E. Forsythe. Pitfalls in computation, or why a math book isn't enough. Technical Report CS 147, Computer Science Department, Stanford University, Stanford, CA, 1970.

[11] George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler. *Computer Methods for Mathematical Computations*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[12] George E. Forsythe and Cleve B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[13] Leslie Fox. How to get meaningless answers in scientific computation (and what to do about it),. *IMA Bulletin*, 7:10:296–302, 1971.

[14] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[15] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Methuen, London, UK, 1964.

[16] Peter Hellekalek. Good random number generators are (not so) easy to find. *Math. Comput. Simulation*, 46:485–505, 1998.

[17] David Kahaner, Cleve B. Moler, and Stephen Nash. *Numerical Methods and Software*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[18] Donald E. Knuth. *The Art of Computer Programming, Vol. 2. Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1997.

[19] Pierre L'Ecuyer. Efficient and portable combined random number generators. *Comm. ACM*, 31:6:742–774, 1988.

[20] Pierre L'Ecuyer. Random number generation. In J. Banks, editor, *The Handbook of Simulation*, pages 485–505. John Wiley, New York, 1998.

[21] Pierre L'Ecuyer. Uniform random number generation. In *Proc. 1998 Winter Simulation Conference*, pages 97–104. IEEE Press, Pistacaway, NJ, 1998.

[22] Pierre L'Ecuyer. Software for uniform random number generation: Distinguishing the good and bad. In *Proc. 2001 Winter Simulation Conference*, pages 95–105. IEEE Press, Pistacaway, NJ, 2001.

[23] Steven J. Leon. *Linear Algebra with Applications*. Macmillan, New York, fourth edition, 1994.

[24] George Marsaglia. Expressing a random variable in terms of uniform random variables. *Ann. Math. Stat.*, 32:894–898, 1961.

[25] George Marsaglia. Random numbers falls mainly in the planes. *Proc. Nat. Acad. Sci.*, 60:5:25–28, 1968.

[26] George Marsaglia. A current view of random number generators. In L. Billard, editor, *Computer Science and Statistics: The Interface*, pages 3–10. Elsevier Science Publishers, Amsterdam, 1985.

[27] George Marsaglia and W. W. Tsang. A fast, easily implemented method for sampling from decreasing or symmetric unimodal density functions. *SIAM J. Sci. Stat. Comput.*, 5:2:349–360, 1984.

[28] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Modeling Comput. Software*, 8:1:3–30, 1998.

[29] Cleve Moler. Random thoughts, 10^{435} years is a very long time. *MATLAB News and Notes*, Fall, 1995.

[30] Cleve Moler. Normal behavior. *MATLAB News and Notes*, Spring, 2001.

[31] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, PA, 1992.

[32] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Comm. ACM*, 22:1192–1201, 1988.

[33] K. Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Phil. Mag. Series 5*, 50:p. 157–175, 1900.

[34] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in Fortran; The Art of Scientific Computing*. Cambridge University Press, Cambridge, GB, second edition, 1992.

[35] Lewis F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with application to the stress in a masonry dam. *Philos. Trans. Roy. Soc.*, A210:307–357, 1910.

[36] George W. Stewart. *Matrix Algorithms Volume I: Basic Decompositions*. SIAM, Philadelphia, PA, 1998.

[37] Gilbert Strang. *Linear Algebra and Its Applications*. Academic Press, New York, fourth edition, 2005.

[38] V. Strassen. Gaussian elimination is not optimal. *Numer. Math..*, 13:354–356, 1969.

Index

algorithm
    back-substitution, 28
    Gaussian elimination, 30
antithetic sequence, 63

back-substitution, 27
band matrix, 31
BLAS, 36

cancellation, 12

deflation, 14
density function, 51
determinant, 30
difference approximation, 9–22
    centered, 9
difference scheme, 11
discrete distributions, 58
discretization error, 9
distribution function, 51
divide and conquer, 17
divide and conquer strategy, 16

erf(x), 18
error function, 18
Euclid's algorithm, 23
Euler's method, 39, 41
exponential distribution, 59
exponential integral, 18

forward-substitution, 27
full matrix, 34

gamma function
    incomplete, 18
Gaussian elimination, 28

Hessenberg matrix, 38

Horner's rule, 14

importance sampling, 65
iteration
    fixed point, 5

Jacobian matrix, 7

linear congruential generator, 54
linear interpolation, 7
linear system
    overdetermined, 20
linearization, 5

matrix
    tridiagonal, 32
mean, 51
Mersenne twister, 55
Monte Carlo Methods, 49–68
multiple recursive generator, 53

Newton's method, 5
normal distribution function, 60
normal equations, 21
normal probability function, 18
numerical instability, 16
numerical integration
    trapezoidal rule, 7
numerical simulation, 39

operation count, 26

pivotal elements, 29
pivoting
    partial, 32
Poisson process, 61
polar algorithm, 60
pseudo-random numbers, 51–61

random
    normal deviates, 60
random numbers, 51–61
    antithetic sequence of, 63
    generating, 53
    uniformly distributed, 52
rectangle-wedge-tail method, 61
recursion formula, 39
reduction of variance, 61–65
rejection method, 61
residual vector, 20
Richardson extrapolation, 8, 43
Richardson's method, 35
Romberg's method, 9

secant method, 7
sparse matrix, 34
splitting technique, 65
square root
    fast method, 4
standard deviation, 51
successive approximation, 2
synthetic division, 14

trapezoidal rule, 7
triangular
    systems of equations, 26–28
tridiagonal system
    algorithm, 32
truncation error, 19

variance, 51
    reduction of, 61–65

Contents

2 How to Obtain and Estimate Accuracy **1**

2.1 Basic Concepts in Error Estimation 1
2.1.1 Sources of Error 1
2.1.2 Absolute and Relative Errors 4
2.1.3 Rounding and Chopping 5
Review Questions 7
2.2 Computer Number Systems 7
2.2.1 The Position System 7
2.2.2 Fixed and Floating Point Representation 10
2.2.3 IEEE Floating Point Standard 13
2.2.4 Elementary Functions 17
2.2.5 Multiple Precision Arithmetic 18
Review Questions 19
Problems and Computer Exercises 20
2.3 Accuracy and Rounding Errors 21
2.3.1 Floating Point Arithmetic 21
2.3.2 Basic Rounding Error Results 27
2.3.3 Statistical Models for Rounding Errors 31
2.3.4 Avoiding Overflow 33
2.3.5 Cancellation of Terms 34
Review Questions 37
Problems 37
Computer Exercises 39
2.4 Error Propagation 41
2.4.1 Numerical Problems, Methods and Algorithms . . . 41
2.4.2 Propagation of Errors 43
2.4.3 Condition Numbers of Problems 47
2.4.4 Perturbation Analysis for Linear Systems 50
2.4.5 Forward and Backward Error Analysis 52
2.4.6 Stability of Algorithms 53
Review Questions 58
Problems 58
Problems and Computer Exercises 60
2.5 Automatic Control of Accuracy and Verified Computing 61

2.5.1 Running Error Analysis 61
2.5.2 Experimental Perturbations 62
2.5.3 Introduction to Interval Arithmetic 63
Review Questions 71
Problems 71

Bibliography **73**

Index **77**

Chapter 2

How to Obtain and Estimate Accuracy

2.1 Basic Concepts in Error Estimation

The main purpose of numerical analysis and scientific computing is to develop efficient and accurate methods to compute approximations to quantities that are difficult or impossible to obtain by analytic means. It has been convincingly argued (N. Trefethen [43]) that controlling rounding errors is just a small part of this, and that the main business of computing is the development of algorithms that converge fast. Even if we acknowledge the truth of this statement, it is still necessary to be able to control different sources of errors, including round-off errors, so that these will not interfere with the computed results.

2.1.1 Sources of Error

Numerical results are affected by many types of errors. Some sources of error are difficult to influence; others can be reduced or even eliminated by, for example, rewriting formulas or making other changes in the computational sequence. Errors are propagated from their sources to quantities computed later, sometimes with a considerable amplification or damping. It is important to distinguish between the new error produced at the computation of a quantity (a source error), and the error inherited (propagated) from the data that the quantity depends on.

A. *Errors in Given Input Data.* Input data can be the result of measurements which have been influenced by systematic errors or by temporary disturbances. A **rounding error** occurs, for example, whenever an irrational number is shortened ("rounded off") to a fixed number of decimals. It can also occur when a decimal fraction is converted to the form used in the computer.

B. *Rounding Errors During the Computations.* The limitation of floating point numbers in a computer leads at times to a loss of information that, depending on the context, may or may not be important. Two typical cases are:

(i) If the computer cannot handle numbers which have more than, say, s digits, then the exact product of two s-digit numbers (which contains $2s$ or $2s-1$ digits) cannot be used in subsequent calculations; the product must be rounded off.

(ii) If, in a floating point computation, a relatively small term b is added to a, then some digits of b are "shifted out" (see Example 2.3.1, and they will not have any effect on future quantities that depend on the value of $a+b$.

The effect of such rounding can be quite noticeable in an extensive calculation, or in an algorithm which is numerically unstable (see Example 1.3.1).

C. *Truncation Errors.* These are errors committed when a limiting process is truncated (broken off) before one has come to the limiting value. A **truncation error** occurs, for example, when an infinite series is broken off after a finite number of terms, or when a derivative is approximated with a difference quotient (although in this case the term **discretization error** is better). Another example is when a nonlinear function is approximated with a linear function as in Newton's method. Observe the distinction between truncation error and rounding error.

D. *Simplifications in the Mathematical Model.* In most of the applications of mathematics, one makes idealizations. In a mechanical problem, for example, one might assume that a string in a pendulum has zero mass. In many other types of problems it is advantageous to consider a given body to be homogeneously filled with matter, instead of being built up of atoms. For a calculation in economics, one might assume that the rate of interest is constant over a given period of time. The effects of such sources of error are usually more difficult to estimate than the types named in A, B, and C.

E. *"Human" Errors and Machine Errors.* In all numerical work, one must expect that clerical errors, errors in hand calculation, and misunderstandings will occur. One should even be aware that textbooks (!), tables and formulas may contain errors. When one uses computers, one can expect errors in the program itself, typing errors in entering the data, operator errors, and (more seldom) pure machine errors.

Errors which are purely machine errors are responsible for only a very small part of the strange results which (occasionally with great publicity) are produced by computers. Most of the errors depend on the so-called human factor. As a rule, the effect of this type of error source cannot be analyzed with the help of the theoretical considerations of this chapter! We take up these sources of error in order to emphasize that both the person who carries out a calculation and the person who guides the work of others can plan so that such sources of error are not damaging. One can reduce the risk for such errors by suitable adjustments in working conditions and routines. Stress and tiredness are common causes of such errors.

Intermediate results that may reveal errors in a computation are not visible when using a computer. Hence the user must be able to verify the correctness of

his results or be able to prove that his process cannot fail! Therefore one should carefully consider what kind of checks can be made, either in the final result or in certain stages of the work, to prevent the necessity of redoing a whole project for the sake of a small error in an early stage. One can often discover whether calculated values are of the wrong order of magnitude or are not sufficiently regular, for example using difference checks (see Sec. 4.5).

Occasionally one can check the credibility of several results at the same time by checking that certain relations are true. In linear problems, one often has the possibility of sum checks. In physical problems, one can check, for example, to see whether energy is conserved, although because of the error sources A–D one cannot expect that it will be exactly conserved. In some situations, it can be best to treat a problem in two independent ways, although one can usually (as intimated above) check a result with less work than this.

Errors of type E do occur, sometimes with serious consequences. For example, the first American Venus probe was lost due to a program fault caused by the inadvertent substitution of a statement in a Fortran program of the form `DO 3 I = 1.3` for one of the form `DO 3 I = 1,3`.[1] A hardware error that got much publicity surfaced in 1994, when it was found that the INTEL Pentium processor gave wrong results for division with floating point numbers of certain patterns. This was discovered during research on prime numbers; see Edelman [18].

From a different point of view, one may distinguish between controllable and uncontrollable (or unavoidable) error sources. Errors of type A and D are usually considered to be uncontrollable in the numerical treatment (although a feedback to the constructor of the mathematical model may sometimes be useful). Errors of type C are usually controllable. For example, the number of iterations in the solution of an algebraic equation, or the step size in a simulation can be chosen, either directly or by setting a tolerance, see Sec. 1.4.3.

The rounding error in the individual arithmetic operation (type B) is, in a computer, controllable only to a limited extent, mainly through the choice between single and double precision. A very important fact is, however, that it can often be controlled by appropriate rewriting of formulas or by other changes of the algorithm, see, e.g., Example 2.3.3.

If it doesn't cost too much, a controllable error source should be controlled so that its effects are evidently negligible, for example compared to the effects of the uncontrollable sources. A reasonable interpretation of "full accuracy" is that the controllable error sources should not increase the error of a result more than about 20%. Sometimes, "full accuracy" may be expensive, for example in terms of computing time, memory space or programming efforts. Then it becomes important to estimate the relation between accuracy and these cost factors. One goal of the rest of this chapter is to introduce concepts and techniques useful to this purpose.

Many real-world problems contain some non-standard features, where understanding the general principles of numerical methods can save much time in the preparation of a program as well as in in the computer runs. Nevertheless, we

[1] The erroneous replaced symbol "," with "." converts the intended loop statement into an assignment statement!

strongly encourage the reader to use quality library programs whenever possible, since a lot of experience and profound theoretical analysis has often been built into these (sometimes far beyond the scope of this text). It is not practical to "reinvent the wheel"!

2.1.2 Absolute and Relative Errors

Approximation is a central concept in almost all the uses of mathematics. One must often be satisfied with approximate values of the quantities with which one works. Another type of approximation occurs when one ignores some quantities which are small compared to others. Such approximations are often necessary to insure that the mathematical and numerical treatment of a problem does not become hopelessly complicated.

We make the following definition.

Definition 2.1.1.

Let $\tilde{x}$ be an approximate value whose exact value is x. Then the **absolute error** *in $\tilde{x}$ is:*

$$\Delta x = \tilde{x} - x,$$

and if $x \neq 0$ the **relative error** *is:*

$$\Delta x/x = (\tilde{x} - x)/x.$$

In some books the error is defined with the opposite sign to that we use here. It makes almost no difference which convention one uses, as long as one is consistent. Using our definition $x - \tilde{x}$ is the correction which should be *added* to $\tilde{x}$ to get rid of the error. The correction and the error have then the same magnitude but different sign.

It is important to distinguish between the error $\tilde{x} - x$, which can be positive or negative, and a *bound* for the magnitude of the error. In many situations one wants to compute strict or approximate **error bounds** for the absolute or relative error. Since it is sometimes rather hard to obtain an error bound that is both strict and sharp, one sometimes prefers to use less strict but often realistic **error estimates**. These can be based on the first neglected term in some expansion or some other asymptotic considerations.

The notation $x = \tilde{x} \pm \epsilon$ means, in this book, $|\tilde{x} - x| \leq \epsilon$. For example, if $x = 0.5876 \pm 0.0014$ then $0.5862 \leq x \leq 0.5890$, and $|\tilde{x} - x| \leq 0.0014$. In other texts, the same plus-minus notation is sometimes used for the "standard error" (see Sec. 2.3.3) or some other measure of deviation of a statistical nature. If x is a vector $\|\cdot\|$ then the error bound and the relative error bound may be defined as bounds for

$$\|\tilde{x} - x\| \quad \text{and} \quad \|\tilde{x} - x\|/\|x\|,$$

respectively, where $\|\cdot\|$ denotes some vector norm (see Sec. 1.6.8). Then a bound $\|\tilde{x} - x\|/\|x\| \leq 1/2 \cdot 10^{-p}$ implies that components $\tilde{x}_i$ with $|\tilde{x}_i| \approx \|x\|$ have about p significant digits but this is not true for components of smaller absolute value. An

alternative is to use componentwise relative errors, e.g.,

$$\max_i |\tilde{x}_i - x_i|/|x_i|, \tag{2.1.1}$$

but this assumes that $x_i \neq 0$, for all i.

We will distinguish between the terms accuracy and precision. By **accuracy** we mean the absolute or relative error of an approximate quantity. The term **precision** will be reserved for the accuracy with which the basic arithmetic operations $+, -, *, /$ are performed. For floating point operations this is given by the unit roundoff; see (2.2.8).

Numerical results which are not followed by any error estimations should often, though not always, be considered as having an uncertainty of $\frac{1}{2}$ a unit in the last decimal place. In presenting numerical results, it is a good habit, if one does not want to go to the difficulty of presenting an error estimate with each result, to give explanatory remarks such as:

- "All the digits given are thought to be significant."
- "The data has an uncertainty of at most 3 units in the last digit."
- "For an ideal two-atomed gas, $c_P/c_V = 1.4$ (exactly)."

We shall also introduce some notations, useful in practice, though their definitions are not exact in a mathematical sense:

> $a \ll b$ ($a \gg b$) is read: "a is much smaller (much greater) than b". What is meant by "much smaller"(or "much greater") depends on the context—among other things, on the desired precision.
>
> $a \approx b$ is read: "a is approximately equal to b" and means the same as $|a - b| \ll c$, where c is chosen appropriate to the context. We *cannot generally* say, for example, that $10^{-6} \approx 0$.
>
> $a \lessapprox b$ (or $b \gtrapprox a$) is read: "a is less than or approximately equal to b" and means the same as "$a \leq b$ or $a \approx b$."

Occasionally we shall have use for the following more precisely defined mathematical concepts:

> $f(x) = O(g(x))$, $x \to a$, means that $|f(x)/g(x)|$ is bounded as $x \to a$ (a can be finite, $+\infty$, or $-\infty$).
>
> $f(x) = o(g(x))$, $x \to a$, means that $\lim_{x\to a} f(x)/g(x) = 0$.
>
> $f(x) \sim g(x)$, $x \to a$, means that $\lim_{x\to a} f(x)/g(x) = 1$.

2.1.3 Rounding and Chopping

When one counts the *number of digits* in a numerical value one should not include zeros in the beginning of the number, as these zeros only help to denote where the decimal point should be. If one is counting the *number of decimals*, one should of

course include leading zeros to the right of the decimal point. For example, the number 0.00147 is given with three digits but has five decimals. The number 12.34 is given with four digits but has two decimals.

If the magnitude of the error in $\tilde{a}$ does not exceed $\frac{1}{2} \cdot 10^{-t}$, then $\tilde{a}$ is said to have t **correct decimals**. The *digits* in $\tilde{a}$ which occupy positions where the unit is greater than or equal to 10^{-t} are called, then, **significant digits** (any initial zeros are not counted). Thus, the number 0.001234 ± 0.000004 has five correct decimals and three significant digits, while 0.001234 ± 0.000006 has four correct decimals and two significant digits. The number of correct decimals gives one an idea of the magnitude of the *absolute error*, while the number of significant digits gives a rough idea of the magnitude of the *relative error*.

We distinguish here between two ways of rounding off a number x to a given number t of decimals. In **chopping** (or round toward zero) one simply leaves off all the decimals to the right of the tth. That way is generally *not recommended* since the rounding error has, systematically, the opposite sign of the number itself. Also, the magnitude of the error can be as large as 10^{-t}.

In **rounding to nearest** (sometimes called "correct" or "optimal" rounding"), one chooses, a number with s decimals which is *nearest* to x. Hence if p is the part of the number which stands to the right of the sth decimal one leaves the tth decimal unchanged if and only if $|p| < 0.5 \cdot 10^{-s}$. Otherwise one raises the sth decimal by 1. In case of a tie, when x is equidistant to two s digit numbers then one raises the sth decimal if it is odd or leaves it unchanged if it is even (round to even). In this way, the error is positive or negative about equally often. The error in rounding a decimal number to s decimals will always lie in the interval $\left[-\frac{1}{2}10^{-s}, \frac{1}{2}10^{-s}\right]$.

Suppose that you are tabulating a transcendental function and a particular entry has been evaluated as 1.2845 correct to the digits given. You want to round the value to three decimals. Should the final digit be 4 or 5? The answer depends on whether there is a nonzero trailing digit. You compute the entry more accurately and find 1.28450, then 1.284500, then 1.2845000, etc. Since the function is transcendental, there clearly is no bound on the number of digits that has to be computed before distinguishing if to round to 1.284 or 1.285. This is called the **tablemaker's dilemma**.[2]

Example 2.1.1.

Shortening to three decimals:

0.2397	rounds to 0.240	(is chopped to 0.239)
−0.2397	rounds to −0.240	(is chopped to −0.239)
0.23750	rounds to 0.238	(is chopped to 0.237)
0.23650	rounds to 0.236	(is chopped to 0.236)
0.23652	rounds to 0.237	(is chopped to 0.236)

[2]This can be used to advantage in order to protect mathematical tables from illegal copying by rounding a few entries incorrectly where the error in doing so is insignificant due to several trailing zeros. An illegal copy could then be exposed simply by looking up these entries!

Observe that when one rounds off a numerical value one produces an error; thus it is occasionally wise to give more decimals than those which are correct. Take, for example, $a = 0.1237 \pm 0.0004$, which has three correct decimals according to the definition given previously. If one rounds to three decimals, one gets 0.124; here the third decimal is not correct, since the least possible value for a is 0.1233.

Example 2.1.2.

The difference between chopping and rounding can be important as is illustrated by the following story. The index of the Vancouver Stock Exchange, founded at the initial value 1000.000 in 1982, was hitting lows in the 500s at the end of 1983 even though the exchange apparently performed well. It was discovered (The Wall Street Journal, Nov. 8, 1983, p. 37) that the discrepancy was caused by a computer program which updated the index thousands of times a day and used chopping instead of rounding to nearest! The rounded calculation gave a value of 1098.892.

Review Questions

1. Clarify with examples the various types of error sources which occur in numerical work.
2. (a) Define "absolute error" and "relative error" for an approximation $\bar{x}$ to a scalar quantity x. What is meant by an error bound?
   (b) Generalize the definitions in (a) to a vector x.
3. (a) How is "rounding to nearest" performed.
4. Give π to four decimals using: (a) chopping; (b) rounding.
5. What is meant by the "tablemaker's dilemma"?

2.2 Computer Number Systems

2.2.1 The Position System

In order to represent numbers, we use in daily life a **position system** with base 10 (the decimal system). Thus to represent the numbers we use ten different characters, and the magnitude with which the digit a contributes to the value of a number depends on the digit's position in the number. If the digit stands n steps to the right of the decimal point, the value contributed is $a \cdot 10^{-n}$. For example, the sequence of digits 4711.303 means

$$4 \cdot 10^3 + 7 \cdot 10^2 + 1 \cdot 10^1 + 1 \cdot 10^0 + 3 \cdot 10^{-1} + 0 \cdot 10^{-2} + 3 \cdot 10^{-3}.$$

Every real number has a unique representation in the above way, except for the possibility of infinite sequences of nines—for example, the infinite decimal fraction $0.3199999\ldots$ represents the same number as 0.32.

One can very well consider other position systems with base different from 10. Any natural number $\beta \geq 2$ (or $\beta \leq -2$) can be used as base. One can show that every positive real number a has, with exceptions analogous to the nines-sequences mentioned above, a unique representation of the form

$$a = d_n\beta^n + d_{n-1}\beta^{n-1} + \ldots + d_1\beta^1 + d_0\beta^0 + d_{-1}\beta^{-1} + d_{-2}\beta^{-2} + \ldots,$$

or more compactly $a = (d_n d_{n-1} \ldots d_0.d_{-1}d_{-2}\ldots)_\beta$, where the coefficients d_i, the "digits" in the system with base β, are positive integers d_i such that $0 \leq d_i \leq \beta - 1$.

One of the greatest advantages of the position system is that one can give simple, general rules for the arithmetic operations. The smaller the base is, the simpler these rules become. This is just one reason why most computers operate in base 2, the **binary number system**. The addition and multiplication tables then take the following simple form:

$$\begin{array}{lll} 0+0=0; & 0+1=1+0=1; & 1+1=10; \\ 0\cdot 0=0; & 0\cdot 1=1\cdot 0=0; & 1\cdot 1=1; \end{array}$$

In the binary system, the number seventeen, for example, becomes 10001, since $1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = \text{sixteen} + \text{one} = \text{seventeen}$. Put another way $(10001)_2 = (17)_{10}$, where the index (in decimal representation) denotes the base of the number system. The numbers become longer written in the binary system; large integers become about 3.3 times as long, since N binary digits suffice to represent integers less than $2^N = 10^{N\log_{10}2} \approx 10^{N/3.3}$.

Occasionally one groups together the binary digits in subsequences of three or four, which is equivalent to using 2^3 and 2^4, respectively, as base. These systems are called the **octal** and **hexadecimal** number systems, respectively. The octal system uses the digits from 0 to 7; in the hexadecimal system the digits 0 through 9 and the letters A, B, C, D, E, F ("ten" through "fifteen") are used.

Example 2.2.1.

$$\begin{aligned} (17)_{10} &= (10001)_2 = (21)_8 = (11)_{16}, \\ (13.25)_{10} &= (1101.01)_2 = (15.2)_8 = (D.4)_{16}, \\ (0.1)_{10} &= (0.000110011001\ldots)_2 = (0.199999\ldots)_{16}. \end{aligned}$$

Note that *the finite decimal fraction* 0.1 *cannot be represented exactly by a finite fraction in the binary number system*! (For this reason some pocket calculators use the base 10.)

Example 2.2.2.

In 1991 a Patriot missile in Saudi Arabia failed to track and interrupt an incoming Scud due to a precision problem. The Scud then hit an Army barrack and killed 28 Americans. The computer used to control the Patriot missile was based on a design dating from the 1970's using 24-bit arithmetic. For the tracking computations time was recorded by the system clock in tenth of a second but converted to

a 24-bit floating point number. Rounding errors in the time conversions caused an error in the tracking. After 100 hours of consecutive operations the calculated time in seconds was 359999.6567 instead of the correct value 360000, an error of 0.3433 seconds leading to an error in the calculated range of 687 meters; see Skeel [41]. Modified software was later installed.

In the binary system the "point" used to separate the integer and fractional part of a number (corresponding to the decimal point) is called the binary point. The digits in the binary system are called **bits**(=**b**inary dig**its**).

We are so accustomed to the position system that we forget that it is built upon an ingenious idea. The reader can puzzle over how the rules for arithmetic operations would look if one used Roman numerals, a number system without the position principle described above.

Recall that rational numbers are precisely those real numbers which can be expressed as a quotient between two integers. Equivalently rational numbers are those whose representation in a position system have a finite number of digits or whose digits are repeating.

We now consider the problem of conversion between two number systems with different base. Since almost all computers use a binary system this problem arises as soon as one want to input data in decimal form or print results in decimal form.

Algorithm 2.2.1 Conversion between number systems:

Let a be an integer given in number systems with base α. We want to determine its representation in a number system with base β:

$$a = b_n\beta^m + b_{m-1}\beta^{n-1} + \cdots + b_0, \quad 0 \le b_i < \beta. \tag{2.2.1}$$

The computations are to be done in the system with base α and thus also β is expressed in this representation. The conversion is done by successive divisions of a with β: Set $q_0 = a$, and

$$q_k/\beta = q_{k+1} + b_k/\beta, \quad k = 0, 1, 2, \ldots \tag{2.2.2}$$

(q_{k+1} is the quotient and b_k the remainder in the division).

If a is not an integer, we write $a = b + c$, where b is the integer part and

$$c = c_{-1}\beta^{-1} + c_{-2}\beta^{-2} + c_{-3}\beta^{-3} + \cdots \tag{2.2.3}$$

is the fractional part, where $c_{-1}, c_{-2}, \ldots$ are to be determined. These digits are obtained as the integer parts when successively multiplying c with β: Set $p_{-1} = c$, and

$$p_k \cdot \beta = c_k\beta + p_{k-1}, \quad k = -1, -2, -3 \ldots. \tag{2.2.4}$$

Since a finite fraction in a number system with base α usually does not correspond to a finite fraction in the number system with base β rounding of the result is in general needed.

When converting by hand between the decimal system and, for example, the binary system all computations are made in the decimal system ($\alpha = 10$ and $\beta = 2$). (It is then more convenient to convert the decimal number first to octal or hexadecimal, from which the binary representation easily follows.) If, on the other hand, the conversion is carried out on a binary computer, the computations are made in the binary system ($\alpha = 2$ and $\beta = 10$).

Example 2.2.3.

Convert the decimal number 176.524 to ternary form (base $\beta = 3$). For the integer part we get $176/3 = 58$ with remainder 2; $58/3 = 19$ with remainder 1; $19/3 = 6$ with remainder 1; $6/3 = 2$ with remainder 0; $2/3 = 0$ with remainder 2. It follows that $(176)_{10} = (20112)_3$.

For the fractional part we compute $.524 \cdot 3 = 1.572$, $.572 \cdot 3 = 1.716$, $.716 \cdot 3 = 2.148, \ldots$. Continuing in this way we obtain $(.524)_{10} = (.112010222\ldots)_3$. The finite decimal fraction does not correspond to a finite fraction in the ternary number system!

2.2.2 Fixed and Floating Point Representation

A computer is in general built to handle pieces of information of a fixed size called a **word**. The number of digits in a word (usually binary) is called the **word-length** of the computer. Typical word-lengths are 32, 48, or 64 bits. A real or integer number is usually stored in a word. Integers can be exactly represented, provided that the word-length suffices to store all the digits in its representation.

In the first generation of computers calculations were made in a **fixed-point** number system, that is, real numbers were represented with a fixed number of t binary digits. If the word-length of the computer is $s + 1$ bits (including the sign bit), then only numbers in the interval $I = [-2^{s-t}, 2^{s-t}]$ are permitted. Some common conventions in fixed point are $t = s$ (fraction convention) or $t = 0$ (integer convention). This limitation causes difficulties, since even when $x \in I$, $y \in I$, we can have, e.g., $x - y \notin I$ or $x/y \notin I$. In a fixed-point number system one must see to it that all numbers, even intermediate results, remain within I. This can be attained by multiplying the variables by appropriate **scale factors**, and then transforming the equations accordingly. This is a tedious process. Moreover it is complicated by the risk that if the scale factors are chosen carelessly, certain intermediate results can have many leading zeros which can lead to poor accuracy in the final results. As a consequence, fixed point is very seldom used for computations with real numbers. An exception is in some on-line real-time computations, e.g., in digital filtering, where fixed point systems still are used. Otherwise it is limited to computations with integers as in subscript expressions for vectors and matrices.

By a **normalized floating point representation** of a real number a, we mean a representation in the form

$$a = \pm m \cdot \beta^e, \qquad \beta^{-1} \le m < 1, \qquad e \text{ an integer.} \tag{2.2.5}$$

Such a representation is possible for all real numbers a, and unique if $a \neq 0$. (The number 0 is treated as a special case.) Here the fraction part m is called the

mantissa[3] or **significand**), e is the **exponent** and β the **base** (also called the **radix**).

In a computer, the number of digits for q and m is limited by the word-length. Suppose that t digits is used to represent m. Then we can only represent floating point numbers of the form

$$\bar{a} = \pm\overline{m} \cdot \beta^e, \quad \overline{m} = (0.d_1 d_2 \cdots d_t)_\beta, \quad 0 \le d_i < \beta, \tag{2.2.6}$$

where $\overline{m}$ is the mantissa m rounded to t digits, and the exponent is limited to a finite range

$$e_{\min} \le e \le e_{\max}. \tag{2.2.7}$$

A floating point number system F is characterized by the base β, the precision t, and the numbers $e_{\min}$, $e_{\max}$. Only a finite set F of rational numbers can be represented in the form (2.2.6). The numbers in this set are called **floating point numbers**. Since $d_1 \neq 0$ this set contains, including the number 0, precisely

$$2(\beta - 1)\beta^{t-1}(e_{\max} - e_{\min} + 1) + 1$$

numbers. (Show this!) The limited number of digits in the exponent implies that a is limited in magnitude to an interval which is called the **range** of the floating point system. If a is larger in magnitude than the largest number in the set F, then a cannot be represented at all (**exponent spill**). The same is true, in a sense, of numbers smaller than the smallest nonzero number in F.

Example 2.2.4.

Consider the floating point number system for $\beta = 2$, $t = 3$, $e_{\min} = -1$, and $e_{\max} = 2$. The positive normalized numbers in the corresponding set F are shown in Fig. 2.2.1. The set F contains exactly $2 \cdot 16 + 1 = 33$ numbers. In this

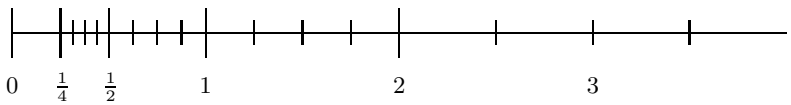


Figure 2.2.1. *Positive normalized numbers when $\beta = 2$, $t = 3$, $e_{\min} = -1$, and $e_{\max} = 2$.*

example the nonzero numbers of smallest magnitude that can be represented are $(0.100)_2 \cdot 2^{-1} = \frac{1}{4}$ and the largest is $(0.111)_2 \cdot 2^2 = \frac{7}{2}$.

Notice that floating point numbers are not equally spaced; the spacing jumps by a factor β at each power of β. This wobbling is smallest for $\beta = 2$.

Definition 2.2.1.

[3] Strictly speaking mantissa refers to the decimal part of a logarithm.

The spacing of floating point numbers is characterized by the **machine epsilon**, *which is the distance ϵ_M from 1.0 to the next larger floating point number.*

The leading significant digit of numbers represented in a number system with base β has been observed to closely fit a logarithmic distribution, i.e., the proportion of numbers with leading digit equal to n is $\ln_\beta(1+1/n)$ $(n = 0, 1, \ldots, \beta-1)$. It has been shown that under this assumption taking the base equal to 2 will minimize the mean square representation error. A discussion of this intriguing fact with historic references is found in Higham [29, Sec. 2.7].

Even if the operands in an arithmetic operation are floating point numbers in F, the *exact* result of the operation may not be in F. For example, the exact product of two floating point t-digit numbers has $2t$ or $2t-1$ digits.

If a real number a is in the range of the floating point system the obvious way is to represent a by $\bar{a} = fl\,(a)$, where $fl\,(a)$ denotes a number in F which is nearest to a. This corresponds to rounding of the mantissa m, and according to Sec. 2.1.3, we have

$$|\overline{m} - m| \leq \frac{1}{2}\beta^{-t}.$$

(There is one exception. If $|m|$ after rounding should be raised to 1, then $|\overline{m}|$ is set equal to 0.1 and e raised by 1.) Since $m \geq 0.1$ this means that the magnitude of the relative error in $\bar{a}$ is at most equal to

$$\frac{\frac{1}{2}\beta^{-t}\cdot\beta^e}{m\cdot\beta^e} \leq \frac{1}{2}\beta^{-t+1}.$$

Even with the exception mentioned above this relative bound still holds. (If chopping is used, this doubles the error bound above.) This proves the following theorem:

Theorem 2.2.2.

In a floating point number system $F = F(\beta, t, e_{\min}, e_{\max})$ every real number in the floating point range of F can be represented with a relative error, which does not exceed the **unit roundoff** *u, which is defined by*

$$u = \begin{cases} \frac{1}{2}\beta^{-t+1}, & \textit{if rounding is used,} \\ \beta^{-t+1}, & \textit{if chopping is used.} \end{cases} \tag{2.2.8}$$

Note that in a floating point system both large and small numbers are represented with nearly *the same relative precision.* The quantity u is, in many contexts, a natural unit for relative changes and relative errors. For example, termination criteria in iterative methods usually depend on the unit roundoff.

To measure the difference between a floating point number and the real number it approximates we shall occasionally use "**unit in last place**" or **ulp**. For example, if in a decimal floating point system the number 3.14159 is represented as $0.3142\cdot10^1$ this has an error of 0.41 ulps. We shall say that "the quantity is perturbed by a few ulps".

Example 2.2.5.

Sometimes it is useful to be able to approximately determine the unit roundoff in a program at run time. This may be done using the observation that $u \approx \mu$, where μ is *the smallest floating point number x such that $fl(1+x) > 1$*. The following program computes a number μ which differs from the unit roundoff u at most by a factor of 2:

$$
\begin{aligned}
&x := 1;\\
&\textbf{while } 1 + x > 1 \quad x := x/2; \textbf{ end};\\
&\mu := x;
\end{aligned}
$$

One reason why u does not exactly equal μ is that so called double rounding occurs. This is when a result is first rounded to extended format and then to the target precision.

A floating point number system can be extended by including **denormalized numbers** (also called subnormal numbers). These are numbers with the minimum exponent and with the most significant digit equal to zero. The three numbers

$$(.001)_2 2^{-1} = 1/16, \quad (.010)_2 2^{-1} = 2/16, \quad (.011)_2 2^{-1} = 3/16,$$

can then also be represented. Denormalized numbers have fewer digits of precision than normalized numbers.

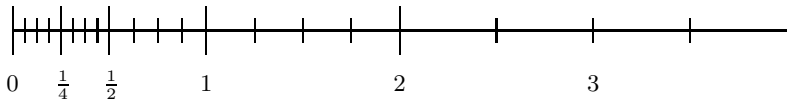


Figure 2.2.2. *Positive normalized and denormalized numbers when $\beta = 2$, $t = 3$, $e_{\min} = -1$, and $e_{\max} = 2$.*

2.2.3 IEEE Floating Point Standard

Actual computer implementations of floating point representations may differ in detail from the one given above. Although some pocket calculators use a floating point number system with base $\beta = 10$, almost all modern computers use base $\beta = 2$. Most current computers now conform to the IEEE 754 standard for binary floating point arithmetic.[4] This standard from 1985 (see [21]), which is the result of several years work by a subcommittee of the IEEE, is now implemented on almost all chips used for personal computers and workstations. (There is also a standard IEEE 854 for floating point arithmetic for base 2 and 10, which is used by several hand calculators.)

The IEEE 754 standard specifies basic and extended formats for floating point numbers, elementary operations and rounding rules available, conversion between

[4] W. Kahan, University of California, Berkeley, was given the Turing Award by the Association of Computing Machinery for his contribution to this standard.

different number formats, and binary-decimal conversion. The handling of exceptional cases like exponent overflow or underflow and division by zero are also specified.

Two main basic formats, single and double precision are defined, using 32 and 64 bits respectively. In **single precision** a floating point number a is stored as a sign s (one bit), the exponent e (8 bits), and the mantissa m (23 bits). In **double precision** of the 64 bits 11 are used for the exponent, and 52 bits for the mantissa; see Fig. 2.2.2. The value v of a is in the normal case

$$v = (-1)^s (1.m)_2 2^e, \quad -e_{\min} \le e \le e_{\max}. \tag{2.2.9}$$

Note that the digit before the binary point is always 1 for a normalized number. Thus the normalization of the mantissa is different from that in (2.2.6). This bit is not stored (the hidden bit). In that way one bit is gained for the mantissa. A biased exponent is stored and no sign bit used for the exponent. For example, in single precision $e_{\min} = -126$ and $e_{\max} = 127$ and $e + 127$ is stored.

The unit roundoff equals

$$u = \begin{cases} 2^{-24} \approx 5.96 \cdot 10^{-8}, & \text{in single precision;} \\ 2^{-53} \approx 1.11 \cdot 10^{-16} & \text{in double precision.} \end{cases}$$

(The machine epsilon is twice as large.) The largest number that can be represented is approximately $2.0 \cdot 2^{127} \approx 3.4028 \times 10^{38}$ in single precision and $2.0 \cdot 2^{1023} \approx 1.7977 \times 10^{308}$ in double precision. The smallest normalized number is $1.0 \cdot 2^{-126} \approx 1.1755 \times 10^{-38}$ in single precision and $1.0 \cdot 2^{-1022} \approx 2.2251 \times 10^{-308}$ in double precision.

An exponent $e = e_{\min} - 1$ and $m \neq 0$, signifies the denormalized number

$$v = (-1)^s (0.m)_2 2^{e_{\min}};$$

The smallest denormalized number that can be represented is $2^{-126-23} \approx 7.14 \cdot 10^{-44}$ in single precision and $2^{-1022-52} \approx 4.94 \cdot 10^{-324}$ in double precision.

There are distinct representations for $+0$ and -0. ± 0 is represented by a sign bit, the exponent $e_{\min} - 1$ and a zero mantissa. Comparisons are defined so that $+0 = -0$. One use of a signed zero is to distinguish between positive and negative underflowed numbers. Another use occurs in complex arithmetic.

Example 2.2.6.

The function $\sqrt{x}$ is multivalued and there is no way to select the values so the function is continuous over the whole complex plane. If a branch cut is made by excluding all real negative numbers from consideration the square root becomes continuous. Signed zero provides a way to distinguish numbers of the form $x+i(+0)$ and $x + i(-0)$ and to select one or the other side of the cut.

Infinity is also signed and $\pm\infty$ is represented by the exponent $e_{\max} + 1$ and a zero mantissa. When overflow occurs the result is set to $\pm\infty$. This is safer than simply returning the largest representable number, that may be nowhere near the

correct answer. The result $\pm\infty$ is also obtained from the illegal operations $a/0$, where $a \neq 0$. The infinity symbol obeys the usual mathematical conventions, such as $\infty + \infty = \infty$, $(-1) \times \infty = -\infty$, $a/\infty = 0$ if $a \neq 0$.

The IEEE standard also includes two extended precision formats that offer extra precision and exponent range. The standard only specifies a lower bound on how many extra bits it provides.[5] Extended formats simplify tasks such as computing elementary functions accurately in single or double precision. Extended precision formats are used also by hand calculators. These will often display 10 decimal digits but use 13 digits internally—"the calculator knows more than it shows"!

The characteristics of the IEEE formats are summarized in Table 2.2.1. (The hidden bit in the mantissa accounts for the +1 in the table. Note that double precision satisfies the requirements for single extended, so three different precisions suffice.)

Table 2.2.1. *IEEE floating point formats.*

	Format	t	e	$e_{\min}$	$e_{\max}$
Single	32 bits	$23+1$	8 bits	-126	127
Single extended	≥ 43 bits	≥ 32	≥ 11 bits	≤ -1022	≥ 1023
Double	64 bits	$52+1$	11 bits	-1022	1023
Double extended	≥ 79 bits	≥ 64	≥ 15 bits	≤ -16382	≥ 16383

Example 2.2.7.

Although the exponent range of the floating point formats seems reassuringly large, even simple programs can quickly give exponent spill. If $x_0 = 2$, $x_{n+1} = x_n^2$, then already $x_{10} = 2^{1024}$ is larger than what IEEE double precision permits. One should also be careful in computations with factorials, e.g., $35! > 10^{40}$ and $459! > 10^{1026}$.

Four rounding modes are supported by the standard. The default rounding mode is round to nearest representable number, with round to even in case of a tie. (Some computers in case of a tie round away from zero, i.e., raise the absolute value of the number, because this is easier to realize technically.) Chopping is also supported as well as directed rounding to ∞ and to $-\infty$. The latter mode simplifies the implementation of interval arithmetic, see Sec. 2.5.3.

The standard specifies that all arithmetic operations should be performed as if they were first calculated to infinite precision and then rounded to a floating point number according to one of the four modes mentioned above. This also includes the square root and conversion between integer and floating point. The standard also requires the conversion between internal formats and decimal to be correctly rounded.

[5] Hardware implementation of extended precision normally does not use a hidden bit, so the double extended format uses 80 bits rather than 79.

This can be implemented using extra **guard digits** in the intermediate result of the operation before normalization and rounding. Using a single guard digit, however, will not always ensure the desired result. However by introducing a second guard digit and a third sticky bit (the logical OR of all succeeding bits) the rounded exact result can be computed at only a little more cost (Goldberg [25]). One reason for specifying precisely the results of arithmetic operations is to improve the portability of software. If a program is moved between two computers both supporting the IEEE standard intermediate results should be the same.

IEEE arithmetic is a closed system, i.e. every operation, even mathematical invalid operations, even $0/0$ or $\sqrt{-1}$ produces a result. To handle exceptional situations without aborting the computations some bit patterns (see Table 2.2.2) are reserved for special quantities like NaN ("Not a Number") and ∞. NaNs (there are more than one NaN) are represented by $e = e_{\max} + 1$ and $m \neq 0$.

Table 2.2.2. IEEE 754 *representation.*

Exponent	Mantissa	Represents
$e = e_{\min} - 1$	$m = 0$	± 0
$e = e_{\min} - 1$	$m \neq 0$	$\pm 0.m \cdot 2^{e_{\min}}$
$e_{\min} < e < e_{\max}$		$\pm 1.m \cdot 2^{e}$
$e = e_{\max} + 1$	$m = 0$	$\pm\infty$
$e = e_{\max} + 1$	$m \neq 0$	NaN

Note that the gap between 0 and the smallest normalized number is $1.0 \times 2^{e_{\min}}$. This is much larger than for the spacing $2^{-p+1} \times 2^{e_{\min}}$ for the normalized numbers for numbers just larger than the underflow threshold; compare Example 2.2.4. With denormalized numbers the spacing becomes more regular and permits what is called **gradual underflow**.. This makes many algorithms well behaved also close to the underflow threshold. Another advantage of having gradual underflow is that it makes it possible to preserve the property

$$x = y \quad \Leftrightarrow \quad x - y = 0$$

as well as other useful relations. Several examples of how denormalized numbers makes writing reliable floating point code easier are analyzed by Demmel [17].

One illustration of the use of extended precision is in converting between IEEE 754 single precision and decimal. The converted single precision number should ideally be converted with enough digits so that when it is converted back the binary single precision number is recovered. It might be expected that since $2^{24} < 10^8$ eight decimal digits in the converted number would suffice. However, it can be shown that nine decimal digits are needed to recover the binary number uniquely (see Goldberg [25, Theorem. 15] and Problem 3). When converting back to binary form a rounding error as small as one ulp will give the wrong answer. To do this conversion efficiently extended single precision is needed![6]

[6]It should be noted that some computer languages do not include input/output routines, but

A NaN is generated by operations such as $0/0$, $+\infty+(-\infty)$, $0\times\infty$ and $\sqrt{-1}$. A NaN compares unequal with everything including itself. (Note that $x \neq x$ is a simple way to test if x equals a NaN.) When a NaN and an ordinary floating-point number is combined the result is the same as the NaN operand. A NaN is often used also for uninitialized or missing data.

Exceptional operations also raise a flag. The default is to set a flag and continue, but it is also possible to pass control to a trap handler. The flags are "sticky" in that they remain set until explicitly cleared. This implies that without a log file everything before the last setting is lost, why it is always wise to use a trap handler. There is one flag for each of the following five exceptions: underflow, overflow, division by zero, invalid operation and inexact. By testing the flags it is, for example, possible to test if an overflow is genuine or the result of division by zero.

Because of cheaper hardware and increasing problem sizes double precision is more and more used in scientific computing. With increasing speed and memory becoming available, bigger and bigger problems are being solved and actual problems may soon require more than IEEE double precision! When the IEEE 754 standard was defined no one expected computers able to execute more than 10^{12} floating point operations per second!

2.2.4 Elementary Functions

Although the square root is included, the IEEE 754 standard does not deal with the implementation of elementary functions, i.e., the exponential function exp, the logarithm ln, and the trigonometric and hyperbolic functions sin, cos, tan, sinh, cosh, tanh, and their inverse functions. With the IEEE 754 standard more accurate implementations are possible which in many cases give almost correctly rounded exact results. To always guarantee correctly rounded exact results sometimes require computing many more digits than the target accuracy (cf. the tablemaker's dilemma) and therefore is in general too costly. It is also important to preserve monotonicity, e.g, $0 \le x \le y \le \pi/2 \Rightarrow \sin x \le \sin y$, and range restrictions, e.g., $\sin x \le 1$, but these demands may conflict with rounded exact results!

The first step in computing an elementary function is to perform a **range reduction**.

- To compute trigonometric functions, e.g., $\sin x$, an additive range reduction is first performed, in which a reduced argument x^*, $-\pi/4 \le x^* \le \pi/4$, is computed by finding an integer k such that

$$x^* = x - k\pi/2, \quad (\pi/2 = 1.57079\,63267\,94896\,61923).$$

  (Quantities such as $\pi/2$, $\ln(2)$, that are often used in standard subroutines are listed in decimal form to 30 digits and octal form to 40 digits in Hart et al. [Appendix C][28] and to 40 and 44 digits in Knuth [32, Appendix A].)

these are developed separately. This can lead to double rounding, which spoils the careful designed accuracy in the IEEE 754 standard. (Some banks use separate routines with chopping even today—you may guess why!)

Then $\sin x = \pm \sin x^*$ or $\sin x = \pm \cos x^*$, depending on if k mod 4 equals $0, 1, 2$ or 3. Hence approximation for $\sin x$ and $\cos x$ need only be provided for $0 \le x \le \pi/4$. If the argument x is very large then cancellation in the range reduction can lead to poor accuracy; see Example 2.3.9.

- To compute the exponential function $\exp(x)$ an integer k is determined such that

$$x^* = x - k \ln 2, \quad x^* \in [0, \ln 2] \quad (\ln 2 = 0.69314\,71805\,59945\,30942\ldots). \tag{2.2.10}$$

  It then holds that $\exp(x) = \exp(x^*) \cdot 2^k$ and hence we only need an approximation of $\exp(x)$ for the range $x \in [0, \ln 2]$; see Problem 13.

- To compute $\ln x$, $x > 0$, a multiplicative range reduction is used. If an integer k is determined such that

$$x^* = x/2^k, \quad x^* \in [1/2, 1],$$

  then $\ln x = \ln x^* + k \cdot \ln 2$.

We remark that rational approximations often give much better accuracy than polynomial approximations. This as related to the fact that continued fraction expansions often converge much faster than those based on power series. We refer to Sec. 3.3 for a discussion of continued fraction and related Padé approximations.

Coefficients of polynomial and rational approximations suitable for software implementations are tabulated in Hart et al. [28] and Cody and Waite [16]. However, approximation of functions can now be simply obtained using software such as Maple [13]. For example in Maple the commands

```
Digits = 40; minimax(exp(x), x = 0..1, [i,k],1,'err')
```

means that we are looking for the coefficients of the minimax approximation of the exponential function on $[0, 1]$ by a rational function with numerator of degree i and denominator of degree k with weight function 1 and that the variable `err` should be equal to the approximation error. The coefficients are to be computed to 40 decimal digits. A trend now is that elementary functions are more and more implemented in hardware. Hardware implementations are discussed by Muller [35]. Carefully implemented algorithms for elementary functions are available from `www.netlib.org/fdlibm` in the library package fdlibm (Freely Distributable Math. Library) developed by Sun Microsystems and used by MATLAB .

To test the implementation of elementary functions a Fortran package ELEFUNT has been developed by Cody [14]. This checks the quality using indentities like $\cos x = \cos(x/3)(4\cos^2(x/3) - 1)$. For complex elementary functions a package CELEFUNT serves the same purpose; see Cody [15].

2.2.5 Multiple Precision Arithmetic

Hardly any quantity in the physical world is known to an accuracy beyond IEEE double precision. A value of π correct to 20 decimal digits would suffice to cal-

culate the circumference of a circle around the sun at the orbit of the earth to within the width of an atom. There seems to be little need for multiple precision calculations. Occasionally, however, one may want to perform some calculations, e.g., the evaluation of some mathematical constant (such as π and Euler's constant γ) or elementary functions, to very high precision.[7] Extremely high precision is sometimes needed in experimental mathematics, e.g., when trying to discover new mathematical identities. Algorithms, which may be used for these purposes include power series, continued fractions, solution of equations with Newton's method or other superlinearly convergent methods, etc.).

For performing such tasks it is convenient to use arrays to represent numbers in a floating point form with a large base and a long mantissa and have routines for performing floating point operations on such numbers. In this way it is possible to *simulate arithmetic of arbitrarily high precision* using standard floating point arithmetic.

Brent [10, 9] developed the first major such multiple-precision package in Fortran 66. His package represents multiple precision numbers as arrays of integers and operates on them with integer arithmetic. It includes subroutines for multiple precision evaluation of elementary functions. A more recent package called MPFUN, written in Fortran 77 code, is that of Bailey [3]. In MPFUN a multiple precision numbers is represented as a vector of single precision floating point numbers with base 2^{24}. Complex multiprecision numbers are also supported. There is also a Fortran 90 version of this package [4], which is easy to use.

Fortran routines for high-precision computation are also provided in Press et al [37, §20.6], and is also supported by symbolic manipulation systems such as Maple [13] and Mathematica [48].

In Appendix A we describe the basics of **Mulprec**, a collection of MATLAB m-files for, in principle, unlimited multiple precision floating point computations and give some examples of its use.

Review Questions

1. What base β is used in the binary, octal and hexadecimal number systems?
2. Show that any *finite* decimal fraction corresponds to a binary fraction that eventually is periodic.
3. (a) What is meant by a normalized floating point representation of a real number?
4. (a) How large can the maximum relative error be in representation of a real number a in the floating point system $F = F(\beta, p, e_{\min}, e_{\max})$? It is assumed that a is in the range of F.

   (b) How are the quantities "machine epsilon" and "unit round off" defined?
5. What are the characteristics of the IEEE single and double precision formats?

[7] In Oct. 1995 Yasumasa Kanada of the University of Tokyo computed π to 6,442,458,938 decimals on a Hitachi supercomputer; see [5].

6. What are the advantages of including denormalized numbers in the IEEE standard?

7. Give examples of operations that give NaN as result.

Problems and Computer Exercises

1. Which rational numbers can be expressed with a finite number of binary digits to the right of the binary point?

2. (a) Prove the algorithm for conversion between number systems given in Sec. 2.2.1.

(b) Give the hexadecimal form of the decimal numbers 0.1 and 0.3. What error is incurred in rounding these numbers to IEEE 754 single and double precision? (c) What is the result of the computation 0.3/0.1 in IEEE 754 single and double precision ?

3. (W. Kahan) An (over-)estimate of u can be obtained for almost any computer by evaluating $|3\times(4/3-1)-1|$ using rounded floating point for every operation. Test this on a calculator or computer available to you.

4. (Goldberg [25]) The binary single precision numbers in the half-open interval $[10^3, 1024)$ have 10 bits to the left and 14 bits to the right of the binary point. Show that there are $(2^{10}-10^3)\cdot 2^{14} = 393,216$ such numbers, but only $(2^{10}-10^3)\cdot 10^4 = 240,000$ decimal numbers with 8 decimal digits in the same interval. Conclude that 8 decimal digits are not enough to uniquely represent single precision binary numbers in the IEEE 754 standard.

5. Suppose one wants to compute the power A^n of a square matrix A, where n is a positive integer. To compute $A^{k+1} = A\cdot A^k$, for $k = 1 : n-1$ requires $n-1$ matrix multiplications. Show that the number of multiplications can be reduced to less than $2\lfloor \log_2 n \rfloor$ by converting n into binary form and successive squaring $A^{2k} = (A^k)^2$, $k = 1 : \lfloor \log_2 n \rfloor$.

6. Give in decimal representation: (a) $(10000)_2$; (b) $(100)_8$; (c) $(64)_{16}$; (d) $(FF)_{16}$; (e) $(0.11)_8$; (g) the largest positive integer which can be written with thirty–one binary digits (answer with one significant digit).

7. (a) Show how the following numbers are stored in the basic single precision format of the IEEE 754 standard: 1.0; −0.0625; 250.25; 0.1.

(b) Give in decimal notation the largest and smallest positive numbers which can be stored in this format.

8. (Goldberg [25, Theorem. 7].) When $\beta = 2$, if m and n are integers with $m < 2^{p-1}$ (p is the number of bits in the mantissa) and n has the special form $n = 2^i + 2^j$, then $fl((m\cdot n)\cdot n) = 1$ provided that floating-point operations are exactly rounded to nearest. The sequence of possible values of n start with $1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 16, 17$. Test the theorem on your computer for these numbers.

9. Let *pi* be the closest floating point number to π in double precision IEEE 754

standard. Find a sufficiently accurate approximation to π from a table and show that $\pi - pi \approx 1.2246 \cdot 10^{-16}$. What value do you get on your computer for $\sin \pi$?

10. (A. Edelman.) Let x, $1 \le x < 2$, be a floating point number in IEEE double precision arithmetic. Show that $fl(x \cdot fl(1/x))$ is either 1 or $1 - \epsilon_M/2$, where $\epsilon_M = 2^{-52}$ (the machine epsilon).

11. (N. J. Higham.) Let a and b be floating point numbers with $a \le b$. Show that the inequalities $a \le fl((a+b)/2) \le b$ can be violated in base 10 arithmetic. Show that $a \le fl(a + (b-a)/2) \le b$ in base β arithmetic, assuming the use of a guard digit.

12. (J.-M. Muller) A rational approximation of $\tan x$ in $[-\pi/4, \pi/4]$ is

$$r(x) = \frac{(0.99999\,99328 - 0.09587\,5045x^2)x}{1 - (0.42920\,9672 + 0.00974\,3234x^2)x^2}.$$

Determine the approximate maximum error of this approximation by comparing with the function on your system on 100 equidistant points in $[0, \pi/4]$.

13. (a) Show how on a binary computer the exponential function can be approximated by first performing a range reduction based on the relation $e^x = 2^y$, $y = x/\ln 2$, and then approximating 2^y on $y \in [0, 1/2]$.

(b) Show that since 2^y satisfies $2^{-y} = 1/2^y$ a rational function $r(y)$ approximating 2^y should have the form

$$r(y) = \frac{q(y^2) + ys(y^2)}{q(y^2) - ys(y^2)},$$

where q and s are polynomials.

(c) Suppose the $r(y)$ in (b) is used for approximating 2^y with

$$q(y) = 20.81892\,37930\,062 + y,$$
$$s(y) = 7.21528\,91511\,493 + 0.05769\,00723\,731y.$$

How many additions, multiplications and divisions are needed in this case to evaluate $r(y)$? Investigate the accuracy achieved for $y \in [0, 1/2]$.

2.3 Accuracy and Rounding Errors

2.3.1 Floating Point Arithmetic

It is useful to have a model of how the basic floating point operations are carried out. If x and y are two floating point numbers, we denote by

$$fl\,(x+y), \quad fl\,(x-y), \quad fl\,(x \cdot y), \quad fl\,(x/y)$$

the results of floating addition, subtraction, multiplication, and division, which the machine stores in memory (after rounding or chopping). We will in the following assume that underflow or overflow does not occur. and that the following **standard model** for the arithmetic holds:

Definition 2.3.1.
Assume that $x, y \in F$. Then in the **standard model** *it holds*

$$fl\,(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \le u, \tag{2.3.1}$$

where u is the unit roundoff and "op" stands for one of the four elementary operations $+$, $-$, $\cdot$, and $/$.

The standard model holds with the default rounding mode for computers implementing the IEEE 754 standard. In this case we also have

$$fl\,(\sqrt{x}) = \sqrt{x}(1+\delta), \quad |\delta| \le u, \tag{2.3.2}$$

If a guard digit is lacking then instead of (2.3.1) only the weaker model

$$fl\,(x \text{ op } y) = x(1+\delta_1) \text{ op } y(1+\delta_2), \quad |\delta_i| \le u, \tag{2.3.3}$$

holds for addition/subtraction. The lack of a guard digit is a serious drawback and can lead to damaging inaccuracy caused by cancellation. Many algorithms can be proved to work satisfactorily only if the standard model (2.3.1) holds. We remark that on current computers multiplication is as fast as addition/subtraction. Division usually is 5–10 times slower than a multiply and a square root about twice slower than division.

Some earlier computers lack a guard digit in addition/subtraction. Notable examples are several models of Cray computers (Cray 1,2, X-MP,Y-MP, and C90) before 1995, which were designed to have the highest possible floating-point performance. The IBM 360, which used a hexadecimal system, lacked a (hexadecimal) guard digit between 1964–1967. The consequences turned out to be so intolerable that a guard digit had to be retrofitted.

Sometimes the floating point computation is more precise than what the standard model assumes. An obvious example is that when the exact value x op y can be represented as a floating point number there is no rounding error at all.

Some computers can perform a *fused multiply-add* operation, i.e. an expression of the form $a \times x + y$ can be evaluated with just one instruction and there is only *one rounding error* at the end

$$fl\,(a \times x + y) = (a \times x + y)(1+\delta), \quad |\delta| \le u.$$

Fused multiply-add can be used to advantage in many algorithms. For example, Horner's rule to evaluate the polynomial $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$, which uses the recurrence relation $b_0 = a_0$, $b_i = b_{i-1} \cdot x + a_i$, $i = 1 : n$, needs only n fused multiply-add operations.

It is important to realize that these floating point operations have, to some degree, other properties than the exact arithmetic operations. For example, floating point addition and multiplication are commutative, but not associative and the distributive law also fails for them. This makes the analysis of floating point computations more difficult.

Example 2.3.1.

To show that associativity does not, in general, hold for floating addition, consider adding the three numbers

$$a = 0.1234567 \cdot 10^0, \qquad b = 0.4711325 \cdot 10^4, \qquad c = -b.$$

in a decimal floating point system with $t = 7$ digits in the mantissa. The following scheme indicates how floating point addition is performed:

$$fl\,(b + c) = 0, \qquad fl\,(a + fl\,(b + c)) = a = 0.1234567 \cdot 10^0$$

$$\begin{array}{rcr|r} a & = & 0.0000123 & 4567 \cdot 10^4 \\ +b & = & 0.4711325 & \cdot 10^4 \\ \hline fl\,(a+b) & = & 0.4711448 & \cdot 10^4 \\ c & = & -0.4711325 & \cdot 10^4 \end{array}$$

The last four digits to the right of the vertical line are lost by **outshifting**, and

$$fl\,(fl\,(a + b) + c) = 0.0000123 \cdot 10^4 = 0.1230000 \cdot 10^0 \neq fl\,(a + fl\,(b + c)).$$

An interesting fact is that assuming a guard digit is used *floating point subtraction of two sufficiently close numbers is always exact.*

Lemma 2.3.2 (Sterbenz**).**

Let the floating point numbers x and y satisfy

$$y/2 \leq x \leq 2y.$$

If subtraction is performed with a guard digit then $fl(x - y) = x - y$, unless $x - y$ underflows.

Proof. By the assumption the exponent of x and y in the floating point representations of x and y can differ at most by one unit. If the exponent is the same then the exact result will be computed. Therefore assume the exponents differ by one. After scaling and, if necessary, interchanging x and y it holds that $x/2 \leq y \leq x < 2$ and the exact difference $z = x - y$ is of the form

$$\begin{array}{rcl} x & = & x_1.x_2 \ldots x_t \\ y & = & \;\;0\,.y_1 \ldots y_{t-1}y_t \\ \hline z & = & z_1.z_2 \ldots z_t z_{t+1} \end{array}$$

But from the assumption $x/2 - y \leq 0$ or $x - y \leq y$. Hence we must have $z_1 = 0$, so after shifting the exact result is obtained also in this case. □

With gradual underflow, as in the IEEE 754 standard, the condition that $x - y$ does not underflow can be dropped.

Example 2.3.2.

A corresponding result holds for any base β. For example, using four digit floating decimal arithmetic we get with guard digit

$$fl\,(0.1000 \cdot 10^1 - 0.9999) = 0.0001 = 1.000 \cdot 10^{-4},$$

(exact) but without guard digit

$$fl\,(0.1000 \cdot 10^1 - 0.9999) = (0.1000 - 0.0999)10^1 = 0.0001 \cdot 10^1 = 1.000 \cdot 10^{-3}.$$

The last result satisfies equation (2.3.3) with $|\delta_i| \leq 0.5 \cdot 10^{-3}$ since $0.10005 \cdot 10^1 - 0.9995 = 10^{-3}$.

Outshiftings are common causes of loss of information that may lead to **catastrophic cancellation** later, in the computations of a quantity that one would have liked to obtain with good relative accuracy.

Example 2.3.3.

An example where the result of Lemma 2.3.2 can be used to advantage is in computing compounded interest. Consider depositing the amount c every day on an account with an interest rate i compounded daily. Then with the accumulated capital at the end of the year equals

$$c[(1+x)^n - 1]/x, \quad x = i/n \ll 1,$$

and $n = 365$. Using this formula does not give accurate results. The reason is that a rounding error occurs in computing $fl(1+x) = 1 + \bar{x}$ and low order bits of x is lost. For example, if $i = 0.06$ then $i/n = 0.0001643836$ and in decimal arithmetic using six digits when this is added to one we get $fl(1+i/n) = 1.000164$ so four low order digits are lost.

The problem then is to accurately compute $(1+x)^n = \exp(n\ln(1+x))$. The formula

$$\ln(1+x) = \begin{cases} x, & \text{if } fl\,(1+x) = 1; \\ x\dfrac{\ln(1+x)}{(1+x)-1}, & \text{otherwise.} \end{cases} \tag{2.3.4}$$

can be shown to yield accurate results when $x \in [0, 3/4]$ provided subtraction is performed with a guard digit and the computed value of $\ln(1+x)$ equals the exact result rounded; see Goldberg [25, p. 12].

To check this formula we recall that the base e of the natural logarithm can be defined by the limit

$$e = \lim_{n\to\infty}(1 + 1/n)^n$$

In Fig. 2.3.1 we show computed values, using double precision floating point arithmetic, of the sequence $|(1+1/n)^n - e|$ for $n = 10^p$, $p = 1:14$. More precisely, the expression was computed as

$$|\exp(n\ln(1+1/n)) - \exp(1)|.$$

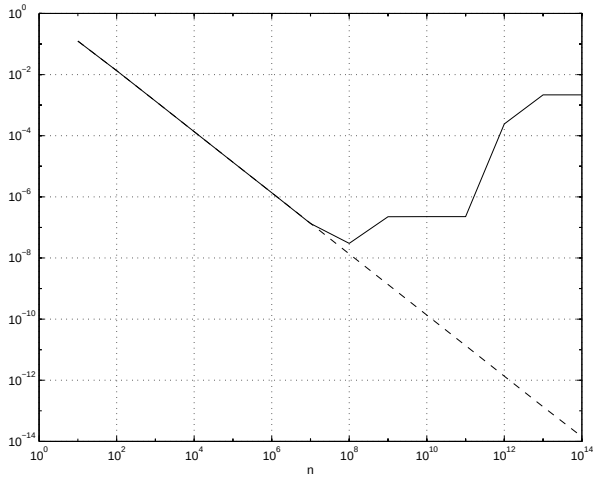


Figure 2.3.1. *Computed values for $n = 10^p$, $p = 1 : 14$, of the sequences: solid line $|(1 + 1/n)^n - e|$; dashed line $|\exp(n \ln(1 + 1/n)) - e|$ using* (2.3.4).

The smallest difference $3 \cdot 10^{-8}$ occurs for $n = 10^8$, for which about half the number of bits in $x = 1/n$ are lost. For larger values of n rounding errors destroy the convergence. However, using (2.3.4) we obtain correct results for all values of n! (The Maclaurin series $\ln(1 + x) = x - x^2/2 + x^3/3 - x^4/4 + \cdots$ will also give good results; see Computer Exercise 1.)

A fundamental insight from the above examples can be expressed in the following way:

> *"mathematically equivalent" formulas or algorithms are not in general "numerically equivalent".*

This adds a new dimension to calculations in finite precision arithmetic and it will be a recurrent theme in the analysis of algorithms in this book!

By **mathematical equivalence** of two algorithms we mean here that the algorithms give exactly the same results from the same input data, if the computations are made without rounding error ("with infinitely many digits"). One algorithm can then, as a rule, formally be derived from the other using the rules of algebra for real numbers, and with the help of mathematical identities. Two algorithms are **numerically equivalent** if their respective floating point results, using the same input data are the same.

In error analysis for compound arithmetic expressions based on the standard model (2.3.1) one often needs an upper bound for quantities of this form

$$\epsilon \equiv |(1 + \delta_1)(1 + \delta_2) \cdots (1 + \delta_n) - 1|, \quad |\delta_i| \leq u, \quad i = 1 : n.$$

Then $\epsilon \le (1+u)^n - 1$. Assuming that $nu < 1$ an elementary calculation gives

$$\begin{aligned}(1+u)^n - 1 &= nu + \frac{n(n-1)}{2!}u^2 + \cdots + \binom{n}{k}u^k + \cdots \\ &< nu\Big(1 + \frac{nu}{2} + \cdots + \Big(\frac{nu}{2}\Big)^{k-1} + \cdots\Big) = \frac{nu}{1-nu/2} \end{aligned} \tag{2.3.5}$$

Similarly it can be shown that $(1-u)^{-n} - 1 < nu/(1-nu)$, and the following useful result follows:

Lemma 2.3.3. [N. J. Higham [29, Lemma 3.1]]
Let $|\delta_i| \le u$, $\rho_i = \pm 1$, $i = 1{:}n$, and

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n.$$

If $nu < 1$, then $|\theta_n| < \gamma_n$, where

$$\gamma_n = nu/(1-nu). \tag{2.3.6}$$

Complex arithmetic can be reduced to real arithmetic. Let $x = a + ib$ and $y = c + id$ be two complex numbers. Then we have:

$$\begin{aligned} x \pm y &= a \pm c + i(b \pm d), \\ xy &= (ac - bd) + i(ad + bc), \\ x/y &= \frac{ac+bd}{c^2+d^2} + i\frac{bc-ad}{c^2+d^2}, \end{aligned} \tag{2.3.7}$$

Using the above formula complex addition (subtraction) needs two real additions and multiplying two complex numbers requires four real multiplications

Lemma 2.3.4. *Assuming the standard model* (2.3.1) *the complex operations computed according to* (2.3.7) *satisfy*

$$\begin{aligned} fl\,(x \pm y) &= (x \pm y)(1+\delta), \quad |\delta| \le u, \\ fl\,(xy) &= xy(1+\delta), \quad |\delta| \le \sqrt{2}\gamma_2, \\ fl\,(x/y) &= x/y(1+\delta), \quad |\delta| \le \sqrt{2}\gamma_4, \end{aligned} \tag{2.3.8}$$

where δ is a complex number and γ_n is defined in (2.3.6).

Proof. See Higham [29, Sec. 3.6]. □

The square root of a complex number $u + iv = \sqrt{x+iy}$ is given by

$$u = \left(\frac{r+x}{2}\right)^{1/2}, \qquad v = \left(\frac{r-x}{2}\right)^{1/2}, \quad r = \sqrt{x^2+y^2}. \tag{2.3.9}$$

When $x > 0$ there will be cancellation when computing v, which can be severe if also $|x| \gg |y|$ (cf. Sec. 2.3.5). To avoid this we note that $uv = \sqrt{r^2 - x^2}/2 = y/2$, so v can be computed from $v = y/(2u)$. When $x < 0$ we instead compute v from (2.3.9) and set $u = y/(2v)$.

Most rounding error analysis given in this book are formulated for real arithmetic. Since the bounds in Lemma 2.3.4 are of the same form as the standard model for real arithmetic, these can simply be extended to complex arithmetic.

In some cases it may be desirable to avoid complex arithmetic when working with complex matrices. This can be achieved in a simple way by replacing the complex matrices and vectors by real ones of twice the order. Suppose that a complex matrix $A \in \mathbf{C}^{n\times n}$ and a complex vector $z \in \mathbf{C}^n$ are given, where

$$A = B + iC, \qquad z = x + iy,$$

with real B, C, x and y. Form the real matrix $\tilde{A} \in \mathbf{R}^{2n\times 2n}$ and real vector $\tilde{z} \in \mathbf{R}^{2n}$ defined by

$$\tilde{A} = \begin{pmatrix} B & -C \\ C & B \end{pmatrix}, \qquad \tilde{z} = \begin{pmatrix} x \\ y \end{pmatrix}.$$

It is easy to verify the following rules

$$\widetilde{(Az)} = \tilde{A}\tilde{z}, \qquad \widetilde{(AB)} = \tilde{A}\tilde{B}, \qquad \widetilde{(A^{-1})} = (\tilde{A})^{-1},$$

etc. Thus we can solve complex valued matrix problems using algorithms for the real case. However, this incurs a penalty in storage and arithmetic operations.

2.3.2 Basic Rounding Error Results

We now use the notation of Sec. 2.3.1 and the standard model of floating point arithmetic (Definition 2.3.1) to carry out rounding error analysis of some basic computations. Most but not all results are still true if only the weaker bound (2.3.3) hold for addition and subtraction. Note that $fl\,(x \text{ op } y) = (x \text{ op } y)(1+\delta)$, $|\delta| \le u$, can be interpreted for multiplication to mean that $fl\,(x \cdot y)$ is the *exact result* of $x \cdot y(1+\delta)$ for some δ, $|\delta| \le u$. In the same way, the results using the three other operations can be interpreted as *the result of exact operations where the operands have been perturbed by a relative amount which does not exceed* u. In **backward error analysis** (see Sec. 2.4.5) one applies the above interpretation step by step backwards in an algorithm.

By repeatedly using formula (2.3.1) in case of multiplication, one can show that

$$fl\,(x_1x_2\cdots x_n) = x_1x_2(1+\delta_2)x_3(1+\delta_3)\cdots x_n(1+\delta_n),$$
$$|\delta_i| \le u, \quad i = 2:n.$$

That is, the computed **product** $fl\,(x_1x_2\cdots x_n)$ is *exactly* equal to a product of the factors

$$\tilde{x}_1 = x_1, \quad \tilde{x}_i = x_i(1+\delta_i), \quad i = 2:n.$$

Using the estimate and notation of (2.3.6) it follows from this analysis that

$$|fl\,(x_1x_2\cdots x_n) - x_1x_2\cdots x_n| < \gamma_{n-1}|x_1x_2\cdots x_n|, \tag{2.3.10}$$

which bounds the **forward error** of the computed result.

For a **sum** of n floating point numbers similar results can be derived. If the sum is computed in the natural order we have

$$\begin{aligned} &fl\,(\cdots(((x_1+x_2)+x_3)+\cdots+x_n)) \\ &\qquad = x_1(1+\delta_1)+x_2(1+\delta_2)+\cdots+x_n(1+\delta_n), \end{aligned}$$

where

$$|\delta_1| < \gamma_{n-1}, \qquad |\delta_i| < \gamma_{n+1-i}. \quad i = 2:n,$$

and thus the computed sum is *the exact sum* of the numbers $x_i(1+\delta_i)$. This also gives an estimate of the forward error

$$\begin{aligned} &|fl\,(\cdots(((x_1+x_2)+x_3)+\cdots+x_n)) - (x_1+x_2+x_3+\cdots+x_n)| \\ &\qquad < \sum_{i=1}^{n} \gamma_{n+1-i}|x_i| \le \gamma_{n-1}\sum_{i=1}^{n}|x_i|, \end{aligned} \tag{2.3.11}$$

where the last upper bound holds independent of the summation order.

Notice that to minimize the first upper bound in equation (2.3.11), the terms *should be added in increasing order of magnitude*! For large n an even better bound can be shown if the summation is done using the divide-and-conquer technique described in Sec. 1.3.2; see Problem 5.

Example 2.3.4.

Using a hexadecimal machine ($\beta = 16$), with $t = 6$ and chopping ($u = 16^{-5} \approx 10^{-6}$) one computed

$$\sum_{n=1}^{10{,}000} n^{-2} \approx 1.644834$$

in two different orders. Using the natural summation order $n = 1, 2, 3, \ldots$ the error was $1.317 \cdot 10^{-3}$. Summing in the opposite order $n = 10,000, 9,999, 9,998 \ldots$ the error was reduced to $2 \cdot 10^{-6}$. This was not unexpected. Each operation is an addition, where the partial sum s is increased by n^{-2}. Thus, in each operation, one commits an error of about $s \cdot u$, and all these errors are added. Using the first summation order, we have $1 \le s \le 2$ in every step, but using the other order of summation we have $s < 10^{-2}$ in $9,900$ of the $10,000$ additions.

Similar bounds for roundoff errors can easily be derived for basic vector and matrix operations; see Wilkinson [46, pp. 114–118]. For an **inner product** x^Ty computed in the natural order we have

$$fl\,(x^Ty) = x_1y_1(1+\delta_1) + x_2y_2(1+\delta_2) + \cdots + x_ny_n(1+\delta_n)$$

where

$$|\delta_1| < \gamma_n, \qquad |\delta_r| < \gamma_{n+2-i}, \quad i = 2:n.$$

The corresponding forward error bound becomes

$$|fl\,(x^Ty) - x^Ty| < \sum_{i=1}^{n} \gamma_{n+2-i}|x_i||y_i| < \gamma_n \sum_{i=1}^{n} |x_i||y_i|,$$

If we let $|x|$, $|y|$ denote vectors with elements $|x_i|$, $|y_i|$ the last estimate can be written in the simple form

$$|fl\,(x^Ty) - x^Ty| < \gamma_n |x^T||y|. \tag{2.3.12}$$

This bound is independent of the summation order and holds also for the weaker model (2.3.3) valid with no guard digit rounding.

The outer product of two vectors $x, y \in \mathbf{R}^n$ is the matrix $xy^T = (x_iy_j)$. In floating point arithmetic we compute the elements $fl\,(x_iy_j) = x_iy_j(1+\delta_{ij})$, $\delta_{ij} \leq u$, and so

$$|fl\,(xy^T) - xy^T| \leq u\,|xy^T|. \tag{2.3.13}$$

This is a satisfactory result for many purposes, but the computed result is not in general a rank one matrix and it is not possible to find Δx and Δy such that $fl(xy^T) = (x + \Delta x)(x + \Delta y)^T$.

The product of two t digit floating point numbers can be exactly represented with at most $2t$ digits. This allows inner products to be computed in extended precision without much extra cost. If fl_e denotes computation with extended precision and u_e the corresponding unit roundoff then the forward error bound for an inner product becomes

$$|fl\,(fl_e((x^Ty)) - x^Ty| < u|x^Ty| + \frac{nu_e}{1 - nu_e/2}(1+u)|x^T||y|, \tag{2.3.14}$$

where the first term comes form the final rounding. If $|x^T||y| \leq u|x^Ty|$ then the computed inner product is almost as accurate as the correctly rounded exact result. These accurate inner products can be used to improve accuracy by iterative refinement in many linear algebra problems (see Chapters 7–9, Volume II). However, since computations in extended precision are machine dependent it has been difficult to make such programs portable.[8] The recent development of Extended and Mixed Precision BLAS (Basic Linear Algebra Subroutines) (see [33] may now make this more feasible!

Similar error bounds can easily be obtained for matrix multiplication. Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$, and denote by $|A|$ and $|B|$ matrices with elements $|a_{ij}|$ and $|b_{ij}|$. Then it holds that

$$|fl\,(AB) - AB| < \gamma_n |A||B|. \tag{2.3.15}$$

where the inequality is to be interpreted elementwise. Often we shall need bounds for some norm of the error matrix. From (2.3.15) it follows that

$$\|fl\,(AB) - AB\| < \gamma_n \| \,|A|\,\| \,\| \,|B|\,\|. \tag{2.3.16}$$

[8] It was suggested that the IEEE 754 standard should require inner products to be precisely specified, but that did not happen.

Hence, for the 1-norm, ∞-norm and the Frobenius norm we have

$$\|fl(AB) - AB\| < \gamma_n \|A\| \, \|B\|. \tag{2.3.17}$$

but unless A and B have non-negative elements, we have for the 2-norm only the weaker bound

$$\|fl(AB) - AB\|_2 < n\gamma_n \|A\|_2 \, \|B\|_2. \tag{2.3.18}$$

To reduce the effects of rounding errors in computing a sum $\sum_{i=1}^{n} x_i$ one can use **compensated summation**. In this algorithm the rounding error in each addition is estimated and then compensated for with a correction term. Compensated summation can be useful when a large number of small terms are to be added as in numerical quadrature. Another example is the case in the numerical solution of initial value problems for ordinary differential equations. Note that in this application the terms have to be added in the order in which they are generated.

Compensated is based on the possibility to *simulate double precision floating point addition in single precision arithmetic*. To illustrate the basic idea we take as in Example 2.3.1

$$a = 0.1234567 \cdot 10^0, \qquad b = 0.4711325 \cdot 10^4,$$

so that $s = fl(a + b) = 0.4711448 \cdot 10^4$, Suppose we form

$$c = fl(fl(b - s) + a) = -0.1230000 \cdot 10^0 + 0.1234567 \cdot 10^0 = 4567000 \cdot 10^{-3}.$$

Note that the variable c is computed without error and picks up the information that was lost in the operation $fl(a + b)$.

The following algorithm uses this idea to accurately computing $\sum_{i=1}^{n} x_i$:

Algorithm 2.3.1 Compensated Summation.

$s := x_1; \quad c := 0;$
for $i = 2 : n$
  $y := c + x_i;$
  $t := s + y;$
  $c := (s - t) + y;$
  $s := t;$
end

It can be proved (see Goldberg [25, 1991]) that on binary machines with a guard digit the computed sum satisfies

$$s = \sum_{i=1}^{n} (1 + \xi_i) x_i, \quad |\xi_i| < 2u + O(nu^2). \tag{2.3.19}$$

This formulation is a typical example of a backward error analysis; see Sec. 2.4.5. The single precision term in the error bound is independent of n.

2.3.3 Statistical Models for Rounding Errors

The bounds for the accumulated rounding error we have derived so far are estimates of the **maximal error**. These bounds ignore the sign of the errors and tend to be much too pessimistic when the number of variables is large. They can still give valuable insight into the behavior of a method and be used for the purpose of comparing different method.

An alternative is a statistical analysis of rounding errors, which is based on the assumption that rounding errors are independent and have some statistical distribution. It was observed already in the 1950s that rounding errors occurring in the solution of differential equations *are not random and often are strongly correlated*. This does not in itself preclude that useful information can sometimes be obtained by modeling them by random uncorrelated variables! For example, in many computational situations and scientific experiments, where the error can be considered to have arisen from the addition of a large number of *independent* error sources of about the same magnitude an assumption that the errors are normally distributed is justified.

Example 2.3.5.

Fig. 2.3.2 illustrates the effect of rounding errors on the evaluation of two different expressions for the polynomial $p(x) = (x-1)^5$ for $x \in [0.999, 1.001]$, using a machine precision of about $2.2 \cdot 10^{-16}$. Among other things it shows that the monotonicity of a function can be lost due to rounding errors. The model of rounding errors as independent random variables works well in this example. It is obvious that it would be impossible to locate the zero of $p(x)$ to a precision better than about $(0.5 \cdot 10^{-14})^{1/6} \approx 0.0007$ using the expanded form of $p(x)$. However, using the expression $p(x) = (1-x)^5$ function values can be evaluated with constant *relative* precision even close to $x = 1$, and the problem disappears!

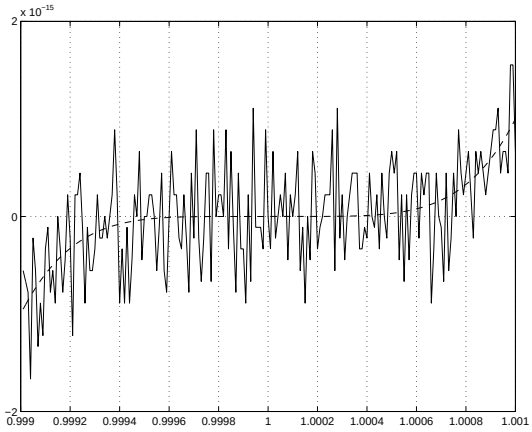


Figure 2.3.2. *Calculated values of a polynomial: solid line* $p(x) = x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1 = 0$*; dashed line* $p(x) = (x-1)^5$.

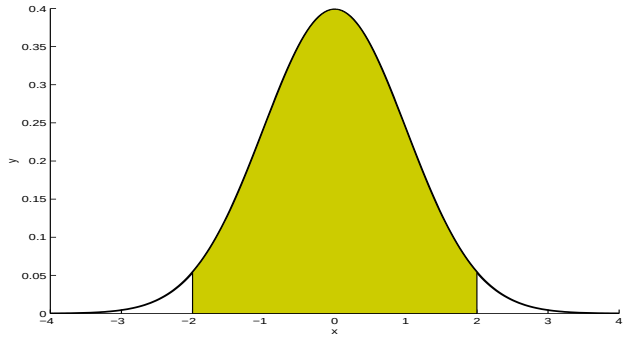


Figure 2.3.3. *The frequency function of the normal distribution for* $\sigma = 1$.

The theory of **standard error** is based on probability theory and will not be treated in detail here. The standard error of an estimate of a given quantity is the same as the *standard deviation of its sampling distribution.*

If in a sum $y = \sum_{i=1}^{n} x_i$ each x_i has error $|\Delta_i| \leq \delta$, then the maximum error bound for y is $n\delta$. Thus, *the maximal error grows proportionally to* n. If n is large—for example, $n = 1000$—then it is in fact highly improbable that the real error will be anywhere near $n\delta$, since that bound is attained only when every Δx_i has the same sign and the same maximal magnitude. Observe, though, that if positive numbers are added, each of which has been abridged to t decimals by chopping, then each Δx_i has the same sign and a magnitude which is on the average $\frac{1}{2}\delta$, where $\delta = 10^{-t}$. Thus, the real error is often about 500δ.

If the numbers are rounded instead of chopped, and if one can assume that the errors in the various terms are stochastically independent with standard deviation ϵ, then the standard error in y becomes (see Theorem 2.4.5)

$$(\epsilon^2 + \epsilon^2 + \ldots + \epsilon^2)^{1/2} = \epsilon\sqrt{n}.$$

Thus the standard error of the sum grows only proportionally to $\sqrt{n}$. This supports the following rule of thumb, suggested by Wilkinson [45, p. 26], that *if a rounding error analysis gives a bound* $f(n)u$ *for the maximum error, then one can expect the real error to be of size* $\sqrt{f(n)}u$.

If $n \gg 1$, then the error in y is, under the assumptions made above, approximately normally distributed with standard deviation $\sigma = \epsilon\sqrt{n}$. The corresponding frequency function,

$$f(t) = \frac{1}{\sqrt{2\pi}} e^{-t^2/2},$$

is illustrated in Fig. 2.3.3; the curve shown there is also called the Gauss curve. The assumption that the error is normally distributed with standard deviation σ means, e.g., that the statement "the magnitude of the error is greater than 2σ" (see the shaded area of Fig. 2.3.3) is false in about only 5 % of all cases. (the clear area under the curve). More generally, the assertion that the magnitude of the error is less than σ, 2σ, and 3σ respectively, is about 32%, 5%, and 0.27%.

One can show that if the individual terms in a sum $y = \sum_{i=1}^{n} x_i$ have a **uniform** probability distribution in the interval $[-\frac{1}{2}\delta, \frac{1}{2}\delta]$, then the standard deviation of an individual term is $\delta/12$. Therefore, in only about 5% of the cases is the error in the sum of $1,000$ terms greater than $2\delta\sqrt{1000/12} \approx 18\delta$, which can be compared to the maximum error 500δ. This shows that rounding can be far superior to chopping when a statistical interpretation (especially, the assumption of independence) can be given to the principal sources of errors. Observe that, in the above, we have only considered the propagation of errors which were present in the original data, and have ignored the effect of possible round-off errors in the additions themselves.

In science and technology, one generally should be careful to discriminate between **systematic errors** and **random errors**. A systematic error can, e.g., be produced by insufficiencies in the construction of an instrument; such an error is the same in each trial. Random errors depend on the variation in the experimental environment which cannot be controlled; then the formula for standard errors is used. For systematic errors, however, the formula for maximal error (2.4.5) should be used.

2.3.4 Avoiding Overflow

In the rare cases when input and output data are so large or small in magnitude that the range of the machine is not sufficient, one can, for example, use higher precision or else work with logarithms or some other transformation of the data. One should, however, keep in mind the risk that *intermediate results* in a calculation can produce an exponent which is too large exponent overflow or too small underflow for the floating point system of the machine. Different machines take different actions in such situations, as well for division by zero. Too small an exponent is usually, but not always, unprovoking. If the machine does not signal underflow, but simply sets the result equal to zero, there is a risk of harmful consequences. Occasionally, "unexplainable errors" in output data are caused by underflow somewhere in the computations.

The **Pythagorean sum** $c = \sqrt{a^2 + b^2}$ occurs frequently, e.g., in conversion to polar coordinates and in computing the complex modulus and complex multiplication. If the obvious algorithm is used, then damaging underflows and overflows may occur in the squaring of a and b even if a and b and the result c are well within the range of the floating point system used. This can be avoided by using instead the algorithm: If $a = b = 0$ then $c = 0$; otherwise set $p = \max(|a|, |b|)$, $q = \min(|a|, |b|)$, and compute

$$\rho = q/p; \qquad c = p\sqrt{1 + \rho^2}. \tag{2.3.20}$$

Example 2.3.6.

The formula (2.3.7) for complex division suffers from the problem that intermediate results can overflow even if the final result is well within the range of the floating point system. This problem can be avoided by rewriting the formula as for

the Pythagorean sum: If $|c| > |d|$ then compute

$$\frac{a+ib}{c+id} = \frac{a+be}{r} + i\frac{b-ae}{r}, \quad e = d/c, \quad r = c+de.$$

If $|d| > |c|$ then $e = c/d$ is computed and a corresponding formula used.

Similar precautions are also needed for computing the Euclidian length (norm) of a vector $\|x\|_2 = \left(\sum_{i=1}^n x_i^2\right)^{1/2}$, $x \neq 0$. We could avoid overflows by first finding $x_{max} = \max_{1\le i\le n} |x_i|$ and then forming

$$s = \sum_{i=1}^{n} (x_i/x_{max})^2, \qquad \|x\|_2 = x_{max}\sqrt{s}. \tag{2.3.21}$$

This has the drawback of needing *two passes* through the data.

Example 2.3.7.

The following algorithm requiring only one pass is due to S. J. Hammarling:

$t = 0;\quad s = 1;$
for $i = 1 : n$
  if $|x_i| > t$
    $s = 1 + s(t/x_i)^2;\quad t = |x_i|;$
  else
    $s = s + s(x_i/t)^2;$
  end
end
$\|x\|_2 = t\sqrt{s};$

On the other hand this code does not vectorize and can therefore be slower if implemented on a vector computer.

2.3.5 Cancellation of Terms

One very common reason for poor accuracy in the result of a calculation is that somewhere a subtraction has been carried out in which the difference between the operands is considerably less than either of the operands.

Consider the computation of $y = x_1 - x_2$ where $\tilde{x}_1 = x_1 + \Delta x_1$, $\tilde{x}_2 = x_2 + \Delta x_2$ are approximations to the exact values. If the operation is carried out exactly the result is $\tilde{y} = y + \Delta y$, where $\Delta y = \Delta x_1 - \Delta x_2$. But, since the errors Δx_1 and Δx_2 can have opposite sign, the best error bound for $\tilde{y}$ is

$$|\Delta y| \le |\Delta x_1| + |\Delta x_2|. \tag{2.3.22}$$

Notice the plus sign! Hence for the relative error we have

$$\left|\frac{\Delta y}{y}\right| \le \frac{|\Delta x_1| + |\Delta x_2|}{|x_1 - x_2|}. \tag{2.3.23}$$

This shows that *there can be very poor relative accuracy in the difference between two nearly equal numbers.* This phenomenon is called **cancellation of terms**.

Example 2.3.8.

For computing the roots of the quadratic equation $ax^2 + bx + c = 0$ $(a \neq 0)$ we have the well-known formula

$$r_{1,2} = \big(-b \pm \sqrt{b^2 - 4ac}\big)/(2a).$$

Hence the quadratic equation $x^2 - 56x + 1 = 0$ has the two roots

$$r_1 = 28 + \sqrt{783} \approx 28 + 27.982 = 55.982 \pm \tfrac{1}{2}10^{-3}.$$
$$r_2 = 28 - \sqrt{783} \approx 28 - 27.982 = 0.018 \pm \tfrac{1}{2}10^{-3}.$$

In spite of the fact that the square root is given to five digits, we get only two significant digits in r_2, while the relative error in r_1 is less than 10^{-5}. Notice that the subtraction in the calculation of r_2 has been carried out exactly.
The cancellation in the subtraction only gives an indication of the unhappy consequence of a loss of information in previous steps, due to the rounding of one of the operands, and is not the cause of the inaccuracy.

In general one should if possible try to avoid cancellation, as in the example above, by an appropriate **rewriting of formulas**, or by other changes in the algorithm. For the quadratic equation above, by comparing coefficients in
$x^2 + (b/a)x + c/a = (x - r_1)(x - r_2) = x^2 - (r_1 + r_2) + r_1 r_2$,
we get the dependence between coefficients and roots

$$r_1 + r_2 = -b/a, \qquad r_1 r_2 = c/a. \tag{2.3.24}$$

Computing the root of *smaller magnitude* from the latter of these relations, we get $x_2 = 1/55.982 = 0.0178629 \pm 0.0000002$, i.e., five significant digits instead of two. In general we can avoid cancellation by using the algorithm:

Algorithm 2.3.2 Solving a quadratic equation.

$d := b^2 - 4ac$;
if $d \geq 0$ % real roots
    $r_1 := -\text{sign}(b)\big(|b| + \sqrt{d}\big)/(2a)$;
    $r_2 := c/(a \cdot r_1)$;
else % complex roots $x + iy$
    $x := -b/(2a)$;
    $y := \sqrt{-d}/(2a)$;
end

Note that we define $\text{sign}(b) = 1$, if $b \geq 0$, else $\text{sign}(b) = -1$.[9] It can be shown that this algorithm computes *a slightly wrong solution to a slightly wrong problem.*

[9] In MATLAB *sign*$(0) = 0$, which can lead to failure of this algorithm!

Lemma 2.3.5.

Assume that the Algorithm 2.3.2 *is used to compute the roots* $r_{1,2}$ *of the quadratic equation* $ax^2 + bx + c = 0$. *Denote the computed roots by* $\bar{r}_{1,2}$ *and let* $\tilde{r}_{1,2}$ *be the exact roots of the nearby equation* $ax^2 + bx + \tilde{c} = 0$, *where* $|\tilde{c} - c| \leq \gamma_2|\tilde{c}|$. *Then* $|\tilde{r}_i - \bar{r}_i| \leq \gamma_5|\tilde{r}_i|$, $i = 1, 2$.

Proof. See Kahan [30]. □

More generally, if $|\delta| \ll x$, then one should rewrite

$$\sqrt{x+\delta} - \sqrt{x} = \frac{x+\delta-x}{\sqrt{x+\delta}+\sqrt{x}} = \frac{\delta}{\sqrt{x+\delta}+\sqrt{x}}.$$

There are other exact ways of rewriting formulas which are as useful as the above; for example,

$$\cos(x+\delta) - \cos x = -2\sin(\delta/2)\sin(x+\delta/2).$$

If one cannot find an exact way of rewriting a given expression of the form $f(x + \delta) - f(x)$, it is often advantageous to use one or more terms in the Taylor series

$$f(x+\delta) - f(x) = f'(x)\delta + \frac{1}{2}f''(x)\delta^2 + \cdots$$

Example 2.3.9. (Cody [14])

To compute $\sin 22$ we first find $\lfloor 22/(\pi/2) \rfloor = 14$. It follows that $\sin 22 = -\sin x^*$, where $x^* = 22 - 14(\pi/2)$. Using the correctly rounded 10 digit approximation $\pi/2 = 1.57079\,6327$ we obtain

$$x^* = 22 - 1.57079\,6327 = 8.85142 \cdot 10^{-3}.$$

Here cancellation has taken place and the reduced argument has a maximal error of $7 \cdot 10^{-9}$, The actual error is slightly smaller since the correctly rounded value is $x^* = 8.85144\,8711 \cdot 10^{-3}$, which corresponds to a relative error in the computed $\sin 22$ of about $2.4 \cdot 10^{-6}$, in spite of using a ten digit approximation to $\pi/2$.

For very large arguments the relative error can be much larger. Techniques for carrying out accurate range reductions without actually needing multiple precision calculations are discussed by Muller [35]; see also Problem 9.

In previous examples we got a warning that cancellation would occur, since x_2 was found as the difference between two nearly equal numbers each of which was, relatively, much larger than the difference itself. In practice, one does not always get such a warning, for two reasons: first, in using a computer one has no direct contact with the individual steps of calculation; secondly, cancellation can be spread over a great number of operations. This may occur in computing a partial sum of an infinite series. For example, in a series where the size of some terms are many order of magnitude larger than the sum of the series small relative errors in the computation of the large terms can then produce large errors in the result.

Example 2.3.10.

Set $y_0 = 28$ and define y_n, for $n = 1 : 100$, by the recursion formula:

$$y_n = y_{n-1} - \sqrt{783}/100.$$

As previously, we use the approximate value 27.982 for the square root. We then compute with five decimals in each operation in order to make the effect of further rounding errors negligible. We get the same bad value for y_{100} that we got for x_1 in the previous example. Of all the subtractions, only the last would lead one to suspect cancellation, $y_{100} = 0.29782 - 0.27982 = 0.01800$, but this result in itself gives one no reason to suspect that only two digits are significant. (With four significant digits, the result is 0.01786.)

Review Questions

1. What is the standard model for floating point arithmetic? What weaker model holds if a guard digit is lacking?
2. Give examples to show that some of the axioms for arithmetic with real numbers do not always hold for floating point arithmetic.
3. (a) Give the results of a backward and forward error analysis for computing $fl\,(x_1 + x_2 + \cdots + x_n)$. It is assumed that the standard model holds.
   (b) Describe the idea in compensated summation.
4. Explain the terms "maximum error" and "standard error". What statistical assumption about rounding errors is often made, for example, when calculating the standard error in a sum due to rounding?
5. Explain, what is meant by "cancellation of terms". Give an example how this can be avoided by rewriting a formula.

Problems

1. Rewrite the following expression to avoid cancellation of terms:
   (a) $1 - \cos x$, $|x| \ll 1$; (b) $\sin x - \cos x$, $|x| \approx \pi/4$;
2. (a) The expression $x^2 - y^2$ exhibits catastrophic cancellation if $|x| \approx |y|$. Show that it is more accurate to evaluate it as $(x + y)(x - y)$.
   (b) Consider using the trigonometric identity $\sin^2 x + \cos^2 x = 1$ to compute $\cos x = (1 - \sin^2 x)^{1/2}$. For which arguments in the range $0 \le x \le \pi/4$ will this formula fail to give good accuracy?
3. The polar representation of a complex number is

$$z = x + iy = r(\sin\phi + \cos\phi) \equiv r \cdot e^{i\phi}.$$

Develop accurate formulas for computing this polar representation from x and y using real operations.

4. (Kahan) Show that with the use of fused multiply-add the algorithm

$$w := bc; \quad c := w - bc; \quad y := (ad - w) + c;$$

computes $x = \det \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with high relative accuracy.

5. Suppose that the sum $s = \sum_{i=1}^{n} x_i$, $n = 2^k$, is computed using the the divide-and-conquer technique described in Sec. 1.3.2. Show that this summation algorithm computes an exact sum

$$\bar{s} = \sum_{i=1}^{n} x_i(1 + \delta_i), \quad |\delta_i| \leq \tilde{u} \log_2 n.$$

Hence for large values of n this summation order can be much more accurate than the conventional order.

6. Show that for the evaluation of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ by Horner's rule the following roundoff error estimate holds:

$$|fl\,(p(x)) - p(x)| < \gamma_1 \sum_{i=0}^{n} (2i+1)|a_i||x|^i, \quad (2nu \leq 0.1).$$

7. In solving linear equations by Gaussian elimination there often occurs expressions of the form $s = (c - \sum_{i=1}^{n-1} a_i b_i)/d$. Show that by a slight extension of the result above shows that the computed $\bar{s}$ satisfies

$$\left| \bar{s}d - c + \sum_{i=1}^{n-1} a_i b_i \right| \leq \gamma_n \Big(|\bar{s}d| + \sum_{i=1}^{n-1} |a_i||b_i| \Big),$$

where the inequality holds independent of the summation order.

8. The zeros of the reduced cubic polynomial $z^3 + 3qz - 2r = 0$, can be found from the Cardano–Tartaglia formula:

$$z = \left(r + \sqrt{q^3 + r^2} \right)^{1/3} + \left(r - \sqrt{q^3 + r^2} \right)^{1/3}.$$

The two cubic roots are to be chosen so that their product equals $-q$. A real root is obtained if $q^3 + r^2 \geq 0$, which is the case unless all three roots are real and distinct.

The above formula can lead to cancellation. Rewrite it so that it becomes more suitable for numerical calculation and requires the calculation of only one cubic root.

9. (Eldén and Wittmeyer-Koch) In the interval reduction for computing $\sin x$ there can be a loss of accuracy through cancellation in the computation of the reduced argument $x^* = x - k \cdot \pi/2$ when k is large. A way to avoid

this without reverting to higher precision has been suggested by Cody and Waite [16]). Write

$$\pi/2 = \pi_0/2 + r,$$

where $\pi_0/2$ is *exactly representable* with a few digits in the (binary) floating point system. The reduced argument is now computed as $x^* = (x - k \cdot \pi_0/2) - kr$. Here, unless k is very large, the first term can be computed without rounding error. The rounding error in the second term is bounded by $k|r|\,u$, where u is the unit roundoff.
In IEEE single precision one takes

$$\pi_0/2 = 201/128 = 1.573125 = (10.1001001)_2, \quad r = 4.838267949 \cdot 10^{-4}$$

Estimate the relative error in the computed reduced argument x^* when $x = 1000$ and r is represented in IEEE single precision.

10. (W. Kahan [1983]) The area A of a triangle with sides equal to a, b, c is given by Heron's formula[10]

$$A = \sqrt{s(s-a)(s-b)(s-c)}, \quad s = (a+b+c)/2.$$

Show that this formula fails for needle-shaped triangles, using five digit decimal floating arithmetic and $a = 100.01$, $b = 99.995$, $c = 0.025$.
The following formula can be proved to work if addition/subtraction satisfies (2.3.20):
Order the sides so that $a \geq b \geq c$, and use

$$A = \frac{1}{4}\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))}.$$

Compute a correct result for the data above using this modified formula. If a person tells you that this gives an imaginary result if $a - b > c$, what do you answer him?

Computer Exercises

1. (a) To compute $\ln(1+x)$ for $0 < x \ll 1$, the Mclaurin series $\ln(1+x) = x - x^2/2 + x^3/3 - x^4/4 + x^5/5 + \cdots$ is useful. How many terms in the series are needed to get IEEE 754 double precision accuracy for all $x < 10^{-3}$?
(b) Show that $\ln(1+x) = \ln(1+y) - \ln(1-y)$, where $y = (x/2)/(1+x/2)$, and deduce that

$$\ln(1+x) = 2(y + y^3/3 + y^5/5 + \cdots).$$

How many terms in this series are needed for the same computation as in (a)?
Hint: Assume that the error from truncating the series can be estimated by the first neglected term.

[10] Heron (or Hero) of Alexandria, 1st century AD.

2. (a) Compute the derivative of the exponential function e^x at $x = 0$, by approximating with the difference quotients $(e^{x+h} - e^x)/h$, for $h = 2^{-i}$, $i = 1 : 20$. Explain your results.

(b) Same as in (a) but approximate instead with the central difference approximation $(e^{x+h} - e^{x-h})/(2h)$.

3. (W. Gautschi) Euler's constant $\gamma = 0.57721566490153286\ldots$ is defined as the limit

$$\gamma = \lim_{n\to\infty} \gamma_n, \quad \text{where} \quad \gamma_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} - \ln n.$$

Assuming that $\gamma - \gamma_n \sim cn^{-d}$, $n \to \infty$, for some constants c and $d > 0$, try to determine c and d experimentally on your computer.

4. In the statistical treatment of data, one often needs to compute the quantities

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

If the numbers x_i are the results of statistically independent measurements of a quantity with expected value m, then $\bar{x}$ is an estimate of m, whose standard deviation is estimated by $s/\sqrt{n-1}$.

(a) The computation of $\bar{x}$ and m using the formulas above have the drawback that they require two passes through the data x_i. Let α be *a provisional mean*, chosen as an approximation to $\bar{x}$, and set $x_i' = x_i - \alpha$. Show that the formulas

$$\bar{x} = \alpha + \frac{1}{n}\sum_{i=1}^{n} x_i', \qquad s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i')^2 - (\bar{x} - \alpha)^2.$$

hold for an arbitrary α.

(b) In sixteen measurements of a quantity x one got the following results:

i	x_i	i	x_i	i	x_i	i	x_i
1	546.85	5	546.81	9	546.96	13	546.84
2	546.79	6	546.82	10	546.94	14	546.86
3	546.82	7	546.88	11	546.84	15	546.84
4	546.78	8	546.89	12	546.82	16	546.84

Compute $\bar{x}$ and s^2 to two significant digits using $\alpha = 546.85$.

(c) In the computations in (b), one never needed more than three digits. If one uses the value $\alpha = 0$, how many digits is needed in $(x_i')^2$ in order to get two significant digits in s^2? If one uses five digits throughout the computations, why is the cancellation in the s^2 more fatal than the cancellation in the subtraction $x_i' - \alpha$? (one can even get negative values for s^2!)

(d) If we define

$$m_k = \frac{1}{k}\sum_{i=1}^{k} x_i, \qquad q_k = \sum_{i=1}^{k}(x_i - m_k)^2 = \sum_{i=1}^{k} x_i^2 - \frac{1}{k}\Big(\sum_{i=1}^{k} x_i\Big)^2,$$

then it holds that $\bar{x} = m_n$, and $s^2 = q_n/n$. Show the recursion formulas

$$m_1{=}x_1, \qquad m_k = m_{k-1} + (x_k - m_{k-1})/k$$
$$q_1{=}0, \qquad q_k = q_{k-1} + (x_k - m_{k-1})^2(k-1)/k$$

3. Compute the sum in Example 2.3.4 using the natural summation ordering in IEEE 754 double precision. Repeat the computations using compensated summation Algorithm 2.3.1.

2.4 Error Propagation

2.4.1 Numerical Problems, Methods and Algorithms

By a **numerical problem** we mean here a clear and unambiguous description of the *functional connection* between **input data** —that is, the "independent variables" in the problem—and **output data**—that is, the desired results. Input data and output data consist of a finite number of real (or complex) quantities and are thus representable by finite dimensional vectors. The functional connection can be expressed in either explicit or implicit form. We require for the following discussion also that *the output data should be uniquely determined and depend continuously on the input data.*

By an **algorithm**[11] for a given numerical problem we mean a *complete description of well-defined operations* through which each permissible input data vector is transformed into an output data vector. By "operations" we mean here arithmetic and logical operations, which a computer can perform, together with references to previously defined algorithms. It should be noted that, as the field of computing has developed, more and more complex functions (e.g., square root, circular and hyperbolic functions) are built into the hardware. In many programming environments operations like matrix multiplication, solution of linear systems, are considered as "elementary operations" and for the user appear as black boxes.

(The concept algorithm can be analogously defined for problems completely different from numerical problems, with other types of input data and fundamental operations—for example, inflection, merging of words, and other transformations of words in a given language.)

Example 2.4.1.

To determine the largest real root of the cubic equation

$$p(z) = a_0 z^3 + a_1 z^2 + a_2 z + a_3 = 0,$$

with real coefficients a_0, a_1, a_2, a_3, is a numerical problem. The input data vector is (a_0, a_1, a_2, a_3). The output data is the desired root x; it is an implicitly defined function of the input data.

[11] The term "algorithm" is a latinization of the name of the Arabic 9th century mathematician Al-Khowârizmî. He also introduced the word algebra (Al-jabr). Western Europe became acquainted with the Hindu positional number system from a latin translation of his book entitled "Algorithmi de numero Indorum".

An algorithm for this problem can be based on Newton's method, supplemented with rules for how the initial approximation should be chosen and how the iteration process is to be terminated. One could also use other iterative methods, or algorithms based upon the formula by Cardano–Tartaglia for the exact solution of the cubic equation (see Problem 2.3.8). Since this uses square roots and cube roots, one needs to assume that algorithms for the computation of these functions have been specified previously.

One often begins the construction of an algorithm for a given problem by breaking down the problem into subproblems in such a way that the output data from one subproblem is the input data to the next subproblem. Thus the distinction between problem and algorithm is not always so clearcut. The essential point is that, in the formulation of the problem, one is only concerned with the initial state and the final state. In an algorithm, however, one should clearly define each step along the way, from start to finish.

We use the term **numerical method** in this book to mean a procedure either to approximate a mathematical problem with a numerical problem or to solve a numerical problem (or at least to transform it to a simpler problem). A numerical method should be more generally applicable than an algorithm, and set lesser emphasize on the completeness of the computational details. The transformation of a differential equation problem to a system of nonlinear equations, as in Example 1.4.1 can be called a numerical method—even without instructions as to how to solve the system of nonlinear equations. Newton's method is a numerical method for determining a root of a large class of nonlinear equations. In order to call it an algorithm conditions for starting and stopping the iteration process should be added.

For a given numerical problem one can consider many differing algorithms. As we have seen in Sec. 2.3 these can, in floating point arithmetic, give approximations of widely varying accuracy to the exact solution.

Example 2.4.2.

The problem of solving the differential equation

$$\frac{d^2y}{dx^2} = x^2 + y^2$$

with boundary conditions $y(0) = 0$, $y(5) = 1$, is not a numerical problem according to the definition stated above. This is because the output data is the *function* y, which cannot in any conspicuous way be specified by a finite number of parameters. The above mathematical problem can be *approximated with a numerical problem* if one specifies the output data to be the values of y for $x = h, 2h, 3h, \ldots, 5 - h$. Also the domain of variation of the unknowns must be restricted in order to show that the problem has a unique solution. This can be done, however, and there are a number of different algorithms for solving the problem approximately, which have different properties with respect to number of arithmetic operations needed and the accuracy obtained.

Before an algorithm can be used it has to be implemented in an algorithmic program language in a reliable and efficient manner. We leave these aspects aside for the moment, but *this is far from a trivial task—it has been said that when the novice thinks the job is done then the expert knows that most of the hard work lies ahead!*

2.4.2 Propagation of Errors

In scientific computing the given input data is usually imprecise. The errors in the input will propagate and give rise to errors in the output. In this section we develop some general tools for studying the propagation of errors. Error-propagation formulas are also of great interest in the *planning and analysis of scientific experiments.*

Note that rounding errors from each step in a calculation are also propagated to give errors in the final result. For many algorithms a rounding error analysis can be given, which shows that the computed result always equals the exact (or slightly perturbed) result of a nearby problem, where the input data has been slightly perturbed (see, e.g, Lemma 2.3.5). The effect of rounding errors on the final result can then be estimated using the tools of this section.

We first consider two simple special cases of error propagation. For a sum of an arbitrary number of terms we get from (2.3.22) by induction:

Lemma 2.4.1.

In addition (and subtraction) a bound for the absolute errors in the result is given by the sum of the bounds for the absolute errors of the operands

$$y = \sum_{i=1}^{n} x_i, \qquad |\Delta y| \le \sum_{i=1}^{n} |\Delta x_i|. \tag{2.4.1}$$

To obtain a corresponding result for the error propagation in multiplication and division, we start with the observations that for $y = \ln(x)$ we have $\Delta(\ln(x)) \approx \Delta(x)/x$. In words: *the relative error in a quantity is approximately equal to the absolute error in its natural logarithm.* This is related to the fact that displacements of the same length at different places on a logarithmic scale, mean the same relative change of the value. From this we obtain the following result:

Lemma 2.4.2.

In multiplication and division, an approximate bound for the relative error is obtained by adding the relative errors of the operands. More generally, for $y = x_1^{m_1} x_2^{m_2} \cdots x_n^{m_n}$,

$$\left|\frac{\Delta y}{y}\right| \lessapprox \sum_{i=1}^{n} |m_i| \left|\frac{\Delta x_i}{x_i}\right|. \tag{2.4.2}$$

Proof. The proof follows by differentiating $\ln y = m_1 \ln x_1 + m_2 \ln x_2 + \cdots + m_n \ln x_n$ and estimating the perturbation in each term. □

We now study the propagation of errors in more general non-linear expressions. Consider first the case when we want to compute a function $y = f(x)$ of a single real variable x. How is the error in x propagated to y? Let $\tilde{x} - x = \Delta x$. Then, a natural way is to approximate $\Delta y = \tilde{y} - y$ with the differential of y (see Fig. 2.4.1). By the mean value theorem,

$$\Delta y = f(x + \Delta x) - f(x) = f'(\xi)\Delta x,$$

where ξ is a number between x and $x + \Delta x$. Suppose that $|\Delta x| \leq \epsilon$. Then it follows that

$$|\Delta y| \leq \max_{\xi} |f'(\xi)|\epsilon, \quad \xi \in [x - \epsilon, x + \epsilon]. \tag{2.4.3}$$

In practice, it is usually sufficient to replace ξ by the available estimate of x. *Even if high precision is needed in the value of $f(x)$, one rarely needs a high relative precision in an error bound or an error estimate.* (In the neighborhood of zeros of the first derivative $f'(x)$ one has to be more careful!)

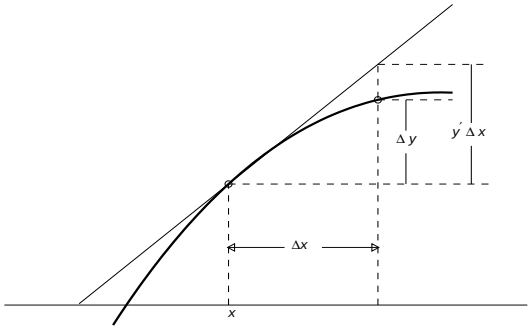


Figure 2.4.1. *Propagated error in function $y = f(x)$.*

By the implicit function theorem a similar result holds if y is an implicit function of x defined by $g(x, y) = 0$. If $g(x, y) = 0$ and $\frac{\partial g}{\partial y}(x, y) \neq 0$, then in a neighborhood of x, y there exists a unique function $y = f(x)$ such that $g(x, f(x)) = 0$ and it holds that

$$f'(x) = \frac{\partial g}{\partial x}(x, f(x)) \Big/ \frac{\partial g}{\partial y}(x, f(x)).$$

Example 2.4.3.

The result in Lemma 2.3.5 does not say that the computed roots of the quadratic equation are close to the exact roots r_1, r_2. To answer that question we must determine how sensitive the roots are to a relative perturbation in the coefficient c. Differentiating $ax^2 + bx + c = 0$, where $x = x(c)$ with respect to c we obtain $(2ax + b)dx/dc + 1 = 0$, $dx/dc = -1/((2ax + b)$. With $x = r_1$ and using $r_1 + r_2 = -b/a$, $r_1 r_2 = c/a$ this can be written

$$\frac{dr_1}{r_1} = -\frac{dc}{c}\frac{r_2}{r_1 - r_2}.$$

This shows that when $|r_1 - r_2| \ll |r_2|$ the roots can be very sensitive to small relative perturbations in c.

When $r_1 = r_2$, i.e. when there is a double root, this linear analysis breaks down. Indeed it is easy to see that the equation $(x - r)^2 + \Delta c = 0$ has roots $x = r \pm \sqrt{\Delta c}$.

To analyze error propagation in a function of several variables we need the following generalization of the mean value theorem:

Theorem 2.4.3.

Assume that the real valued function f is differentiable in a neighborhood of the point $x = (x_1, x_2, \ldots, x_n)$, and let $x = x + \Delta x$ be a point in this neighborhood. Then there exists a number θ, such that

$$\Delta f = f(x + \Delta x) - f(x) = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}(x + \theta \Delta x)\Delta x_i, \quad 0 \le \theta \le 1.$$

Proof. The proof follows by considering the function $F(t) = f(x + t\Delta x)$ and using the mean value theorem for functions of one variable and the chain rule. □

From Theorem 2.4.3 it follows that the perturbation Δf is approximately equal to the total differential. The use of this approximation means that the function $f(x)$ is, in a neighborhood of x that contains the point $x + \Delta x$, approximated by a linear function. All the techniques of differential calculus, such as logarithmic differentiation, implicit differentiation etc. may be useful for the calculation of the total differential; see the examples and the problems at the end of this section.

Theorem 2.4.4. General Formula for Error Propagation*:*

Let the real valued function $f = f(x_1, x_2, \ldots, x_n)$ be differentiable in a neighborhood of the point $x = (x_1, x_2, \ldots, x_n)$ with errors $\Delta x_1, \Delta x_2, \ldots, \Delta x_n$. Then it holds

$$\Delta f \approx \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\Delta x_i. \tag{2.4.4}$$

where the partial derivatives are evaluated at x.

For the maximal error in $f(x_1, x_2, \ldots, x_n)$ we have the approximate bound

$$|\Delta f| \lessapprox \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}\right| |\Delta x_i|. \tag{2.4.5}$$

In order to get a *strict* bound for $|\Delta f|$, one should use in (2.4.5) the maximum absolute values of the partial derivatives in a neighborhood of the known point x. In most practical situations it suffices to calculate $|\partial f/\partial x_i|$ at x and then add a certain marginal amount (5 to 10 percent, say) for safety. Only if the Δx_i are large or if the derivatives have a large relative variation in the neighborhood of x,

need the maximal values be used. (The latter situation occurs, for example, in a neighborhood of an extremal point of $f(x)$.)

The bound in Theorem 2.4.4 is the best possible, unless one knows some dependence between the errors of the terms. Sometimes it can, for various reasons, be a coarse overestimate of the real error, as we have seen in Example 2.3.10.

Example 2.4.4.

Compute error bounds for $f = x_1^2 - x_2$, where $x_1 = 1.03 \pm 0.01$, $x_2 = 0.45 \pm 0.01$. We obtain

$$\left|\frac{\partial f}{\partial x_1}\right| = |2x_1| \leq 2.1, \qquad \left|\frac{\partial f}{\partial x_2}\right| = |-1| = 1,$$

and find $|\Delta f| \leq 2.1 \cdot 0.01 + 1 \cdot 0.01 = 0.031$, or $f = 1.061 - 0.450 \pm 0.032 = 0.611 \pm 0.032$; the error bound has been raised 0.001 because of the rounding in the calculation of x_1^2.

One is seldom asked to give mathematically guaranteed error bounds. More often it is satisfactory to give an estimate of the *order of magnitude* of the anticipated error. The bound for $|\Delta f|$ obtained with Theorem 2.4.3 estimates the maximal error, i.e, covers the worst possible cases, where the sources of error Δx_i contribute with the same sign and magnitudes equal to the error bounds for the individual variables.

In practice, the trouble with formula (2.4.5) is that it often gives bounds which are too coarse. More realistic estimates are often obtained using the standard error introduced in Sec. 2.3.3. Here we give without proof the result for the general case, which can be derived using probability theory and the formula (2.4.4). (Compare with the result for the standard error of a sum given in Sec. 2.3.3.)

Theorem 2.4.5.

Assume that the errors $\Delta x_1, \Delta x_2, \ldots, \Delta x_n$ are independent random variables with mean zero and standard deviations $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$. Then the standard error ϵ for $f(x_1, x_2, \ldots, x_n)$ is given by the formula:

$$\epsilon \approx \left(\sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i} \right)^2 \epsilon_i^2 \right)^{1/2} \tag{2.4.6}$$

Analysis of error propagation is more than just a means for judging the reliability of calculated results. As remarked above, it has an equally important function as a means for *the planning of a calculation or scientific experiment.* For example, it can help in the choice of algorithm, and in making certain decisions during a calculation. Examples of such decisions are the choice of step length during a numerical integration. Increased accuracy often has to be bought at the price of more costly or complicated calculations. One can also shed some light, to what degree it is advisable to obtain a new apparatus to improve the measurements of a given variable, when the measurements of other variables are subject to error as well.

Example 2.4.5.

In Newton's method for solving a nonlinear equation a correction is to be calculated as a quotient $\Delta x = f(x_k)/f'(x_k)$. Close to a root the relative error in the computed value of $f(x_k)$ can be quite large due to cancellation. How accurately should one compute $f'(x_k)$, assuming that the work grows as one demands higher accuracy? Since the limit for the relative error in Δx is equal to the sum of the bounds for the relative errors in $f(x_k)$ and $f'(x_k)$, there is no gain in making the relative error in $f'(x_k)$ very much less than the relative error in $f(x_k)$. This observation is of great importance in particular in the generalization of Newton's method to *systems* of nonlinear equations.

2.4.3 Condition Numbers of Problems

It is useful to have a measure of how sensitive the output data is for variations in the input data. In general, if "small" changes in the input data can result in "large" changes in the output data, we call the problem **ill-conditioned**; otherwise it is called **well-conditioned**. (The definition of large may differ from problem to problem depending on the accuracy of the data and the accuracy needed in the solution.)

We have seen in Sec. 2.4.2 that $|f'(x)|$ can be taken as a measure of the sensitivity of $f(x)$ to a perturbation Δx of x.

Definition 2.4.6.

Assume that $f(x)$ is differentiable at x. Then the **absolute condition number** *for the numerical problem of computing $y = f(x)$ given x is*

$$\kappa_{\text{abs}} = \lim_{|\Delta x| \to 0} \frac{|f(x + \Delta x) - f(x)|}{|\Delta x|} = |f'(x)|. \tag{2.4.7}$$

Usually it is preferable to use condition numbers that are invariant with respect of scaling. Then the ratio of the *relative* perturbations in $f(x)$ and x is the relevant quantity.

Definition 2.4.7.

Assume that $f(x)$ is differentiable at x and that $x \neq 0$ and $f(x) \neq 0$. Then the **relative condition number** κ_{rel} *is*

$$\kappa_{\text{rel}} = \lim_{|\Delta x| \to 0} \frac{|f(x + \Delta x) - f(x)|}{|f(x)|} \Big/ \frac{|\Delta x|}{|x|} = |x| \frac{|f'(x)|}{|f(x)|}. \tag{2.4.8}$$

We say that the problem of computing $f(x)$ given x is ill-conditioned if κ is "large" and well-conditioned otherwise.

It is important to note that the condition number is a property of the numerical problem and does not depend on the algorithm used! An ill-conditioned problem is *intrinsically* difficult to solve accurately using *any* numerical algorithm. Even if the

input data is exact rounding errors made during the calculations in floating point arithmetic may cause large perturbations in the final result. Hence, in some sense an ill-conditioned problem is not well posed.

Example 2.4.6.

If we get an inaccurate solution to an ill-conditioned problem, then often nothing can be done about the situation. (If you ask a stupid questions you get a stupid answer!) However, sometimes the difficulty can depend on the form one has chosen to represent the input and output data of the problem.

The polynomial

$$P(x) = (x-10)^4 + 0.200(x-10)^3 + 0.0500(x-10)^2 - 0.00500(x-10) + 0.00100,$$

is identical with a polynomial Q which if the coefficients are rounded to six digits, becomes

$$\tilde{Q}(x) = x^4 - 39.8000x^3 + 594.050x^2 - 3941.00x + 9805.05.$$

One finds that $P(10.11) = 0.0015 \pm 10^{-4}$, where only three digits are needed in the computation, while $\tilde{Q}(10.11) = -0.0481 \pm \frac{1}{2} \cdot 10^{-4}$, in spite of the fact that eight digits were used in the computation. The rounding to six digits of the coefficients of Q has thus caused an error in the polynomial's value at $x = 10.11$; the erroneous value is more than 30 times larger than the correct value and has the wrong sign. When the coefficients of Q are input data, the problem of computing the value of the polynomial for $x \approx 10$ is far more ill-conditioned than when the coefficients of P are input data.

Consider now a multivariate numerical problem, where the solution is given by the function $y = f(x)$, or in component form

$$y_j = f_j(x_1, \ldots, x_n), \quad j = 1:m.$$

It is usually more convenient to have a single number to measure the conditioning. This can be achieved by using norms, e.g., the special cases ($p = 1, 2$ and ∞) of the family of vector p-norms, (see Sec. 1.6.8)

$$\|x\|_p = (x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}, \qquad 1 \le p < \infty,$$

and the corresponding matrix norms.

Definition 2.4.8.

Consider a problem of computing $y = f(x)$, where the input data is $(x_1, \ldots, x_n)$ and the output data $(y_1, \ldots, y_m)$. The absolute condition number of this problem is

$$\kappa_{\text{abs}} = \lim_{\epsilon \to 0} \sup \frac{1}{\epsilon} \left\{ \|f(\tilde{x}) - f(x)\| \ : \ \|\tilde{x} - x\| \le \epsilon \right\}. \tag{2.4.9}$$

If $x \neq 0$ and $f(x) \neq 0$, then the (normwise) relative condition number is

$$\kappa_{\text{rel}} = \lim_{\epsilon \to 0} \sup \frac{1}{\epsilon} \left\{ \frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|} \ : \ \|\tilde{x} - x\| \le \epsilon \|x\| \right\}. \tag{2.4.10}$$

The (absolute or relative) condition number is a function of the input data x and also depends on the choice of norms in the data space and in the solution space. If the relative condition number of a problem is κ_{rel}, then for sufficiently small ϵ we have the estimate

$$\|\tilde{y} - y\| \leq \kappa\epsilon\|y\| + O(\epsilon^2).$$

It follows that the solution will have roughly $s = \log_{10}\kappa$ less significant decimal digits than the input data, but *this may not hold for all components of the output.*

The conditioning of a problem can to some degree be illustrated geometrically. A numerical problem P means a mapping of the space X of possible input data onto the space Y of the output data. The dimensions of these spaces are usually quite large. In Fig 2.4.2 we picture a mapping in two dimensions. Since we are considering relative changes, we take the coordinate axis to be logarithmically scaled. A small circle of radius r is mapped onto an ellipse whose major axis is κr, where κ is the condition number of the problem P.

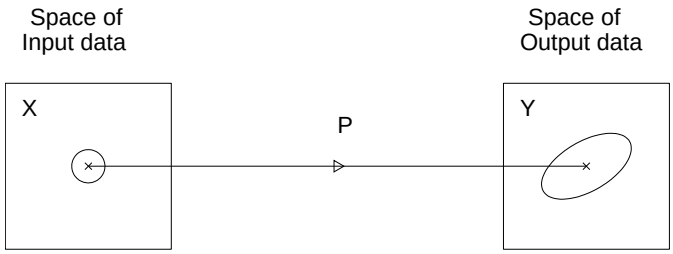


Figure 2.4.2. *Geometrical illustration of the condition number.*

Assume that each function f_j has partial derivatives with respect to all n variables x_i, $i = 1 : n$ and let J be the **Jacobian matrix** with elements

$$J_{ij} = \frac{\partial f_j}{\partial x_i}, \quad j = 1 : m, \quad i = 1 : n. \tag{2.4.11}$$

Conditioning numbers of general differentiable functions have been studied already by Rice [38], who showed that the condition numbers defined above can then be expressed as

$$\kappa_{\text{abs}} = \|J\|, \qquad \kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|}\|J\|. \tag{2.4.12}$$

where the matrix norm is subordinate to the vector norm.

The normwise analysis used above is usually satisfactory provided the problem is "well scaled", i.e., when the error in the components of x have roughly similar magnitude. If this is not the case then a **component-wise perturbation** analysis may give sharper bounds.

2.4.4 Perturbation Analysis for Linear Systems

An important special case is the perturbation analysis for a linear systems. $Ax = b$, where $x, b \in \mathbf{R}^n$. We assume that A is nonsingular and $b \neq 0$ so that the system has a unique solution $x \neq 0$. We shall investigate the sensitivity of x to perturbations δA and δb in A and b.

The perturbed solution $x + \delta x$ satisfies the linear system

$$(A + \delta A)(x + \delta x) = b + \delta b.$$

Subtracting $Ax = b$ we obtain $(A + \delta A)\delta x = \delta b - \delta A x$. Assuming that also the matrix $(A + \delta A) = A(I + A^{-1}\delta A)$ is nonsingular, and solving for δx yields

$$\delta x = (I + A^{-1}\delta A)^{-1}A^{-1}(\delta b - \delta A x), \tag{2.4.13}$$

which is the basic identity for the analysis. Taking norms gives

$$\|\delta x\| \leq \|(I + A^{-1}\delta A)^{-1}\|\,\|A^{-1}\|\,(\|\delta A\|\,\|x\| + \|\delta b\|)\,.$$

It can be shown (see Problem 9) that if $\|A^{-1}\delta A\| < 1$, then $A + \delta A$ is nonsingular and

$$\|(I + A^{-1}\delta A)^{-1}\| < 1/(1 - \|A^{-1}\delta A\|).$$

When $\delta A = 0$ we have $\delta x = A^{-1}\delta b$. It follows that $\|\delta x\| \leq \|A^{-1}\|\,\|\delta b\|$, and hence $\kappa_{\text{abs}} = \|A^{-1}\|$ is the absolute condition number. For the (normwise) relative perturbation, we get the upper bound

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa_{\text{rel}}(A, b)\frac{\|\delta b\|}{\|b\|}, \quad \kappa_{\text{rel}}(A, b) := \frac{\|Ax\|}{\|x\|}\|A^{-1}\|, \tag{2.4.14}$$

This inequality is sharp in the sense that for any matrix norm and for any A and b there exists a perturbation δb such that equality holds.

Consider now the case $\delta b = 0$. From (2.4.13)we obtain, neglecting second order terms,

$$\frac{\|\delta x\|}{\|x\|} \lessapprox \kappa_{rel}(A)\frac{\|\delta A\|}{\|A\|}, \quad \kappa_{rel}(A) = \kappa = \|A\|\,\|A^{-1}\|. \tag{2.4.15}$$

We have

$$\kappa_{\text{rel}}(A, b) = \frac{\|Ax\|}{\|x\|}\|A^{-1}\| \leq \|A\|\,\|A^{-1}\| = \kappa_{\text{rel}}(A).$$

although, for given x (or b), this upper bound may not be achievable for any perturbation δb. However, usually the factor $\kappa = \|A\|\,\|A^{-1}\|$ is used as condition number for both perturbations in A and in b.

For the Euclidian vector and matrix norm ($p = 2$) we define:

Definition 2.4.9.

The **condition number** *for a square nonsingular matrix A is*

$$\kappa_2 = \kappa_2(A) = \|A\|_2\,\|A^{-1}\|_2 = \sigma_1/\sigma_n, \tag{2.4.16}$$

where σ_1 and σ_n are the largest and smallest singular value of A.

Note that $\kappa(\alpha A) = \kappa(A)$, i.e., the condition number is invariant under multiplication of A by a scalar. From the definition and the identity $AA^{-1} = I$ it also follows that $\kappa(AB) \leq \kappa(A)\kappa(B)$ and

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 \geq \|I\| = 1,$$

that is, the condition number κ_2 is always greater or equal to one. Matrices with small condition numbers are said to be **well-conditioned**.

For any real, orthogonal matrix Q we have

$$\kappa_2(Q) = \|Q\|_2\|Q^{-1}\|_2 = 1,$$

so Q is perfectly conditioned. By Lemma 1.6.3 we have $\|QAP\|_2 = \|A\|_2$ for any orthogonal P and Q. It follows that

$$\kappa_2(PAQ) = \kappa_2(A),$$

i.e. *the condition number of a matrix A is invariant under orthogonal transformations.* This important fact is one reason why orthogonal transformations play a central role in numerical linear algebra!

How large may κ be before we consider the problem to be ill-conditioned? That depends on the accuracy of the data and the accuracy desired in the solution. If the data have a relative error of 10^{-7} then we can guarantee a (normwise) relative error in the solution $\leq 10^{-3}$ if $\kappa \leq 0.5 \cdot 10^4$. However, to guarantee a (normwise) relative error in the solution $\leq 10^{-6}$ we need to have $\kappa \leq 5$.

Example 2.4.7.

The **Hilbert matrix** H_n of order n with elements

$$H_n(i,j) = h_{ij} = 1/(i+j-1), \quad 1 \leq i,j \leq n.$$

is a notable example of an ill-conditioned matrix. In Table 2.4.1 approximate condition numbers of Hilbert matrices of order ≤ 12, computed in IEEE double precision, are given. For $n > 12$ the Hilbert matrices are too ill-conditioned even for IEEE double precision! From a result by G. Szegö (see Gautschi [24, p. 34]) it follows that

$$\kappa_2(H_n) \approx \frac{(\sqrt{2}+1)^{4(n+1)}}{2^{15/4}\sqrt{\pi n}} \sim e^{3.5n},$$

that is, the condition numbers grows exponentially with n. Although the severe ill-conditioning exhibited by the Hilbert matrices is rare, moderately ill-conditioned linear systems do occur regularly in many practical applications!

The normwise analysis in the previous section usually is satisfactory when the linear system is "well scaled". If this is not the case then a **component-wise perturbation** analysis may give sharper bounds.

Table 2.4.1. *Condition numbers of Hilbert matrices of order* ≤ 12.

n	$\kappa_2(H_n)$	n	$\kappa_2(H_n)$
1	1	7	$4.753{\cdot}10^{8}$
2	19.281	8	$1.526{\cdot}10^{10}$
3	$5.241{\cdot}10^{2}$	9	$4.932{\cdot}10^{11}$
4	$1.551{\cdot}10^{4}$	10	$1.602{\cdot}10^{13}$
5	$4.766{\cdot}10^{5}$	11	$5.220{\cdot}10^{14}$
6	$1.495{\cdot}10^{7}$	12	$1.678{\cdot}10^{16}$

We first introduce some notations. The absolute values $|A|$ and $|b|$ of a matrix A and vector b is interpreted componentwise,

$$|A|_{ij} = (|a_{ij}|), \qquad |b|_i = (|b_i|).$$

The **partial ordering** "$\leq$" for the absolute values of matrices $|A|$, $|B|$ and vectors $|b|$, $|c|$, is to be interpreted component-wise[12]

$$|A| \leq |B| \iff |a_{ij}| \leq |b_{ij}|, \qquad |b| \leq |c| \iff |b_i| \leq |c_i|.$$

It follows easily that $|AB| \leq |A|\,|B|$ and a similar rule holds for matrix-vector multiplication.

Assume now that we have component-wise bounds for the perturbations in A and b,

$$|\delta A| \leq \omega |E|, \qquad |\delta b| \leq \omega |f|, \tag{2.4.17}$$

where E and f are known. Taking absolute values in (2.4.13) gives component-wise error bounds for the corresponding perturbations in x,

$$|\delta x| \leq |(I + A^{-1}\delta A)^{-1}|\,|A^{-1}|(|\delta A||x| + |\delta b|)$$

The matrix $(I - |A^{-1}||\delta A|)$ is guaranteed to be nonsingular if $\| \,|A^{-1}|\,|\delta A|\,\| < 1$. Neglecting second order terms in ω and using (2.4.17) gives

$$|\delta x| \lessapprox |A^{-1}|(|\delta A||x| + |\delta b|) \leq \omega |A^{-1}|(|E|\,|x| + |f|), \tag{2.4.18}$$

If we set $E = |A|$ and $f = |b|$, then taking norms in (2.4.18) we get

$$\|\delta x\| \lessapprox \omega \| \,|A^{-1}|(|A|\,|x| + |b|)\,\| + O(\omega^2). \tag{2.4.19}$$

2.4.5 Forward and Backward Error Analysis

Consider a finite algorithm with input data $(a_1, \ldots, a_r)$, in which by a sequence of arithmetic operations the output data $(w_1, \ldots, w_s)$ is computed. There are two basic forms of roundoff error analysis for such an algorithm, which both are useful:

[12]Note that $A \leq B$ in other contexts means that $B - A$ is positive semidefinite.

(i) In **forward error analysis** one attempts to find bounds for the errors in the solution $|\overline{w}_i - w_i|$, $i = 1 : s$, where $\overline{w}_i$ denotes the computed value of w_i.

(ii) In **backward error analysis**, pioneered by J. H. Wilkinson in the late fifties, one attempts to determine a modified set of data $\tilde{a}_i$ such that the computed solution $\overline{w}_i$ is the *exact solution*, and give bounds for $|\tilde{a}_i - a_i|$. There may be an infinite number of such sets; sometimes there is just one and it can happen, even for very simple algorithms, that no such set exists.

By means of backward error analysis it has been shown, even for many quite complicated algorithms, that the computed results the algorithm produces under the influence of roundoff error are the *exact* output data of a problem of the same type in which the relative change data only is of the order of the unit roundoff u.

Sometimes, when a pure backward error analysis is difficult to achieve, one can show that the computed solution is a slightly perturbed solution to a problem with slightly modified input data. An example of such a **mixed error analysis** is the error analysis given in Lemma 2.3.5 for the solution of a quadratic equation.

In backward error analysis no reference is made to the exact solution for the original data. In practice, when the data is known only to a certain accuracy, the "exact" solution may not be well-defined. Then any solution, whose backward error is smaller than the domain of uncertainty of the data, may be considered to a satisfactory result.

To yield error bounds for $\overline{w}_i$, a backward error analysis has to be complemented with a perturbation analysis. For this the error propagation formulas in Sec. 2.4.2 can often be used. A great advantage of backward error analysis is that when it applies, it tends to give much sharper results than a forward error analysis. Perhaps more important, it usually also gives a better insight into the stability (or lack of it) of the algorithm. It should be stressed that *the primary purpose of a rounding error analysis is to give insight in the properties of the algorithm.*

2.4.6 Stability of Algorithms

One common reason for poor accuracy in the computed solution is that the problem is ill-conditioned. But poor accuracy can also be caused by a poorly constructed algorithm. We say in general that an algorithm is **unstable** if it can introduce large errors in the computed solutions to a well-conditioned problem.

There are different definitions of stability of algorithms for different classes of numerical problems. The treatment here is geared towards stationary problems and may not be very useful for time dependent problems in ordinary and partial differential equations. We defer the treatment of suitable definitions of stability for these classes of problems until Volume 3; see also Sec. 3.??.

Example 2.4.8.

For $\epsilon = 10^{-6}$ the system

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

is well-conditioned and has the exact solution $x_1 = -x_2 = -1/(1-\epsilon) \approx -1$. If Gaussian elimination is used, multiplying the first equation by 10^6 and subtracting from the second, we obtain $(1-10^6)x_2 = -10^6$. Rounding this to $x_2 = 1$ is correct to six digits. In the back-substitution to obtain x_1, we then get $10^{-6}x_1 = 1-1$, or $x_1 = 0$, which is a completely wrong result. This shows that Gaussian elimination can be an unstable algorithm. To ensure stability it is necessary to perform row (and/or column) interchanges *not only when a pivotal element is exactly zero, but also when it is small.*

Definition 2.4.10.

An algorithm is **backward stable** *if the computed solution $\overline{w}$ for the data a is the exact solution of a problem with slightly perturbed data $\bar{a}$ such that for some norm $\|\cdot\|$ it holds*

$$\|\bar{a} - a\|/\|a\| < c_1 u, \tag{2.4.20}$$

where c_1 is a not too large constant and u is the unit roundoff.

We are usually satisfied if we can prove normwise forward or backward stability for some norm, e.g., $\|\cdot\|_2$ or $\|\cdot\|_\infty$. Occasionally we may like the estimates to hold element-wise, e.g.

$$|\bar{a}_i - a_i|/|a_i| < c_2 u, \qquad i = 1:r. \tag{2.4.21}$$

For example, by equation (2.3.15) the usual algorithm for computing an inner product $x^T y$ is backward stable, for element-wise relative perturbations.

We would like stability to hold for some *class of input data*. For example, a numerical algorithm for solving systems of linear equations $Ax = b$ is backward stable for a class of matrices $\mathcal{A}$ if for each $A \in \mathcal{A}$ and for each b the computed solution $\bar{x}$ satisfies $\bar{A}\bar{x} = \bar{b}$ where $\bar{A}$ and $\bar{b}$ are close to A and b.

A backward stable algorithm will not necessarily compute an accurate solution. However, if the condition number of the problem is κ, then it follows that

$$\|\overline{w} - w\| \leq c_1 u \kappa \|w\| + O(u^2). \tag{2.4.22}$$

Hence the error in the solution may still be large if the problem is ill-conditioned. However, we have obtained an answer which is the exact mathematical solution to a problem with data close to the one we wanted to solve. If the perturbations $\bar{a} - a$ are within the uncertainties of the given data, *the computed solution is as good as our data warrants*!

An important property of backward stable algorithms for the solution of linear systems is given in the following theorem.

Theorem 2.4.11.

An algorithm for solving $Ax = b$ is backward stable according to Definition 2.4.10 if and only if the computed solution $\bar{x}$ has a small residual, that is,

$$\|b - A\bar{x}\| \leq c_3 u \|A\|\|\bar{x}\|. \tag{2.4.23}$$

Proof. Suppose that (2.4.23) holds. If we define for the 2-norm

$$\delta A = r\bar{x}^T/\|\bar{x}\|_2^2, \qquad r = b - A\bar{x},$$

then it holds exactly that $(A + \delta A)\bar{x} = A\bar{x} + r = b$, where

$$\|\delta A\|_2 \le \|r\|_2/\|\bar{x}\|_2 \le c_3 u\|A\|_2.$$

We can take $\delta b = 0$ and hence the algorithm is backward stable by Definition 2.4.10.

Conversely, if the algorithm is backward stable then, $\bar{A}\bar{x} = \bar{b}$, where

$$\|\bar{A} - A\| \le c_2 u\|A\|, \qquad \|\bar{b} - b\| \le c_2 u\|b\|.$$

Since $b - A\bar{x} = (\bar{A} - A)\bar{x} + b - \bar{b}$ it follows that an estimate of the form (2.4.23) holds for the norm of the residual. □

Many important algorithms for solving linear systems, for example, most iterative methods, are not backward stable. The following weaker definition of stability is also useful.

Definition 2.4.12.

An algorithm is **stable** *if the computed solution $\overline{w}$ satisfies* (2.4.22)*, where c_1 is a not too large constant, u is the unit roundoff, and κ is the condition number of the problem.*

By the definition of the condition number κ it follows that backward stability implies forward stability, but *the converse is not true.*

Sometimes it is necessary to weaken the definition of stability. Often an algorithm can be considered stable if *it produces accurate solutions for well-conditioned problems.* Such an algorithm can be called **weakly stable**. Weak stability may be sufficient for giving confidence in an algorithm.

Example 2.4.9.

In the method of normal equations for computing the solution of a linear least squares problem one first forms the matrix A^TA. This product matrix can be expressed in outer form as

$$A^TA = \sum_{i=1}^{m} a_i a_i^T,$$

where a_i^T is the ith row of A, i.e. $A^T = (\, a_1 \quad a_2 \quad \dots \quad a_m \,)$. From (2.3.13) it follows that this computation is not backward stable, i.e. it is not true that $fl(A^TA) = (A+E)^T(A+E)$ for some small error matrix E. In order to avoid loss of significant information higher precision need to be used.

Backward stability is easier to prove when there is a sufficiently large set of input data compared to the number of output data. In Example 2.4.9 that computing the outer product xY^T, where we have $2n$ data and n^2 results is not a backward stable operations. Also it is harder (sometimes impossible) to show backward stability when the input data is structured rather than general.

Example 2.4.10.

It can be shown that many algorithms for solving a linear system $Ax = b$ are backward stable, i.e. the computed solution is the exact solution of a system $(A + E)x = b$, where $\|E\|$ is not much larger than the machine precision. In many cases the system matrix is **structured**. An important example is **Toeplitz matrices** T. A Toeplitz matrix has entries that are constant along every diagonal

$$T = (t_{i-j})_{1\le i,j\le n} = \begin{pmatrix} t_0 & t_1 & \dots & t_{n-1} \\ t_{-1} & t_0 & \dots & t_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{-n+1} & t_{-n+2} & \dots & t_0 \end{pmatrix} \in \mathbf{R}^{n\times n}$$

and is defined by the vector of $2n-1$ quantities $t = (t_{-n+1}, \dots, t_0, \dots, t_{n-1})$.

Ideally, in a strict backward error analysis, we would like to show that a solution algorithm always computes *an exact solution to a nearby Toeplitz system* defined by $t + s$, where s is small. It has been shown that no such algorithm can exist! We have to be content with algorithms that (at best) compute the exact solution of $(T + E)x = b$, where $\|E\|$ is small but E unstructured.

In the construction of an algorithm for a given problem, one often breaks down the problem into a chain of subproblems, $P_1, P_2, \dots, P_k$ for which algorithms $A_1, A_2, \dots, A_k$ are known, in such a way that the output data from P_{i-1} is the input data to P_i. Different ways of decomposing the problem give numerically different algorithms. It is dangerous if the last subproblem in such a chain is ill-conditioned.

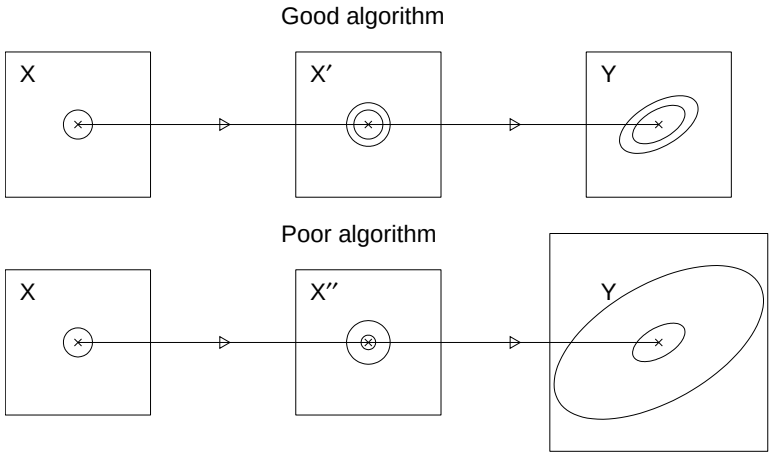


Figure 2.4.3. *Two examples of a decomposition of a problem P into two subproblems.*

In Fig. 2.4.3 we see two examples of a decomposition of the problem P into two subproblems. From X to X'' there is a strong contraction which is followed

by an expansion about equally strong in the mapping from X'' to Y. The roundoff errors which are made in X'' when the intermediate results are stored have as a consequence that one arrives somewhere in the surrounding circle, which is then transformed into a very large region in Y. *The important conclusion is that even if the algorithms for the subproblems are stable we cannot conclude that the composed algorithm is stable!*

Example 2.4.11.

The problem of computing the eigenvalues λ_i of a symmetric matrix A, given its elements (a_{ij}), is always a well-conditioned numerical problem with condition number equal to 1. Consider an algorithm which breaks down this problem into two subproblems:

P_1: compute the coefficients of the characteristic polynomial of the matrix A $p(\lambda) = \det(A - \lambda I)$ of the matrix A.

P_2: compute the roots of the polynomial $p(\lambda)$ obtained from P_1.

It is well known that the second subproblem P_2 can be very ill-conditioned. For example, for a symmetric matrix A with eigenvalues $\pm 1, \pm 2, \ldots, \pm 20$ the condition number for P_2 is 10^{14} in spite of the fact that the origin lies exactly between the largest and smallest eigenvalues, so that one cannot blame the high condition number on a difficulty of the same type as that encountered in Example 2.4.7.

The important conclusion that eigenvalues should not be computed as outlined above is further discussed in Sec. 6.4.1.

On the other hand, as the next example show, it need not be dangerous if the first subproblem of a decomposition is ill-conditioned, even if the problem itself is well-conditioned.

Example 2.4.12.

The first step in many algorithms for computing the eigenvalues λ_i of a symmetric matrix A is to use orthogonal similarity transformations to symmetric tridiagonal form,

$$Q^T A Q = T = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}.$$

In the second step the eigenvalues of T, which coincide with those of A, are computed.

Wilkinson [46, §5.28] showed that the computed tridiagonal matrix can differ a lot from the matrix corresponding to exact computation. Hence here the first subproblem is ill-conditioned. (This fact is not as well known as it should be and still alarms many users!) However, the second subproblem is well-conditioned and the combined algorithm is known to be backward stable, i.e. the computed eigenvalues are the exact eigenvalues of a matrix $A + E$, where $\|E\|_2 < c(n)u\|A\|_2$. This is a more complex example of a calculation, where rounding errors cancel!

In Sec. 1.4 some methods for the numerical solution of differential equations were illustrated. It should be realized that there are possibilities for catastrophic growth of errors in such processes. The notion of stability for such methods is related to the stability of linear difference equations and will be treated in Sec. 3.2 and at length in Vol. III.

Review Questions

1. The maximal error bounds for addition and subtraction can for various reasons be a coarse overestimate of the real error. Give, preferably with examples, two such reasons.

2. How is the condition number $\kappa(A)$ of a matrix A defined? How does $\kappa(A)$ relate to perturbations in the solution x to a linear system $Ax = b$, when A and b are perturbed?

3. Define the condition number of a numerical problem P of computing output data $y_1, \ldots, y_m$ given input data $x_1, \ldots, x_n$.

4. Give examples of well-conditioned and ill-conditioned problems.

5. What is meant by (a) a forward error analysis; (b) a backward error analysis; (c) a mixed error analysis?

6. What is meant by (a) a backward stable algorithm; (b) a forward stable algorithm; (c) a mixed stable algorithm; (d) a weakly stable algorithm?

Problems

1. (a) Determine the maximum error for $y = x_1 x_2^2/\sqrt{x_3}$, where $x_1 = 2.0 \pm 0.1$, $x_2 = 3.0 \pm 0.2$, and $x_3 = 1.0 \pm 0.1$. Which variable contributes most to the error?

(b) Compute the standard error using the same data as in (a), assuming that the error estimates for the x_i indicate standard deviations.

2. One wishes to compute $f = (\sqrt{2} - 1)^6$, using the approximate value 1.4 for $\sqrt{2}$. Which of the following mathematically equivalent expressions gives the best result

$$\frac{1}{(\sqrt{2}+1)^6}; \quad (3-2\sqrt{2})^3; \quad \frac{1}{(3+2\sqrt{2})^3}; \quad 99-70\sqrt{2}; \quad \frac{1}{99+70\sqrt{2}}?$$

3. Analyze the error propagation in x^α:

(a) If x is exact and α in error. (b) If α is exact and x in error.

4. One is observing a satellite in order to determine its speed. At the first observation, $R = 30,000 \pm 10$ miles. Five seconds later, the distance has increased by $r = 125.0 \pm 0.5$ miles and the change in the angle was $\phi = 0.00750 \pm 0.00002$ radians. What is the speed of the satellite, assuming that it moves in a straight line and with constant speed in the interval?

5. One has measured two sides and the included angle of a triangle to be $a = 100.0 \pm 0.1$, $b = 101.0 \pm 0.1$, and the angle $C = 1.00^o \pm 0.01^o$. Then the third side is given by the cosine theorem

$$c = (a^2 + b^2 - 2ab\cos C)^{1/2}.$$

(a) How accurately is it possible to determine c from the given data?

(b) How accurately does one get c if one uses the value $\cos 1^o = 0.9998$, which is correct to four decimal places.

(c) Rewrite the cosine theorem so that it is possible to compute c to full accuracy using only a four-decimal table for the trigonometric functions.

6. Consider the linear system

$$\begin{pmatrix} 1 & \alpha \\ \alpha & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

where $\alpha \neq 1$. What is the relative condition number for computing x and show that the problem of computing x. Using Gaussian elimination and four decimal digits compute x and y for $\alpha = 0.9950$ and compare with the exact solution $x = 1/(1-\alpha^2)$, $y = -\alpha/(1-\alpha^2)$.

7. (a) Let two vectors u and v be given with components (u_1, u_2) and (v_1, v_2). The angle ϕ between u and v is given by the formula

$$\cos\phi = \frac{u_1v_1 + u_2v_2}{(u_1^2 + u_2^2)^{1/2}(v_1^2 + v_2^2)^{1/2}}.$$

Show that computing the angle ϕ from the components of u and v is a well-conditioned problem.

Hint: Take the partial derivative of $\cos\phi$ with respect to u_1, and from this compute $\partial\phi/\partial u_1$. The other partial derivatives are obtained by symmetry.

(b) Show that the formula in (a) is *not* stable for small angles ϕ.

(c) Show that the following algorithm is stable. First normalize the vectors $\tilde{u} = u/\|u\|_2$, $\tilde{v} = v/\|v\|_2$. Then compute $\alpha = \|\tilde{u} - \tilde{v}\|_2$, $\beta = \|\tilde{u} + \tilde{v}\|_2$ and set

$$\phi = \begin{cases} 2\arctan(\alpha/\beta), & \text{if } \alpha \leq \beta; \\ \pi - 2\arctan(\beta/\alpha), & \text{if } \alpha > \beta. \end{cases}$$

8. One has an algorithm for computing the integral

$$I(a,b) = \int_0^1 \frac{e^{-bx}}{a + x^2} dx.$$

The physical quantities a and b have been measured to be $a = 0.4000 \pm 0.003$, $b = 0.340 \pm 0.005$. Using the algorithms for various values of a and b one

performs experimental perturbations and obtains:

a	b	I
0.39	0.34	1.425032
0.40	0.32	1.408845
0.40	0.34	1.398464
0.40	0.36	1.388198
0.41	0.34	1.372950

How large is the uncertainty in $I(a, b)$?

9. Let $B \in \mathbf{R}^{n\times n}$ be a matrix for which $\|B\| < 1$. Show that the infinite sum and product

$$(I-B)^{-1} = \begin{cases} I + B + B^2 + B^3 + B^4 \cdots, \\ (I+B)(I+B^2)(I+B^4)(I+B^8)\cdots \end{cases}$$

both converge to the indicated limit.

Hint: Use the identity $(I - B)(I + B + \cdots + B^k) = I - B^{k+1}$.

(b) Show that the matrix $(I - B)$ is nonsingular and that

$$\|(I-B)^{-1}\| \le 1/(1-\|B\|).$$

10. Solve the linear system in Example 2.4.8 with Gaussian elimination after exchanging the two equations. Do you now get an accurate result?

11. Derive forward and backward recursion formulas for calculating the integrals

$$I_n = \int_0^1 \frac{x^n}{4x+1}\,dx.$$

Why is one algorithm stable and the other unstable?

Problems and Computer Exercises

1. (a) Use the results in Table 2.4.1 to determine constants c and q such that $\kappa(H_n) \approx c \cdot 10^q$.

(b) Compute the Bauer–Skeel condition number $\text{cond}\,(H_n) = \| \, |H_n^{-1}||H_n| \, \|_2$, of the Hilbert matrices for $n = 1 : 12$. Compare the result with the values of $\kappa(H_n)$ given in Table 2.4.1.

2. Vandermonde matrices are structured matrices of the form

$$V_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \cdots & \alpha_n^{n-1} \end{pmatrix}.$$

Let $\alpha_j = 1 - 2(j-1)/(n-1)$, $j = 1 : n$. Compute the condition numbers $\kappa_2(V_n)$ for $n = 5, 10, 15, 20, 25$. Is the growth in $\kappa_2(V_n)$ exponential in n?

2.5 Automatic Control of Accuracy and Verified Computing

2.5.1 Running Error Analysis

A different approach to rounding error analysis is to perform the analysis automatically, for *each particular computation*. This gives an *a posteriori* error analysis as compared to the *a priori* error analysis discussed above.

A simple form of a posteriori analysis, called running error analysis, was used in the early days of computing, see Wilkinson [47]. To illustrate his idea we rewrite the basic model for floating point arithmetic as

$$x \operatorname{op} y = fl\,(x \operatorname{op} y)(1+\epsilon).$$

These are also satisfied for most implementations of floating point arithmetic. Then, the actual error can be estimated $|fl\,(x \operatorname{op} y) - x \operatorname{op} y| \leq u|fl\,(x \operatorname{op} y)|$. Note that the error is now given in terms of the *computed* result and is available in the computer at the time the operation is performed. This running error analysis can often be easily implemented. We just take an existing program and modify it, so that as each arithmetic operation is performed, the absolute value of the computed results is added into the accumulating error bound.

Example 2.5.1.

The inner product $fl\,(x^T y)$ is computed by the program

$$\begin{aligned}
&s = 0; \quad \eta = 0;\\
&\textbf{for } i = 1, 2, \ldots, n\\
&\qquad t = fl\,(x_i y_i); \quad \eta = \eta + |t|;\\
&\qquad s = fl\,(s + t); \quad \eta = \eta + |s|;\\
&\textbf{end}
\end{aligned}$$

For the final error we have the estimate $|fl\,(x^T y) - x^T y| \leq \eta u$. Note that a running error analysis takes advantage of cancellations in the sum. This is in contrast to the previous estimates, which we call a priori error analysis, where the error estimate is the same for all distribution of signs of the elements x_i and y_i.

Efforts have been made to design the computational unit of a computer so that it gives, in every arithmetic operation, only those digits of the result which are judged to be significant (possibly with a fixed number of extra digits), so-called *unnormalized floating arithmetic*. This method reveals poor construction in algorithms, but in many other cases it gives a significant and unnecessary loss of accuracy. The mechanization of the rules, which a knowledgeable and experienced person would use for control of accuracy in hand calculation, is not as free from problems as one might expect. As complement to arithmetical operations of conventional type, the above type of arithmetic is of some interest, but it is doubtful that it will ever be widely used.

A fundamental difficulty in automatic control of accuracy is that to decide how many digits is needed in a quantity to be used in later computation, one needs to consider the *entire context of the computations*. It can in fact occur that the errors in many operands depend on each other in such a way that they cancel each other. Such **cancellation of error**, is a completely different phenomenon from the previously discussed cancellation of terms, is most common in larger problems, but will be illustrated here with a simple example.

Example 2.5.2.

Suppose we want to compute $y = z_1 + z_2$, where $z_1 = \sqrt{x^2+1}$, $z_2 = 200 - x$, $x = 100 \pm 1$, with a rounding error which is negligible compared to that resulting from the errors in z_1 and z_2. The best possible error bounds in the intermediate results are $z_1 = 100 \pm 1$, $z_2 = 100 \pm 1$. It is then tempting to be satisfied with the result $y = 200 \pm 2$.

However, the errors in z_1 and z_2 due to the uncertainty in x will, to a large extent, cancel each other! This becomes clear if we rewrite the expression as

$$y = 200 + (\sqrt{x^2+1} - x) = 200 + \frac{1}{\sqrt{x^2+1} + x}.$$

It follows that $y = 200 + u$, where $1/202 \lessapprox u \leq 1/198$. Thus y can be computed with an absolute error less than about $2/(200)^2 = 0.5 \cdot 10^{-4}$. Therefore using the expression $y = z_1 + z_2$ the intermediate results z_1 and z_2 should be computed to four decimals even though the last integer in these is uncertain! The result is $y = 200.0050 \pm \frac{1}{2}10^{-4}$.

In larger problems, such a cancellation of errors can occur even though one cannot easily give a way to rewrite the expressions involved. The authors have seen examples where the final result, a sum of seven terms, was obtained correctly to eight decimals even though the terms, which were complicated functions of the solution to a system of nonlinear equations with fourteen unknowns, were correct only to three decimals! Another nontrivial example is given in Example 2.4.12.

2.5.2 Experimental Perturbations

In many practical problems, the functional dependence between input data and output data are so complicated that it is difficult to directly apply the general formulas for error propagation derived in Sec. 2.4.4. One can then investigate the sensitivity of the output data for perturbations in the input data by means of an **experimental perturbational calculation**: one performs the calculations many times with perturbed input data and studies the perturbations in the output data.

Important data, such as the step length in a numerical integration or the parameter which determines when an iterative process is going to be broken off, should be varied with all the other data left unchanged. If one can easily *vary the precision of the machine* in the arithmetic operations one can get an idea of the influence of rounding errors. It is generally not necessary to make a perturbational

calculation for each and every data component; one can instead *perturb many input data simultaneously*–for example, by using random numbers.

A perturbational calculation often gives not only an error estimate, but also greater insight into the problem. Occasionally, it can be difficult to interpret the perturbational data correctly, since the disturbances in the output data depend not only on the mathematical problem, but also on the choice of numerical method and the details in the design of the algorithm. The rounding errors during the computation are not the same for the perturbed and unperturbed problem. Thus if the output data reacts more sensitively than one had anticipated, it can be difficult to immediately point out the source of the error. It can then be profitable to plan a series of perturbation experiments with the help of which one can separate the effects of the various sources of error. If the dominant source of error is the method or the algorithm, then one should try another method or another algorithm.

It is beyond the scope of this book to give further comments on the planning of such experiments; imagination and the general insights regarding error analysis which this chapter is meant to give play a large role. Even in the special literature, the discussion of planning of such experiments is surprisingly meager. An exception is the collection of software tools called PRECISE, developed by Chaitin-Chatelin et al., see [11, 12]. These are designed to help the user set up computer experiments to explore the impact of the quality of convergence of numerical methods. It involves a statistical analysis of the effect on a computed solution of random perturbations in data

2.5.3 Introduction to Interval Arithmetic

In **interval arithmetic** one assumes that all input values are given as intervals and systematically calculates an inclusion interval for each intermediate result. It is partly an automatization of calculation with maximal error bounds. The importance of interval arithmetic is that it provides a tool for computing validated answers to mathematical problems. This has played an important role in mathematics. A noteworthy example is the verification of the existence of a Lorenz attractor by W. Tucker [44].

The most frequently used representations for the intervals are the **infimum-supremum** representation

$$I = [a, b] := \{x \mid a \le x \le b\}, \quad (a \le b). \tag{2.5.1}$$

The absolute value or the **magnitude** of an interval is defined as

$$|\,[a, b]\,| = \mathrm{mag}([a, b]) = \max\{|x| \mid x \in [a, b]\}, \tag{2.5.2}$$

and the **mignitude** of an interval is defined as

$$\mathrm{mig}([a, b]) = \min\{|x| \mid x \in [a, b]\}. \tag{2.5.3}$$

In terms of the endpoints we have

$$\mathrm{mag}([a, b]) = \max\{|a|, |b|\},$$
$$\mathrm{mig}([a, b]) = \begin{cases} \min\{|a|, |b|\}, & \text{if } 0 \notin [a, b], \\ 0, & \text{otherwise} \end{cases}.$$

The result of an interval operation equals the range of the corresponding real operation. For example, the difference between the intervals $[a_1, a_2]$ and $[b_1, b_2]$, is defined as the shortest interval which contains all the numbers $x_1 - x_2$, where $x_1 \in [a_1, a_2]$, $x_2 \in [b_1, b_2]$, i.e. $[a_1, a_2] - [b_1, b_2] := [a_1 - b_2, a_2 - b_1]$. Other elementary interval arithmetic operations are similarly defined:

$$[a_1, a_2] \,\mathrm{op}\, [b_1, b_2] := \{x_1 \,\mathrm{op}\, x_2 \mid x_1 \in [a_1, a_2],\ x_2 \in [b_1, b_2]\}, \tag{2.5.4}$$

where $\mathrm{op} \in \{+, -, \cdot, \mathrm{div}\}$. The interval value of a function ϕ (e.g., the elementary functions $\sin, \cos, \exp, \ln$) evaluated on an interval is defined as

$$\phi([a, b]) = [\inf_{x \in [a,b]} \phi(x),\ \sup_{x \in [a,b]} \phi(x)].$$

Operational Definitions

Although (2.5.4) characterizes interval arithmetic operations we also need **operational definitions**. We take

$$\begin{aligned}
&[a_1, a_2] + [b_1, b_2] = [a_1 + b_1, a_2 + b_2],\\
&[a_1, a_2] - [b_1, b_2] = [a_1 - b_2, a_2 - b_1],\\
&[a_1, a_2] \cdot [b_1, b_2] = \big[\min\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}, \max\{a_1b_1, a_1b_2, a_2b_1, a_2b_2\}\big],\\
&1/[a_1, a_2] = [1/a_2, 1/a_1], \quad \text{for} \quad a_1a_2 > 0,\\
&[a_1, a_2]/[b_1, b_2] = [a_1, a_2] \cdot (1/[b_1, b_2]).
\end{aligned} \tag{2.5.5}$$

It is easy to prove that in exact interval arithmetic the operational definitions above give the exact range (2.5.4) of the interval operations. Division by an interval containing zero is not defined and may cause an interval computation to come to a premature end.

A degenerate interval with radius zero is called a point interval and can be identified with a single number $a \equiv [a, a]$. In this way the usual arithmetic is recovered as a special case. The intervals $0 = [0, 0]$ and $1 = [1, 1]$ are the neutral elements with respect to interval addition and interval multiplication, respectively. A non-degenerate interval has no inverse with respect to addition or multiplication For example, we have

$$[1, 2] - [1, 2] = [-1, 1], \qquad [1, 2]/[1, 2] = [0.5, 2].$$

For interval operations the **commutative law**

$$[a_1, a_2] \,\mathrm{op}\, [b_1, b_2] = [b_1, b_2] \,\mathrm{op}\, [a_1, a_2]$$

holds. However, the distributive law has to be replaced by so called **subdistributivity**

$$[a_1, a_2]([b_1, b_2] + [c_1, c_2]) \subseteq [a_1, a_2][b_1, b_2] + [a_1, a_2][c_1, c_2]. \tag{2.5.6}$$

This unfortunately means that expressions, which are equivalent in real arithmetic, differ in exact interval arithmetic. The simple example

$$[-1, 1]([1, 1] + [-1, -1]) = 0 \subset [-1, 1][1, 1] + [-1, 1][-1, -1] = [-2, 2]$$

shows that $-[-1,1]$ is not the additive inverse to $[-1,1]$ and also illustrates (2.5.6).

The operations introduced are **inclusion monotonic**, i.e,

$$[a_1,a_2] \subseteq [c_1,c_2],\ [b_1,b_2] \subseteq [d_1,d_2] \quad \Rightarrow \quad [a_1,a_2]\,\mathrm{op}\,[b_1,b_2] \subseteq [c_1,c_2]\,\mathrm{op}\,[d_1,d_2]. \tag{2.5.7}$$

An alternative representation for an interval is the **midpoint-radius** representation, for which we use brackets

$$\langle c,r\rangle := \{x \,|\, |x-c| \le r\} \quad (0 \le r), \tag{2.5.8}$$

where the midpoint and radius of the interval $[a,b]$ are defined as

$$\mathrm{mid}\,([a,b]) = \frac{1}{2}(a+b), \qquad \mathrm{rad}\,([a,b]) = \tfrac{1}{2}(b-a). \tag{2.5.9}$$

For intervals in the midpoint-radius representation we take as operational definitions

$$\begin{aligned}
&\langle c_1,r_1\rangle + \langle c_2,r_2\rangle = \langle c_1+c_2, r_1+r_2\rangle,\\
&\langle c_1,r_1\rangle - \langle c_2,r_2\rangle = \langle c_1-c_2, r_1+r_2\rangle,\\
&\langle c_1,r_1\rangle \cdot \langle c_2,r_2\rangle = \langle c_1c_2, |c_1|r_2 + r_1|c_2| + r_1r_2\rangle,\\
&1/\langle c,r\rangle = \langle \bar{c}/(|c|^2-r^2), r/(|c|^2-r^2)\rangle, \quad (|c|>r),\\
&\langle c_1,r_1\rangle/\langle c_2,r_2\rangle = \langle c_1,r_1\rangle \cdot (1/\langle c_2,r_2\rangle),
\end{aligned} \tag{2.5.10}$$

where $\bar{c}$ is the complex conjugate of c. For addition, subtraction and inversion, these give the exact range, but for multiplication and division they overestimate the range (see Problem 2). For multiplication we have for any $x_1 \in \langle c_1,r_1\rangle$ and $x_2 \in \langle c_2,r_2\rangle$

$$\begin{aligned}
|x_1x_2 - c_1c_2| &= |c_1(x_2-c_2) + c_2(x_1-c_1) + (x_1-c_1)(x_2-c_2)|\\
&\le |c_1|r_2 + |c_2|r_1 + r_1r_2.
\end{aligned}$$

In implementing interval arithmetic using floating point arithmetic the operational interval results may not be exactly representable as floating point numbers. Then if the lower bound is rounded down to the nearest smaller machine number and the upper bound rounded up, the exact result must be contained in the resulting interval. Recall that these rounding modes (rounding to $-\infty$ and $+\infty$) are supported by the IEEE 754 standard. For example, using 5 significant decimal arithmetic, we would like to get

$$[1,1] + [-10^{-10}, 10^{-10}] = [0.99999, 1.0001],$$

or in midpoint-radius representation

$$\langle 1,0\rangle + \langle 0,10^{-10}\rangle = \langle 1,10^{-10}\rangle.$$

Note that in the conversion between decimal and binary representation rounding the appropriate rounding mode must also be used where needed. For example, converting the point interval 0.1 to binary IEEE double precision we get an interval

with radius $1.3878 \cdot 10^{-17}$. The conversion between the infimum-supremum representation is straightforward but the infimum-supremum and midpoint may not be exactly representable.

Interval arithmetic applies also to complex numbers. A complex interval in the infimum-supremum representation is

$$[z_1, z_2] = \{z = x + iy \mid x \in [x_1, x_2],\ y \in [y_1, y_2]\}.$$

This defines a closed *rectangle in the complex plane* defined by the two real intervals,

$$[z_1, z_2] = [x_1, x_2] + i[y_1, y_2], \quad x_1 \le x_2, \quad y_1 \le y_2.$$

This can be written more compactly as $[z_1, z_2] := \{z \mid z_1 \le z \le z_2\}$, where we use the partial ordering

$$z \le w \quad \Longleftrightarrow \quad \Re z \le \Re w \ \ \& \ \ \Im z \le \Im w.$$

Complex interval operations in the infimum-supremum arithmetic are defined in terms of the real intervals in the same way as the complex operations are defined for complex numbers $z = x + iy$. For addition and subtraction the result coincides with the exact range. This is not the case for complex interval multiplication, where the result is a rectangle in the complex plane, whereas the actual range is not of this shape. Therefore, for complex intervals multiplication will result in an overestimation.

In the complex case the midpoint-radius representation is

$$\langle c, r\rangle := \{z \in \mathbf{C} \mid |z - c| \le r\}, \quad 0 \le r,$$

where the midpoint c now is a complex number. This represents a *closed circular disc in the complex plane*. The operational definitions (2.5.10) are still valid, except that some operations now are complex operations and that inversion becomes

$$1/\langle c, r\rangle = \langle \bar{c}/(|c|^2 - r^2), r/(|c|^2 - r^2)\rangle, \quad \text{for} \quad |c| > r,$$

where $\bar{c}$ is the complex conjugate of c. Complex interval midpoint-radius arithmetic is also called **circular arithmetic**. For complex multiplications it generates less overestimation than the infimum-supremum notation.

Although the midpoint-radius arithmetic seems more appropriate for complex intervals, most older implementations of interval arithmetic use infimum-supremum arithmetic. One reason for this is the overestimation caused also for real intervals by the operational definitions for midpoint-radius multiplication and division. Recently Rump [39] has shown that the overestimation is bounded by a factor 1.5 in radius and that midpoint-radius arithmetic allows for a much faster implementation for modern vector and parallel computers.

Range of Functions

One use of interval arithmetic is to enclose the range of a real valued function. This can be used, e.g., for localizing and enclosing global minimizers and global minima

of a real function of one or several variables in a region. It can also be used for verifying the nonexistence of a zero of $f(x)$ in a given interval.

Let $f(x)$ be a real function composed of a finite number of elementary operations and standard functions. If one replaces the variable x by an interval $[\underline{x}, \overline{x}]$ and evaluates the resulting interval expression one gets as result an interval denoted by $f([\underline{x}, \overline{x}])$. (It is assumed that all operations can be carried out.) A fundamental result in interval arithmetic is that this evaluation is inclusion monotonic, i.e.,

$$[\underline{x}, \overline{x}] \subseteq [\underline{y}, \overline{y}], \quad \Rightarrow \quad f([\underline{x}, \overline{x}]) \subseteq f([\underline{y}, \overline{y}]).$$

In particular this means that

$$x \subseteq [\underline{x}, \overline{x}] \;\Rightarrow\; f(x) \subseteq f([\underline{x}, \overline{x}]),$$

i.e., $f([x])$ contains the range of $f(x)$ over the interval $[\underline{x}, \overline{x}]$. A similar result holds also for functions of several variables $f(x_1, \ldots, x_n)$.

An important case when interval evaluation gives the exact range of a function is when $f(x_1, \ldots, x_n)$ is a rational expression, where each variable x_i occurs only once in the function.

Example 2.5.3.

In general narrow bounds cannot be guaranteed. For example, if $f(x) = x/(1-x)$ then

$$f([2,3]) = [2,3]/(1-[2,3]) = [2,3]/[-2,-1] = [-3,-1].$$

The result contains but does not coincide with the exact range $[-2, -3/2]$. However, if we rewrite the expression as $f(x) = 1/(1/x - 1)$, where x only occurs once, then we get

$$f([2,3]) = 1/(1/[2,3] - 1) = 1/[-2/3, -1/2] = [-2, -3/2],$$

which is the exact range.

When interval analysis is used in a naive manner as a simple technique for simulating forward error analysis it does not in general give sharp bounds on the total computational error. To get useful results the computations in general need to be arranged so that overestimation as far as possible is minimized. Often a refined design of the algorithm is required in order to prevent the bounds for the intervals from becoming unacceptably coarse. The answer $[-\infty, \infty]$ is of course always correct but not at all useful!

The remainder term in Taylor expansions includes a variable ξ, which is known to lie in an interval $\xi \in [a, b]$. This makes it suitable to treat the remainder term with interval arithmetic.

Example 2.5.4.

Evaluate for $[x] = [2, 3]$ the polynomial

$$p(x) = 1 - x + x^2 - x^3 + x^4 - x^5$$

Using exact interval arithmetic we find

$$p([2,3]) = [-252, 49]$$

(verify this!). This is an overestimate of the exact range, which is $[-182, -21]$. Rewriting $p(x)$ in the form $p(x) = (1-x)(1+x^2+x^4)$ we obtain the correct range. In the first example there is a **cancellation of errors** in the intermediate results (cf. Example 2.5.2), which is not revealed by the interval calculations.

Sometimes it is desired to compute a tiny interval that is guaranteed to enclose a real simple root x^* of $f(x)$. This can be done using an interval version of Newton's method. Suppose that the function $f(x)$ is continuously differentiable. Let $f'([x_0])$ denote an interval containing $f'(x)$ for all x in a finite interval $[x] := [a, b]$. Define the Newton operator on $N[x]$ by

$$N([x]) := m - \frac{f(|\ (x))}{f'([x])}. \tag{2.5.11}$$

For the properties of the iteration $[x_{k+1}) = N([x_k])$, see Sec. 6.3.4.

Another important application of interval arithmetic is to initial value problems for ordinary differential equations

$$y' = f(x, y), \quad y(x_0) = y_0, \quad y \in \mathbf{R}^n.$$

Interval techniques can be used to provide for errors in the initial values, as well as truncation and rounding errors, so that at each step intervals are computed that contain the actual solution. However, it is a most demanding task to construct an interval algorithm for the initial value problem, and they tend to be significantly slower than corresponding point algorithms. One problem is that a wrapping effect occurs at each step and causes the computed interval widths to grow exponentially. This is illustrated in the following example.

Example 2.5.5.

The recursion formulas

$$x_{n+1} = (x_n - y_n)/\sqrt{2}, \qquad y_{n+1} = (x_n + y_n)/\sqrt{2},$$

mean a series of 45-degree rotations in the xy-plane (see Fig. 2.3.5). By a two-dimensional interval one means a rectangle whose sides are parallel to the coordinate axes.

If the initial value (x_0, y_0) is given as an interval $[x_0] = [1-\epsilon, 1+\epsilon]$, $[y_0] = [-\epsilon, \epsilon]$ (see the dashed square, in the leftmost portion of Fig. 2.3.5), then (x_n, y_n) will, with *exact* performance of the transformations, also be a square with side 2ϵ, for all n (see the other squares in Fig. 2.3.5). If the computations are made using interval arithmetic, rectangles with sides parallel to the coordinate axis will, in each step, be circumscribed about the exact image of the interval one had in the previous step. Thus the interval is multiplied by $\sqrt{2}$ in each step. After 40 steps, for example, the interval has been multiplied by $2^{20} > 10^6$. This phenomenon, intrinsic to interval

computations, is called the **wrapping effect**. (Note that If one used discs instead of rectangular, there would not have been any difficulties of this example.)

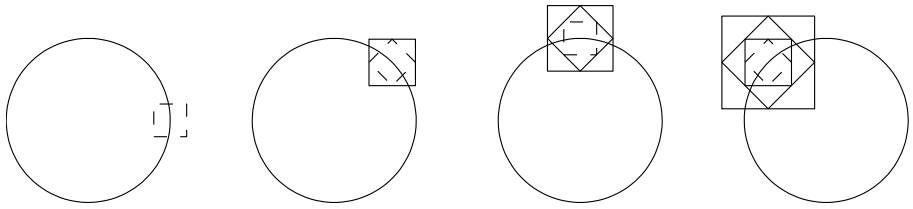


Figure 2.5.1. *Wrapping effect in interval analysis.*

Interval Matrix Computations

In order to treat multidimensional problems we introduce interval vectors and matrices. An interval vector is denoted by $[x]$ and has interval components $[x_i] = [\underline{x_i}, \overline{x_i}])$, $i = 1 : n$. Likewise an interval matrix $[A] = ([a_{ij}])$ has interval elements $[a_{ij}] = [\underline{a}_{ij}, \overline{a}_{ij}]$, $i = 1 : m$, $j = 1 : n$,

Operations between interval matrices and interval vectors are defined in an obvious manner. The interval matrix-vector product $[A][x]$ is the smallest interval vector, which contains the set $\{Ax \mid A \in [A],\ x \in [x]\}$, but normally does not coincide with it. By the inclusion property it holds that

$$\{Ax \mid A \in [A],\ x \in [x]\} \subseteq [A][x] = \left(\sum_{j=1}^{n} [a_{ij}][x_j] \right). \tag{2.5.12}$$

In general, there will be an overestimation in enclosing the image with an interval vector caused by the fact that the image of an interval vector under a transformation in general is not an interval vector. This phenomenon, intrinsic to interval computations, is called the **wrapping effect**.

Example 2.5.6.
We have

$$A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad [x] = \begin{pmatrix} [0,1] \\ [0,1] \end{pmatrix}, \quad \Rightarrow \quad A[x] = \begin{pmatrix} [0,2] \\ [-1,1] \end{pmatrix}.$$

Hence $b = (\,2 \quad -1\,)^T \in A[x]$, but there is no $x \in [x]$ such that $Ax = b$. (The solution to $Ax = b$ is $x = (\,3/2 \quad 1/2\,)^T$.)

The magnitude of an interval vector or matrix is interpreted component-wise and defined by

$$|\,[x]\,| = (|\,[x_1]\,|, |\,[x_2]\,|, \ldots, |\,[x_n]\,|)^T,$$

where the magnitude of the components are defined by

$$|\,[a,b]\,| = \max\{|x| \mid x \in [a,b]\}, \tag{2.5.13}$$

The ∞-norm of an interval vector or matrix is defined as the ∞-norm of their magnitude,

$$\|\,[x]\,\|_\infty = \|\,|\,[x]\,|\,\|_\infty, \qquad \|\,[A]\,\|_\infty = \|\,|\,[A]\,|\,\|_\infty. \tag{2.5.14}$$

In implementing matrix multiplication it is important to avoid case distinctions in the inner loops, because that would make it impossible to use fast vector and matrix operations. Using interval arithmetic it is possible to compute strict enclosures of the product of two matrices. Note that this is needed also in the case of the product of two point matrices since rounding errors will in general occur.

We assume that the command

$$\text{setround}(i), \quad i = -1, 0, 1,$$

sets the rounding mode to $-\infty$, to nearest, and to $+\infty$, respectively. (Recall that these rounding modes are supported by the IEEE standard.) Let A and B be point matrices and suppose we want to compute an interval matrix $[C]$ such that

$$fl(A \cdot B) \subset [C] = [C_{\inf}, C_{\sup}].$$

Then the following simple code, using two matrix multiplications, does that:

$$\begin{aligned} &\text{setround}(-1); \quad C_{\inf} = A \cdot B; \\ &\text{setround}(1); \quad\;\; C_{\sup} = A \cdot B; \end{aligned}$$

We next consider the product of a point matrix A and an interval matrix $[B] = [B_{\inf}, B_{\sup}]$. The following code performs this using four matrix multiplications:

$$\begin{aligned} &A_- = \min(A,\ 0); \quad A_+ = \max(A,\ 0); \\ &\text{setround}(-1); \\ &C_{\inf} = A_+ \cdot B_{\inf} + A_- \cdot B_{\sup}; \\ &\text{setround}(1); \\ &C_{\sup} = A_- \cdot B_{\inf} + A_+ \cdot B_{\sup}; \end{aligned}$$

(Note that the commands $A_- = \min(A,\ 0)$ and $A_+ = \max(A,\ 0)$ acts componentwise.) For an algorithm for computing the product of two interval matrices using eight matrix multiplications; see Rump [40].

Fast portable codes for interval matrix computations are now available. that makes use of the Basic Linear Algebra Subroutines (BLAS) and IEEE 754 standard. This makes it possible to efficiently use high-performance computers for interval computation. The MATLAB toolbox INTLAB (INTerval LABoratory)., by Rump [40, 39] is partcularly easy to use. It includes many useful subroutines, e.g., one to compute an enclosure of the difference between the solution and an approximate solution $x_m = C\text{mid}\,[b]$. Verified solutions of linear least squares problems can also be computed.

Review Questions

1. (a) Define the magnitude and mignitude of an interval $I = [a, b]$.
   (b) How is the ∞-norm of an interval vector defined?
2. Describe the two different ways of representing intervals used in real and complex interval arithmetic. Mention some of the advantages and drawbacks of each of these!
3. An important property of interval arithmetic is that the operations are inclusion monotonic. Define this term!
4. What is meant by the "wrapping effect" in interval arithmetic and what are its implications? Give some examples of where it occurs.

Problems

1. Carry out the following calculations in exact interval arithmetic:
   (a) $[0, 1] + [1, 2]$; (b) $[3, 3.1] - [0, 0, 2]$; (c) $[-4. - 1] \cdot [-6, 5]$;
   (d) $[2, 2] \cdot [-1, 2]$; (e) $[-1, 1]/[-2, -0.5]$; (f) $[-3, 2] \cdot [-3.1, 2.1]$;
2. Show that using the operational definitions (2.5.5) the product of the discs $\langle c_1, r_1\rangle$ and $\langle c_2, r_2\rangle$ contains zero if $c_1 = c_2 = 1$ and $r_1 = r_2 = \sqrt{2} - 1$.
3. (J. Stoer) Evaluate using Horner's scheme and exact interval arithmetic the cubic polynomial
   $$p(x) = ((x - 3)x + 3)x, \quad [x] = [0.9, 1.1].$$
   Compare the result with the exact range, which can be determined by observing that $p(x) = (x - 1)^3 + 1$.
4. Treat the Example 1.3.2 using interval analysis and four decimal digits. Starting with the inclusion interval $I_{10} = [0, 1/60] = [0, 0.01667]$ generate successively intervals I_k, $k = 9 : -1 : 5$, using interval arithmetic and the recursion
   $$I_{n-1} = 1/(5n) - I_n/5.$$

Notes and References

A treatment of many different aspects of number systems and floating point computations is given in Knuth [32, Chapter 4]. It contains an interesting overview of the historical development of number representation. Leibniz 1703 seems to have been the first to discuss binary arithmetic. He did not advocate it for practical calculations, but stressed its importance for number-theoretic investigations. King Charles XII of Sweden came upon the idea of radix 8 arithmetic in 1717. He felt this to be more convenient than the decimal notation and considered introducing octal arithmetic into Sweden. He died in battle before decreeing such a change!

In the early days of computing floating point computations were not built into the hardware but implemented in software. The earliest subroutines for floating point arithmetic were probably those developed by J. H. Wilkinson at the National Physical Laboratory, England, in 1947. A general source on floating point computation is Sterbenz [42]. An excellent tutorial on IEEE 754 standard for floating-point arithmetic, defined in [20, 1985], is Goldberg [25, 1991]. A self-contained, accessible and easy to read introduction with many illustrating examples is the monograph by Overton [36, 2001]. An excellent treatment on floating point computation, rounding error analysis, and related topics is given in Higham [29, Chapter 2]. Different aspects of accuracy and reliability are discussed in [19].

Forsythe [22, 1970] gives a good introduction for a mathematical audience of some problems inherent in numerical computations. Numerous examples in which incorrect answers are obtained from plausible numerical methods can be found in Fox [23, 1971].

Statistical analysis of rounding errors goes back to an early paper of Goldstine and von Neumann [26, 1951]. Barlow and Bairess [6] have investigated the distribution of rounding errors for different modes of rounding under the assumption that the mantissa of the operands are from a logarithmic distribution.

Backward error analysis was developed and popularized by J. H. Wilkinson in the 1950s and 1960s and the classic treatise on rounding error analysis is [45]. The more recent survey [47] gives a good summary and a historical background. Kahan [30] gives an in depth discussion of rounding error analysis with examples how flaws in the design of hardware and software in computer systems can have undesirable effects on accuracy. The normwise analysis is natural for studying the effect of orthogonal transformations in matrix computations; see Wilkinson [45]. The componentwise approach, used in perturbation analysis for linear systems by Bauer [7], has recently gained in popularity.

Condition numbers are often viewed pragmatically as the coefficients of the backward errors in bounds on forward errors. Wilkinson in [45] avoids a precise definition of condition numbers in order to use them more freely. The more precise limsup definition in Definition 2.4.8 is usually attributed to Rice [38].

The modern development of interval arithmetic was initiated by the work of R. E. Moore [34, 1966]. Interval arithmetic has since been developed into a useful tool for many problems in scientific computing and engineering. A more general interval arithmetic, which allows unbounded intervals, which occur when dividing by an interval containing zero was introduced by W. Kahan in unpublished lecture notes; see Hargreaves [27], which also give a good introduction to interval arithmetic, and includes a short tutorial on INTLAB.

Bibliography

[1] Götz Alefeld and Jürgen Herzberger. *Introduction to Interval Computation*. Academic Press, New York, 1983. translated frmo the German by Jon Rokne.

[2] Götz Alefeld and Günter Mayer. Interval analysis: theory and applications. *J. Comput. Appl. Math.*, 121:421–464, 1985.

[3] David H. Bailey. Algorithm 719: Multiprecision translation and execution of FORTRAN programs. *ACM Trans. Math. Software*, 19:3:288–319, 1993.

[4] David H. Bailey. A fortran 90-based multiprecision system. *ACM Trans. Math. Software*, 21:4:379–387, 1995.

[5] David H. Bailey, Jon M. Borwein, Peter B. Borwein, and Simon Plouffe. The quest for pi. *Notes Amer. Math. Soc.*, 19:1:50–57, 1975.

[6] Jesse L. Barlow and E. H. Bairess. On roundoff error distributions in floating point and logarithmic arithmetic. *Computing*, 34:325–347, 1985.

[7] F. L. Bauer. Genauigkeitsfragen bei der Lösung linearer Gleichungssysteme. *ZAMM*, 46:409–421, 1966.

[8] Folkmar Bornemann, Dirk Laurie, Stan Wagon, and Jürg Waldvogel. *The SIAM 100-digit Challenge. A Study in High-Accuracy Numerical Computing*. SIAM, Philadelphia, PA, 2004.

[9] Richard P. Brent. Algorithm 524: A Fortran multiple-precision arithmetic package. *ACM Trans. Math. Software*, 4:1:71–81, 1978.

[10] Richard P. Brent. A Fortran multiple-precision arithmetic package. *ACM Trans. Math. Software*, 4:1:57–70, 1978.

[11] Frančoise Chaitin-Chatelin and Valerie Frayssé. *Lectures on Finite Precision Computations*. SIAM, Philadelphia, PA, 1996.

[12] Frančoise Chaitin-Chatelin and Elisabeth Traviesas-Cassan. PRECISE and the quality of reliable numerical software. In Bo Einarsson, editor, *Accuracy and Reliability in Scientific Computing*, pages 197–243. SIAM Publications, Philadelphia, 2005.

[13] B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *Maple V Library Reference Manual*. Springer-Verlag, Berlin, 1991.

[14] W. J. Cody. Implementation and testing of function software. In P. C. Messina and A. Murli, editors, *Problems and Methodologies in Mathematical Software*, pages 24–47. Springer-Verlag, Berlin, 1982.

[15] W. J. Cody. Algorithm 714: CELEFUNT: A portable test package for complex elementary functions. *ACM Trans. Math. Software*, 14:4:121, 1993.

[16] W. J. Cody and W. Waite. *Software Manual for the Elementary Functions*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[17] James W. Demmel. Underflow and the reliability of numerical software. *SIAM J. Sci. Stat. Comput.*, 5:4:887–919, 1984.

[18] Alan Edelman. The mathematics of the Pentium division bug. *SIAM Review*, 39:54–67, 1997.

[19] Bo Einarsson. *Accuracy and Reliability in Scientific Computing*. SIAM Publications, Philadelphia, 2005.

[20] IEEE Standard for Binary Floating-Point Arithmetic. ANSI/IEEE standard 754-1985. *Reprinted in SIGPLAN Notices*, 22:2:9–25, 1987.

[21] IEEE Standard for Radix-Independent Floating Point Arithmetic. *ANSI/IEEE Standard 854-1987*. IEEE Computer Society, New York, 1987, 1987.

[22] George E. Forsythe. Pitfalls in computation, or why a math book isn't enough. Technical Report CS 147, Computer Science Department, Stanford University, Stanford, CA, 1970.

[23] Leslie Fox. How to get meaningless answers in scientific computation (and what to do about it),. *IMA Bulletin*, 7:10:296–302, 1971.

[24] Walter Gautschi. *Numerical Analysis, an Introduction*. Birkhäuser, Boston, MA, 1997.

[25] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23:5–48, 1991.

[26] H. H. Goldstine and John von Neumann. Numerical inverting of matrices of high order II. *Proceedings Amer. Math. Soc.*, 2:188–202, 1951.

[27] G. I. Hargreaves. Interval analysis in MATLAB. Numer. anal. report 418, Department of Mathematics, University of Manchester, 2002.

[28] J. F. Hart, E. W. Cheney, C. L. Lawson, H. J. Maehly, C. K. Mesztenyi, J. F. Rice, Jr. H. G. Thacher, and C. Witzgall. *Computer Approximations*. John Wiley, New York, New York, 1968.

[29] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, second edition, 2002.

[30] W. Kahan. A survey of error analysis. In B. Dejon and P. Henrici, editors, *Proceedings IFIP Congress Ljubljana, Information Processing 1971*, pages 1214–1239. North-Holland, Amsterdam, 1971.

[31] R. Kearfott. Interval computations: Introduction, uses, and resources. *Euromath. Bulletin*, 2:1:95–112, 1996.

[32] Donald E. Knuth. *The Art of Computer Programming, Vol. 2. Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1997.

[33] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, A. Kapur, M. C. Martin, T. Tung, and D. J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. LAPACK working note 149 Tech. Report CS-00-451, Department of Computer Science, University of Tennessee, Knoxville, TN, 2000.

[34] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[35] J.-M. Muller. *Elementary Functions: Algorithm and Implementation*. Birkhäuser, Boston, MA, 1997.

[36] Michael L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. SIAM, Philadelphia, PA, 2001.

[37] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in Fortran; The Art of Scientific Computing*. Cambridge University Press, Cambridge, GB, second edition, 1992.

[38] John R. Rice. A theory of condition. *SIAM J. Numer. Anal.*, 3:2:287–310, 1966.

[39] Sigfried M. Rump. Fast and parallel interval arithmetic. *BIT*, 39:3:534–554, 1999.

[40] Sigfried M. Rump. INTLAB—INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[41] Robert D. Skeel. Roundoff error and the patriot missile. *SIAM Review*, 25:11, 1992.

[42] P. H. Sterbenz. *Floating Point Computation*. Prentice-Hall, Englewood Cliffs, NJ, 1974.

[43] Lloyd N. Trefethen. The definition of numerical analysis. *SIAM News*, November 1992, 1992.

[44] W. Tucker. A rigorous ODE solver and Smale's 14th problem. *Found. Comput. Math.*, 2:1:53–117, 2002.

[45] James H. Wilkinson. *Rounding Error in Algebraic Processes*. Notes on Applied Science No. 32. Her Majesty's Stationery Office, London, UK, 1963. Reprinted by Dover, New York, 1994.

[46] James H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.

[47] James H. Wilkinson. Error analysis revisited. *IMA Bull.*, 22:192–200, 1986.

[48] Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Champaign, IL, fourth edition, 1999.

Index

absolute error, 4
accuracy, 5
    automatic control of, 61–62
algorithm, 41
    divide and conquer, 38
    mathematically equivalent, 25
    numerically equivalent, 25
    stable, 55
    unstable, 53
arithmetic
    circular, 66
    complex, 26
    fixed-point, 10
    floating-point, 21–27
    interval, 63–68
    multiple precision, 18–19
    standard model, 22
    unnormalized floating point, 61

backward error
    analysis, 27
backward stability, 54
binary
    number system, 8
    point, 9
    system, 8
BLAS, 29

cancellation
    of errors, 62
    of terms, 34–37
circular arithmetic, 66
compensated summation, 30
complex arithmetic, 26
condition number, 47
    absolute, 47
    of matrix, 50
    of problem, 41–49
    relative, 47
conversion
    between number systems, 9
correct decimals, 6

double precision
    simulating, 30
double rounding, 13

elementary functions, 17–18
error
    absolute, 4
    bound, 43
    floating point rounding, 27–30
    human, 2
    maximal, 46
    propagation, 43–68
        general formula, 45
    relative, 4
    rounding, 1
    sources of, 1–4
    standard, 32
    systematic, 1
    truncation, 2
error analysis
    backward, 53
    forward, 53
    running, 61
error bounds, 4
Euclidian norm
    computing, 34
experimental perturbations, 62
exponent, 11
    overflow, 33
    underflow, 33

floating point

number, 11
representation, 10
standard arithmetic, 13–17
forward stability, 54
fused multiply-add, 22

gradual underflow, 16
guard digits, 16

Heron's formula, 39
Hilbert matrix, 51

IEEE 754 standard, 13–17
ill-conditioned
problem, 48
inner product
accurate, 29
error analysis, 28
input data, 41
interval
complex, 66
midpoint-radius representation, 66
interval arithmetic
infimum-supremum representation, 63
midpoint-radius representation, 65
operational definitions, 64
interval operations
inclusion monotonic, 65
INTLAB, 71

Jacobian matrix, 49

machine epsilon, 12
magnitude
of interval, 63
mantissa, 11
matrix
ill-conditioned, 51
well-conditioned, 51
matrix multiplication
error bound, 29
maximal
error, 46
mignitude
of interval, 63
number system
binary, 8
floating-point, 11
hexadecimal, 8
octal, 8
numerical method, 42
numerical problem, 41

output data, 41

perturbation
experimental, 62
of linear systems, 50–52
perturbation bound
component-wise, 51
position system, 7–9
precision, 5
double, 14
single, 14
problem
ill-conditioned, 48
well-conditioned, 48
Pythagorean sum, 33

radix, 11
range reduction, 17
relative error, 4
rounding, 6
chopping, 6

scale factors (fixed point), 10
significant digits, 6
single precision, 14
standard error, 4
subdistributivity, 64
summation
compensated, 30

tablemaker's dilemma, 6
Toeplitz matrix, 56
total differential, 45
truncation error, 2

ulp, 12
unit roundoff, 12

well-conditioned

problem, 48
wobbling, 11
word-length, 10
wrapping effect, 68

Contents

3 Series, Operators and Continued Fractions **1**

3.1 Some Basic Facts about Series 1
3.1.1 Introduction 1
3.1.2 Power Series 7
3.1.3 Analytic Continuation 15
3.1.4 Manipulating Power Series 17
3.1.5 Formal Power Series 23
Review Questions 26
Problems and Computer Exercises 27
3.2 More About Series 36
3.2.1 Laurent and Fourier Series 36
3.2.2 The Cauchy–FFT Method 38
3.2.3 Chebyshev Polynomials 44
3.2.4 Perturbation Expansions 49
3.2.5 Ill-Conditioned Series 52
3.2.6 Divergent or Semiconvergent Series 58
Review Questions 61
Problems and Computer Exercises 61
3.3 Difference Operators and Operator Expansions 67
3.3.1 Properties of Difference Operators 67
3.3.2 The Calculus of Operators 72
3.3.3 The Peano Theorem 84
3.3.4 Approximation Formulas by Operator Methods . . . 89
3.3.5 Single Linear Difference Equations 96
Review Questions 106
Problems and Computer Exercises 107
3.4 Acceleration of Convergence 117
3.4.1 Introduction 117
3.4.2 Comparison Series and Aitken Acceleration 118
3.4.3 Euler's Transformation 124
3.4.4 Euler–Maclaurin's Formula 131
3.4.5 Repeated Richardson Extrapolation 141
Review Questions 148
Problems and Computer Exercises 148

3.5 Continued Fractions and Padé Approximants 161
3.5.1 Continued Fractions 161
3.5.2 Padé Approximants. 169
3.5.3 The Epsilon Algorithm. 174
3.5.4 The QD Algorithm. 176
Review Questions 180
Problems and Computer Exercises 181

Bibliography **185**

Index **188**

Chapter 3

Series, Operators and Continued Fractions

3.1 Some Basic Facts about Series

3.1.1 Introduction

Series expansions are a very important aid in numerical calculations, especially for quick estimates made in hand calculation—for example, in evaluating functions, integrals, or derivatives. Solutions to differential equations can often be expressed in terms of series expansions. Since the advent of computers it has, however, become more common to treat differential equations directly, using, e.g., finite difference or finite element approximations instead of series expansions. Series have some advantages, especially in problems containing parameters. Automatic methods for formula manipulation and some new numerical methods provide, however, new possibilities for series.

In this section we will discuss general questions concerning the use of infinite series for numerical computations including, e.g., the estimation of remainders, power series and various algorithms for computing their coefficients. Often a series expansion can be derived by simple operations with a known series. We also give an introduction to formal power series. The next section treats perturbation expansions, ill-conditioned and semi-convergent expansions, from the point of view of computing.

Methods and results will sometimes be formulated in terms of *series*, sometimes in terms of *sequences*. These formulations are equivalent, since the sum of an infinite series is defined as the limit of the the sequence s_n of its partial sums

$$S_n = a_1 + a_2 + \ldots + a_n.$$

Conversely, any sequence $S_1, S_2, S_3, \ldots$ can be written as the partial sums of a series,

$$S_1 + (S_2 - S_1) + (S_3 - S_2) + \ldots.$$

We start with some simple examples and some general rules for the approximation of remainders.

Example 3.1.1.

Compute, to five decimals, $y(0.5)$, where $y(x)$ is the solution to the differential equation $y'' = -xy$, with initial conditions $y(0) = 1$, $y'(0) = 0$. The solution cannot be simply expressed in terms of elementary functions. We shall use the method of undetermined coefficients. Thus we try substituting a series of the form:

$$y(x) = \sum_{n=0}^{\infty} c_n x^n = c_0 + c_1 x + c_2 x^2 + \cdots.$$

Differentiating twice we get

$$\begin{aligned} y''(x) &= \sum_{n=0}^{\infty} n(n-1)c_n x^{n-2} \\ &= 2c_2 + 6c_3 x + 12c_4 x^2 + \cdots + (m+2)(m+1)c_{m+2}x^m + \cdots, \\ -xy(x) &= -c_0 x - c_1 x^2 - c_2 x^3 - \cdots - c_{m-1}x^m - \cdots. \end{aligned}$$

Equating coefficients of x^m in these series gives

$$c_2 = 0, \qquad (m+2)(m+1)c_{m+2} = -c_{m-1}, \quad m \geq 1.$$

It follows from the initial conditions that $c_0 = 1$, $c_1 = 0$. Thus $c_n = 0$, if n is not a multiple of 3, and using the recursion we obtain

$$y(x) = 1 - \frac{x^3}{6} + \frac{x^6}{180} - \frac{x^9}{12,960} + \cdots.$$

This gives $y(0.5) = 0.97925$. The x^9 term is ignored, since it is less than $2 \cdot 10^{-7}$. In this example also the first neglected term gives a rigorous bound for the error (i.e. for the remaining terms), since the absolute value of the term decreases, and the terms alternate in sign.

Since the calculation was based on a trial substitution, one should, strictly speaking, prove that the series obtained defines a function which satisfies the given problem. Clearly, the series converges at least for $|x| < 1$, since the coefficients are bounded. (In fact the series converges for all x.) Since a power series can be differentiated term by term in the interior of its interval of convergence, the proof presents no difficulty. Note, in addition, that the finite series obtained for $y(x)$ by breaking off after the x^9-term is the exact solution to the following *modified differential equation*:

$$y'' = -xy - \frac{x^{10}}{12,960}, \qquad y(0) = 1, \qquad y'(0) = 0,$$

where the "perturbation term" $-x^{10}/12,960$ has magnitude less than 10^{-7} for $|x| \leq 0.5$.[1]

[1] We shall see, in Volume III, Chapter 13, how to find a rigorous bound for the difference between the solutions of a differential system and a modified differential system.

In practice, one is seldom seriously concerned about a rigorous error bound when the computed terms decrease rapidly, and it is "obvious" that the terms will continue to decrease equally quickly. One can then break off the series and use either the last included term or a coarse estimate of the **first neglected term** as an estimate of the remainder.

This rule is not very precise. *How rapidly is "rapidly"?* Questions like this occur everywhere in scientific computing. If mathematical rigor costs little effort or little extra computing time, then it should, of course, be used. Often, however, an error bound that is both rigorous and realistic may cost more than what is felt reasonable for (say) a one-off problem.

In problems, where guaranteed error bounds are not asked for, when it is enough to obtain a feeling for the reliability of the results, one can handle these matters in the same spirit as one handles risks in every day life. It is then a matter of experience to formulate a simple and *sufficiently* reliable **termination criterion** based on the automatic inspection of the successive terms.[2]

The unexperienced scientific programmer may, however, find such questions hard, also in simple cases. In the production of general purpose mathematical software, or in a context where an inaccurate numerical result can cause a disaster, such questions are serious and sometimes hard also for the experienced scientific programmer. For this reason, we shall formulate a few theorems, with which one can often transform the feeling that "the remainder is negligible" to a mathematical proof. There are, in addition, actually numerically useful *divergent* series; see Sec. 3.2.6. When one uses such series, estimates of the remainder are clearly essential.

Assume that we want to compute a quantity S, which can be expressed in a series expansion, $S = \sum_{j=0}^{\infty} a_j$, and set

$$S_n = \sum_{j=0}^{n} a_j, \qquad R_n = S - S_n.$$

We call $\sum_{j=n+1}^{\infty} a_j$ the **tail** of the series; a_n is the "last included term" and a_{n+1} is the "first neglected term". The *remainder* R_n with reversed sign is called the *truncation error*.[3]

The tail of a convergent series can often be compared to a series with a known sum, for example, a geometric series, or with an integral which can be computed directly.

Theorem 3.1.1. *Comparison with a Geometric Series.*

If $|a_{j+1}| \le k|a_j|$, $\forall j \ge n$, *where* $k < 1$, *then*

$$|R_n| \le \frac{|a_{n+1}|}{1-k} \le \frac{k|a_n|}{1-k}.$$

In particular if $k < 1/2$, *then it is true that the absolute value of the remainder is less than the last included term.*

[2] Termination criteria for iterative methods will be discussed in Sec. 6.1.3.

[3] In this terminology the remainder is the *correction* one has to make in order to eliminate the error.

Proof. By induction, one finds that $|a_j| \le k^{j-1-n}|a_{n+1}|$, $j \ge n+1$, since

$$|a_j| \le k^{j-1-n}|a_{n+1}| \quad \Rightarrow \quad |a_{j+1}| \le k|a_j| \le k^{j-n}|a_{n+1}|.$$

Thus

$$|R_n| \le \sum_{j=n+1}^{\infty} |a_j| \le \sum_{j=n+1}^{\infty} k^{j-1-n}|a_{n+1}| = \frac{|a_{n+1}|}{1-k} \le \frac{k|a_n|}{1-k},$$

according to the formula for the sum of an infinite geometric series. The last statement follows from the inequality $k/(1-k) < 1$, when $k < 1/2$. □

Example 3.1.2. *Power series with slowly varying coefficients.*

Let $a_j = j^{1/2}\pi^{-2j}$. Then $a_6 = 2.4 \cdot 0.0000011 < 3 \cdot 10^{-6}$. Further,

$$\frac{|a_{j+1}|}{|a_j|} \le \frac{(j+1)^{1/2}}{j^{1/2}} \frac{\pi^{2j-2}}{\pi^{-2j}} \le (1+1/6)^{1/2}\pi^{-2} < 0.11,$$

for $j \ge 6$. Thus, by Theorem 3.1.1 $|R_6| < 3 \cdot 10^{-6} \dfrac{0.11}{1-0.11} < 4 \cdot 10^{-7}$.

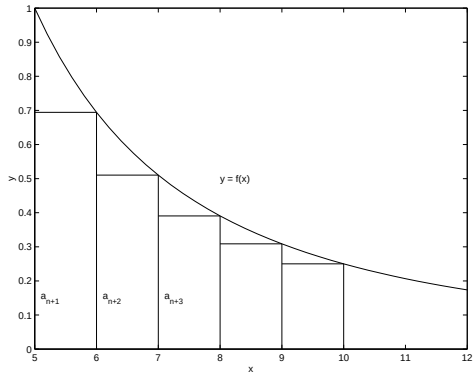


Figure 3.1.1. *Comparison with an integral, (n=5).*

Theorem 3.1.2. *Comparison of a Series with an Integral.*

If $|a_j| \le f(j)$ for all $j \ge n$, where $f(x)$ is a nonincreasing function for $x \ge n$, then

$$|R_n| \le \int_n^{\infty} f(x)dx,$$

which yields an upper bound for $|R_n|$, if the integral is finite.

If $a_j = f(j) > 0$ for all $j \ge n+1$, we also obtain a lower bound for the error, namely $\int_{n+1}^{\infty} f(x)dx$.

Proof. See Figure 3.1.1. □

Example 3.1.3.

When a_j is slowly decreasing, the two error bounds are typically rather close to each other, and are hence rather realistic bounds, much larger than the first neglected term a_{n+1}. Let $a_j = 1/(j^3+1)$, $f(x) = x^{-3}$. It follows that

$$0 < R_n \le \int_n^\infty x^{-3}dx = n^{-2}/2.$$

In addition this bound gives an asymptotically correct estimate of the remainder, as $n \to \infty$. which shows that R_n is here significantly larger than the first neglected term.

For alternating series, however, the situation is typically quite different.

Definition 3.1.3.

A series is **alternating** *for $j \ge n$ if, for all $j \ge n$, a_j and a_{j+1} have opposite signs, or equivalently* sign $a_j = -$sign a_{j+1}, *where* sign x *(read "signum" of x), is defined by*

$$\text{sign } x = \begin{cases} +1, & \text{if } x > 0; \\ 0, & \text{if } x = 0; \\ -1, & \text{if } x < 0. \end{cases}$$

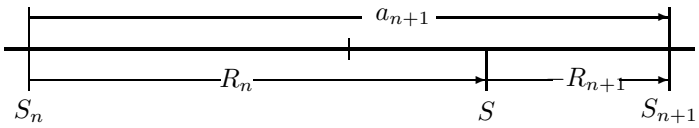


Figure 3.1.2. *Illustration to Theorem 3.1.4*

Theorem 3.1.4.

If R_n and R_{n+1} have opposite signs, then S lies between S_n and S_{n+1}. Furthermore

$$S = \frac{1}{2}(S_n + S_{n+1}) \pm \frac{1}{2}|a_{n+1}|.$$

We also have the weaker results:

$$|R_n| \le |a_{n+1}|, \qquad |R_{n+1}| \le |a_{n+1}|, \qquad \text{sign } R_n = \text{sign } a_{n+1}.$$

This theorem has non-trivial applications to practically important divergent sequences; see Sec. 3.2.6.

Proof. The fact that R_{n+1} and R_n have opposite signs means, quite simply, that one of S_{n+1} and S_n is too large and the other is too small, i.e. that S lies between S_{n+1} and S_n. Since $a_{n+1} = S_{n+1} - S_n$, one has for positive values of a_{n+1}, the situation shown in Figure 3.1.2. From this figure, and an analogous one for the case of $a_{n+1} < 0$, the remaining assertions of the theorem clearly follow. □

The actual error of the average $\frac{1}{2}(S_n + S_{n+1})$ is, for slowly convergent alternating series, usually much smaller than the error bound $\frac{1}{2}|a_{n+1}|$. For example, if $S_n = 1 - \frac{1}{2} + \frac{1}{3} - \ldots \pm \frac{1}{n}$, $\lim S_n = \ln 2 \approx 0.6931$, the error bound for $n = 4$ is 0.1, while the actual error is less than 0.01. A systematic exploration of this observation, by means of repeated averaging. is carried out in Sec. 3.4.3.

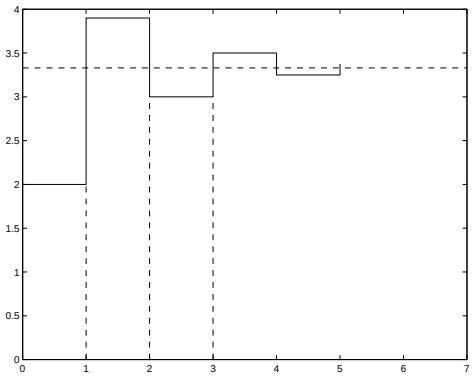


Figure 3.1.3. *The sum of an alternating series.*

In Example 1.2.3 the error function was approximated for $|x| \in [-1, 1]$ by a power series. The series has terms of alternating sign, and the absolute values of the terms decrease monotonically to zero. For such a series the above theorem can be used to prove that the first neglected term gives a rigorous error estimate.

Theorem 3.1.5.

For an alternating series, the absolute values of whose terms approach zero monotonically, the remainder has the same sign as the first neglected term a_{n+1}, and the absolute value of the remainder does not exceed $|a_{n+1}|$. (It is well known that such a series is convergent).

Proof. (Sketch) That the theorem is true is almost clear from Figure 3.1.3. The figure shows how S_j depends on j when the premises of the theorem are fulfilled. A formal proof is left to the reader. □

The use of this theorem was illustrated in Examples 3.1.1 and 3.1.2. An important generalization is given as Problem 3.2.1(g).

In the preceding theorems the ideas of well known convergence criteria are extended to bounds or estimates of the error of a truncated expansion. In Sec. sec3.4,

we shall see a further extension of these ideas, namely for *improving the accuracy* obtained from a sequence of truncated expansions. This is known as *convergence acceleration*.

3.1.2 Power Series

Consider an expansion into powers of a complex variable z, and suppose that it is convergent for some $z \neq 0$, and denote its sum by $f(z)$,

$$f(z) = \sum_{j=0}^{\infty} a_j z^j, \qquad z \in \mathbf{C}. \tag{3.1.1}$$

It is then known from complex analysis that the series (3.1.1) either converges for all z, or it has a **circle of convergence** with radius ρ, such that it either converges for all $|z| < \rho$, and diverges for $|z| > \rho$. (For $|z| = \rho$ convergence or divergence is possible). The radius of convergence is determined by the relation

$$\rho = \limsup |a_n|^{-1/n}. \tag{3.1.2}$$

Another formula is $\rho = \lim |a_n|/|a_{n+1}|$, *if this limit exists*.

The function $f(z)$ can be expanded into powers of $z - a$ around any point of analyticity,

$$f(z) = \sum_{j=0}^{\infty} a_j (z-a)^j, \quad z \in \mathbf{C}. \tag{3.1.3}$$

By **Taylor's formula** the coefficients are given by

$$a_0 = f(a), \qquad a_j = f^{(j)}(a)/j!, \quad j \geq 1. \tag{3.1.4}$$

This infinite series is in the general case called a Taylor series, while the special case, $a = 0$, is by tradition called a Maclaurin series.[4]

The function $f(z)$ is analytic inside its circle of convergence, and has at least one singular point on its boundary. The singularity of f, which is closest to the origin, can often be found easily from the expression that defines $f(z)$; so the radius of convergence of a Maclaurin series can often be easily found.

Note that these Taylor coefficients are *uniquely determined* for the function f. This is true also for a non-analytic function, for example if $f \in C^p[a, b]$, although in this case the coefficient a_j exists only for $j \leq p$. Also the remainder formulas (3.1.5), (3.1.7), require only that $f \in C^n$. It is thus not necessary that the infinite expansion converges or even exists.

There are several expressions for the remainder $R_n(z)$, when the expansion for $f(z)$ is truncated after the term that contains z^{n-1}. In order to simplify the notation, *we put* $a = 0$, *i.e. we consider the Maclaurin series*. The following

[4] Brook Taylor (1685–1731), who announced his theorem in 1712, and Colin Maclaurin (1698–1746) were British mathematicians.

integral form can be obtained by the application of repeated integration by parts to the integral $z\int_0^1 f'(zt)\,dt$:

$$R_n(z) = z^n \int_0^1 \frac{(1-t)^{n-1}}{(n-1)!} f^{(n)}(zt)\,dt; \tag{3.1.5}$$

the details are left for Problem 24 (b). From this follows the upper bound

$$|R_n(z)| \le \frac{1}{n!}|z|^n \max_{0\le t\le 1} |f^{(n)}(zt)|. \tag{3.1.6}$$

This holds also in the complex case; if f is analytic on the segment from 0 to z, one integrates along this segment, i.e. for $0 \le t \le 1$, otherwise another path is to be chosen.

For a real-valued function, **Lagrange's formula**[5] for the remainder

$$R_n(x) = \frac{f^{(n)}(\xi)x^n}{n!}, \qquad \xi \in [0, x], \tag{3.1.7}$$

is obtained by the mean value theorem of integral calculus.

For complex-valued functions and, more generally, for vector-valued functions the mean value theorem and Lagrange's remainder term are not valid with a single ξ. (Sometimes componentwise application with different ξ is possible.) A different form for the remainder, valid in the complex plane is given in Sec. sec3.1.cfft, in terms of the **maximum modulus** $M(r) = \max_{|z|=r} |f(z)|$, which may sometimes be easier to estimate than the nth derivative. A power series is uniformly convergent in any closed bounded region strictly inside its circle of convergence. Roughly speaking, the series can be manipulated like a polynomial, as long as z belongs to such a region;

- it can be integrated or differentiated term by term,
- substitutions can be performed, and terms can be rearranged,
- it can be multiplied by another power series, etc.

Theorem 3.1.6.
If $f(z) = \sum a_j z^j$, $g(z) = \sum b_k z^k$, *then*

$$f(z)g(z) = \sum c_n z^n, \quad c_n = \sum_{j=0}^{n} a_j b_{n-j}. \tag{3.1.8}$$

The expression on the right side of (3.1.8) *is called the* **convolution** *or the* **Cauchy product** *of the coefficient sequences of* f *and* g.

[5] Joseph Louis Lagrange (1736–1813) was born in Turin, Italy. In 1766 he succeeded Euler in Berlin but in 1787 went to Paris where he remained until his death. He gave fundamental contributions to most branches of Mathematics and Mechanics.

The use of the Taylor coefficient formula and Lagrange's form of the remainder may be inconvenient, and it is often easier to obtain an expansion by manipulating some known expansions. The geometric series,

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \cdots + z^{n-1} + \frac{z^n}{1-z}, \quad z \neq 1, \tag{3.1.9}$$

is of particular importance; note that the remainder $z^n/(1-z)$ is valid even when the expansion is divergent.

Example 3.1.4.
Set $z = -t^2$ in the geometric series, and integrate:

$$\int_0^x (1+t^2)^{-1}\,dt = \sum_{j=0}^{n-1} \int_0^x (-t^2)^j\,dt + \int_0^x (-t^2)^n (1+t^2)^{-1}\,dt.$$

Using the mean-value theorem of integral calculus on the last term we get

$$\arctan x = \sum_{j=0}^{n-1} \frac{(-1)^j x^{2j+1}}{2j+1} + \frac{(1+\xi^2)^{-1}(-1)^n x^{2n+1}}{2n+1}, \tag{3.1.10}$$

for some $\xi \in \operatorname{int}[0, x]$. Both the remainder term and the actual derivation are much simpler than what one would get by using Taylor's formula with Lagrange's remainder term. Note also that Theorem 3.1.4 is applicable to the series obtained above for all x and n, even for $|x| > 1$, when the infinite power series is divergent.

Some useful expansions are collected in Table 3.1.1.These formulas will be used often without a reference; the reader is advised to memorize the expansions. "Remainder ratio" means the *ratio* of the remainder to the first neglected term, if $x \in \mathbf{R}$; ξ means a number between 0 and x. Otherwise these expansions are valid in the unit circle of $\mathbf{C}$ or in the whole of $\mathbf{C}$.

The binomial coefficients are, also for non-integer k, defined by

$$\binom{k}{n} = \frac{k(k-1)\cdots(k-n+1)}{1\cdot 2\cdots n}.$$

Depending on the context, they may be computed by one of the following well known recurrences:

$$\binom{k}{(n+1)} = \binom{k}{n}\frac{(k-n)}{(n+1)}; \quad \text{or} \quad \binom{k+1}{n} = \binom{k}{n} + \binom{k}{n-1}, \tag{3.1.11}$$

with appropriate initial conditions. The latter recurrence follows from the matching of the coefficients of t^n in the equation $(1+t)^{k+1} = (1+t)(1+t)^k$. (Compare the Pascal triangle.) The explicit formula $\binom{k}{n} = \frac{k!}{n!(k-n)!}$, for integers k, n, is to be avoided, if k can become large, because $k!$ has overflow for $k \geq 170$ in IEEE double precision.

Table 3.1.1. *Maclaurin expansions for some elementary functions.*

Function	Expansion ($x \in \mathbf{C}$)	Remainder ratio ($x \in \mathbf{R}$)
$(1-x)^{-1}$	$1 + x + x^2 + x^3 + \cdots$ if $\|x\| < 1$	$(1-x)^{-1}$ if $x \neq 1$
$(1+x)^k$	$1 + kx + \binom{k}{2}x^2 + \cdots$ if $\|x\| < 1$	$(1+\xi)^{k-n}$ if $x > -1$
$\ln(1+x)$	$x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$ if $\|x\| < 1$	$(1+\xi)^{-1}$ if $x > -1$
e^x	$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$ all x	e^ξ, all x
$\sin x$	$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$ all x	$\cos\xi$, all x, n odd
$\cos x$	$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$ all x	$\cos\xi$, all x, n even
$\frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$	$x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots$ if $\|x\| < 1$	$\frac{1}{1-\xi^2}$, $\|x\| < 1$, n even
$\arctan x$	$x - \frac{x^3}{3} + \frac{x^5}{5} + \cdots$ if $\|x\| < 1$	$\frac{1}{1+\xi^2}$, all x

The exponent k in $(1+x)^k$ is not necessarily an integer; it can even be an irrational or a complex number. This function may be defined as $(1+x)^k = e^{k\ln(1+x)}$. Since $\ln(1+x)$ is **multi-valued**, $(1+x)^k$ is multi-valued too, unless k is an integer. We can, however, make them single-valued by forbidding the complex variable x to take real values less than -1. In other words, we make a **cut** along the real axis from -1 to $-\infty$ that the complex variable must not cross. (The cut is outside the circle of convergence.) We obtain the **principal branch** by requiring that $\ln(1+x) > 0$ if $x > 0$. Let $1 + x = re^{i\phi}$, $r > 0$, $\phi \to \pm\pi$. Note that

$$1 + x \to -r, \qquad \ln(1+x) \to \ln r + \begin{cases} +i\pi, & \text{if } \phi \to \pi; \\ -i\pi, & \text{if } \phi \to -\pi. \end{cases} \tag{3.1.12}$$

Two important power series, not given in Table 3.1.1, are:

Gauss' hypergeometric function

$$F(a,b,c;z) = 1 + \frac{ab}{c}\frac{z}{1!} + \frac{a(a+1)b(b+1)}{c(c+1)}\frac{z^2}{2!} + \frac{a(a+1)(a+2)b(b+1)(b+2)}{c(c+1)(c+2)}\frac{z^3}{3!} + \ldots, \tag{3.1.13}$$

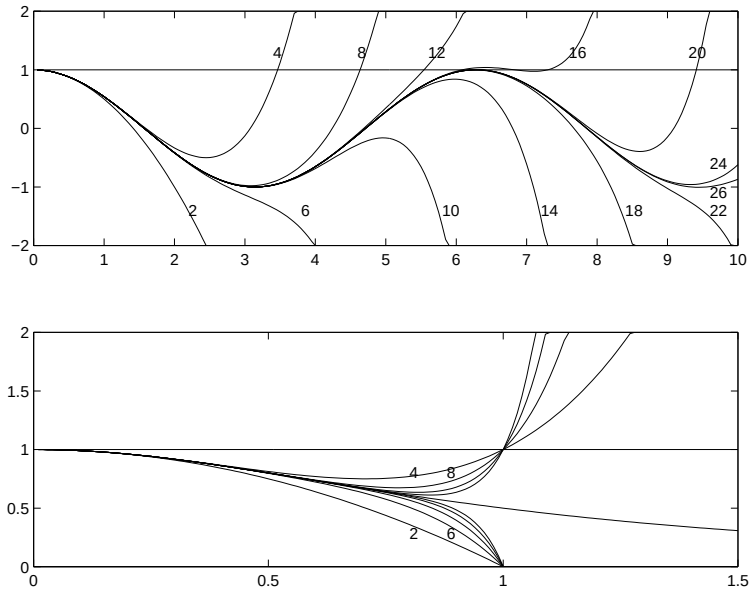


Figure 3.1.4. *The partial sums of the Maclaurin expansions for two functions. The upper curves are for* $f(x) = \cos x$, $n = 0 : 2 : 26$, $0 \le x \le 10$. *This series converges for all* x, *but the rounding errors cause trouble for large values of* x; *see Sec. 3.2.5, Ill-conditioned series. The lower curves are for* $f(x) = 1/(1+x^2)$, $n = 0 : 2 : 18$, $0 \le x \le 1.5$. *The convergence radius is* 1 *in this case.*

where a and b are complex constants and $c \neq -1, -2 \ldots$. The radius of convergence for this series equals unity. (see [1, Chap. 15])

Kummer's confluent hypergeometric function[6]

$$M(a,b;z) = 1 + \frac{a}{b}\frac{z}{1!} + \frac{a(a+1)}{b(b+1)}\frac{z^2}{2!} + \frac{a(a+1)(a+2)}{b(b+1)(b+2)}\frac{z^3}{3!} + \ldots, \tag{3.1.14}$$

converges for all z (see [1, Ch. 13]). It is named "confluent" because

$$M(a,c;z) = \lim_{b\to\infty} F(a,b,c,z/b).$$

The coefficients of these series are easily computed and the functions are easily evaluated by recurrence relations. (You also need some criterion for the truncation of the series, adapted to your demands of accuracy.) In Sec. 3.5, these functions are also expressed in terms of infinite *continued fractions* that typically converge faster and in larger regions than the power series do.

[6] Ernst Eduard Kummer (1810–1893) German mathematician, professor in Berlin from 1855. He extended Gauss work on hypergeometric series. He, together with Weierstrass and Kronecker, made Berlin into one of the leading centers of mathematics.

Example 3.1.5.

The following procedure can generally be used in order to find the *expansion of the quotient of two expansions*. We illustrate it in a case, where the result is of interest to us later.

The **Bernoulli**[7] **numbers** B_n are defined by the Maclaurin series

$$\frac{x}{e^x-1} \equiv \sum_{j=0}^{\infty} \frac{B_j x^j}{j!} \tag{3.1.15}$$

For $x = 0$ the left hand side is defined by Hôpital's rule; the value is 1. If we multiply this equation by the denominator, we obtain

$$x \equiv \Big(\sum_{i=1}^{\infty} \frac{x^i}{i!}\Big)\Big(\sum_{j=0}^{\infty} \frac{B_j x^j}{j!}\Big).$$

By matching the coefficients of x^n, $n \geq 1$, on both sides, we obtain a recurrence relation for the Bernoulli numbers, which can be written in the form

$$B_0 = 1, \quad \sum_{j=0}^{n-1} \frac{1}{(n-j)!}\frac{B_j}{j!} = 0, \quad n \geq 2, \quad \text{i.e.} \sum_{j=0}^{n-1} \binom{n}{j} B_j = 0. \tag{3.1.16}$$

The last equation is a recurrence that determines B_{n-1} in terms of Bernoulli numbers with smaller subscripts, hence $B_0 = 1$, $B_1 = -\frac{1}{2}$, $B_2 = \frac{1}{6}$, $B_3 = 0$, $B_4 = -\frac{1}{30}$, $B_5 = 0$, $B_6 = \frac{1}{42}, \ldots$.

We see that the Bernoulli numbers are rational. We shall now demonstrate that $B_n = 0$, *when n is odd, except for $n = 1$.*

$$\frac{x}{e^x-1} + \frac{x}{2} = \frac{x}{2}\frac{e^x+1}{e^x-1} = \frac{x}{2}\frac{e^{x/2}+e^{-x/2}}{e^{x/2}-e^{-x/2}} = \sum_{n=0}^{\infty} \frac{B_{2n}x^{2n}}{(2n)!}. \tag{3.1.17}$$

Since the next to last term is an even function, i.e. its value is unchanged when x is replaced by $-x$, its Maclaurin expansion contains only even powers of x, and hence the last expansion is also true.

The recurrence obtained for the Bernoulli numbers by the matching of coefficients in the equation,

$$(e^{x/2}-e^{-x/2})\Big(\sum_{n=0}^{\infty} B_{2n}x^{2n}/(2n)!\Big) = \tfrac{1}{2}x\left(e^{x/2}+e^{-x/2}\right),$$

is not the same as the one we found above. It turns out to have better properties of numerical stability. We shall look into this experimentally in Problem 10(g).

[7] Jacob (or James) Bernoulli (1654-1705), Swiss mathematician, one of the earliest to realize how powerful is the infinitesimal calculus. The Bernoulli numbers were published posthumously in 1713, in his fundamental work Ars Conjectandi (on Probability). The notation for Bernoulli numbers varies in the literature. Our notation seems to be the most common in modern texts. Several members of the same family enriched mathematics by their teaching and writing. Their role in the history of mathematics resembles the role of the Bach family in the history of music.

The singularities of the function $x/(e^x - 1)$ are poles at $x = 2n\pi i$, $n = \pm1, \pm2, \pm3, \ldots$, hence the radius of convergence is 2π. Further properties of Bernoulli numbers and the related Bernoulli polynomials and periodic functions, are presented in Sec. 3.4.4, where they occur as coefficients in the important Euler–Maclaurin formula.

If r is large the following formula is very efficient; the series on its right hand side then converges rapidly.

$$B_{2r}/(2r)! = (-1)^{r-1}2(2\pi)^{-2r}\Big(1 + \sum_{n=2}^{\infty} n^{-2r}\Big). \tag{3.1.18}$$

This is a particular case ($t = 0$) of a Fourier series for the Bernoulli functions that we shall encounter in Lemma 3.4.2(c). In fact, you obtain IEEE double accuracy for $r > 26$, even if the infinite sum on the right hand side is totally ignored. Thanks to (3.1.18) we do not need to worry much over the instability of the recurrences. When r is *very* large, however, we must be careful about underflow and overflow.

The **Euler numbers** E_n, which will be used later, are similarly defined by the generating function

$$\frac{1}{\cosh z} \equiv \sum_{n=0}^{\infty} \frac{E_n z^n}{n!}, \quad |z| < \frac{\pi}{2}. \tag{3.1.19}$$

Obviously $E_n = 0$ for all odd n. It can be shown that the Euler numbers are integers, $E_0 = 1$, $E_2 = -1$, $E_4 = 5$, $E_6 = -61$; see Problem 7e.

Example 3.1.6.

Let $f(x) = (x^3+1)^{-\frac{1}{2}}$. Compute $\int_{10}^{\infty} f(x)dx$ to 9 decimal places, and $f'''(10)$, with at most 1% error. Since x^{-1} is fairly small, we expand in powers of x^{-1}:

$$\begin{aligned} f(x) &= x^{-3/2}(1 + x^{-3})^{-1/2} = x^{-3/2}\Big(1 - \frac{1}{2}x^{-3} + \frac{1\cdot3}{8}x^{-6} - \ldots\Big) \\ &= x^{-1.5} - \frac{1}{2}x^{-4.5} + \frac{3}{8}x^{-7.5} - \ldots. \end{aligned}$$

By integration,

$$\int_{10}^{\infty} f(x)dx = 2\cdot10^{-0.5} - \frac{1}{7}10^{-3.5} + \frac{3}{52}10^{-6.5} + \ldots = 0.632410375.$$

Each term is less than 0.001 of the previous term.

By differentiating the series three times, we similarly obtain

$$f'''(x) = -\frac{105}{8}x^{-4.5} + \frac{1,287}{16}x^{-7.5} + \ldots.$$

For $x = 10$ the second term is less than 1% of the first; the terms after the second decrease quickly and are negligible. One can show that the magnitude of each term is less than $8\,x^{-3}$ of the previous term. We get $f'''(10) = -4.12\,10^{-4}$ to the desired accuracy. The reader is advised to carry through the calculation in more detail.

Example 3.1.7. *How to compute* $\sinh x$.

On a computer using IEEE double precision the roundoff unit is $u = 2^{-53} \approx 1.1 \cdot 10^{-16}$. One wishes to compute $\sinh x$ with good *relative* accuracy, both for small and large $|x|$, at least moderately large. Assume that e^x is computed with a relative error less than u in the given interval. The formula $(e^x - e^{-x})/2$ for $\sinh x$ is sufficiently accurate except when $|x|$ is very small and cancellation occurs. Hence for $|x| \ll 1$, e^x and e^{-x} and hence $(e^x - e^{-x})/2$ can have *absolute* errors of order of magnitude (say) u. Then the *relative* error in $(e^x - e^{-x})/2$ can have magnitude $\approx u/|x|$; for example, this is more than 100% for $x \approx 10^{-16}$.

For $|x| \ll 1$ one can instead use (say) two terms in the series expansion for $\sinh x$,

$$\sinh x = x + x^3/3! + x^5/5! + \dots.$$

Then one gets an absolute truncation error which is about $x^5/120$, and a round-off error of the order of $2u|x|$. Thus the formula $x + x^3/6$ is better than $(e^x - e^{-x})/2$ if

$$|x|^5/120 + 2u|x| < u.$$

If $2u|x| \ll u$, we have $|x|^5 < 120u \approx 15 \cdot 2^{-50}$, or $|x| < 15^{1/5} \cdot 2^{-10} \approx 0.00168$, (which shows that $2u|x|$ really could be ignored in this rough calculation). Thus, if one switches from $(e^x - e^{-x})/2$ to $x + x^3/6$ for $|x| < 0.00168$, the relative error will nowhere exceed $u/0.00168 \approx 0.66 \cdot 10^{-14}$. If one needs higher accuracy, one can take more terms in the series, so that the switch can occur at a larger value of $|x|$.

For very large values of $|x|$ one must expect a relative error of order of magnitude $|xu|$ because of round-off error in the argument x. Compare the discussion of range reduction in Sec. 2.2.4 and Problem 2.2.9.

In numerical computation a series should be regarded as a finite expansion together with a remainder. Taylor's formula with the remainder (3.1.5) is valid for any function $f \in C^n[a, a+x]$, but *the infinite series is valid only if the function is analytic in a complex neighborhood of* a.

If a function is not analytic at 0*, it can happen that the Maclaurin expansion converges to a wrong result.* A classical example (see Appendix to Chapter 6 in Courant [15]) is

$$f(x) = e^{-1/x^2}, \quad x \neq 0, \quad f(0) = 0.$$

It can be shown that all its Maclaurin coefficients are zero. This trivial Maclaurin expansion converges for all x, *but the sum is wrong for* $x \neq 0$. There is nothing wrong with the use of Maclaurin's formula as a finite expansion with a remainder. Although the remainder that in this case equals $f(x)$ itself, does not tend to 0 *as* $n \to \infty$ *for a fixed* $x \neq 0$, it tends to 0 faster than any power of x, *as* $x \to 0$*, for any fixed* n. The "expansion" gives, e.g., an *absolute* error less than 10^{-43} for $x = 0.1$, but the *relative* error is 100%. Also note that this function (and there are lots of other examples) can be added to any function without changing its Maclaurin expansion.

From the point of view of complex analysis, however, the origin is a singular point for this function, note, e.g., that $|f(z)| \to \infty$ as $z \to 0$ along the imaginary

axis, and this prevents the application of any theorem that would guarantee that the infinite Maclaurin series represents the function. This trouble does not occur for a truncated Maclaurin expansion around a point, where the function under consideration is analytic. The size of the first non-vanishing neglected term then gives a good hint about the truncation error, when $|z|$ is a small fraction of the radius of convergence.

The above example may sound like a purely theoretical matter of curiosity. We emphasize this *distinction between the convergence and the validity of an infinite expansion* in this text, as a background to other expansions of importance in numerical computation, e.g., the Euler–Maclaurin expansion in Sec. 3.4.4, which may converge to the wrong result, also in the application to a well-behaved analytic function. On the other hand, we shall see, e.g., in Sec. 3.1.8, that divergent expansions can sometimes be very useful. The universal recipe *in numerical computation* is to consider an infinite series as a finite expansion plus a remainder term. A more algebraic point of view on a series is, however, often useful *in the design of a numerical method*. See, e.g., Sec. 3.1.5 (Formal Power Series) and Sec. 3.3.2 (The Calculus of Operators). Convergence of an expansion is neither necessary nor sufficient for its success in practical computation.

3.1.3 Analytic Continuation

Analytic functions have many important properties that you may find in any text on complex analysis. A good summary for the purpose of numerical mathematics is found in the first chapter of Stenger [43]. Two important properties are contained in the following lemma.

We remark that the region of analyticity of a function $f(z)$ is an *open* set. If, e.g., we say that $f(z)$ is analytic on a closed real interval, it means that there exists an open set in $\mathbf{C}$ that contains this interval, where $f(z)$ is analytic.

Lemma 3.1.7.

An analytic function can only have a finite number of zeros in a compact subset of the region of analyticity, unless the function is identically zero.

Suppose that two functions f_1 and f_2 are analytic in regions D_1 and D_2, respectively. Suppose that $D_1 \cap D_2$ contains an interval throughout which $f_1(z) = f_2(z)$. Then $f_1(z) = f_2(z)$ in the intersection $D_1 \cap D_2$.

Proof. We refer, for the first part, to any text on Complex Analysis. We here follow Titchmarsh [45] closely. The second part follows by the application of the first part to the function $f_1 - f_2$. □

A consequence of this is known as *the permanence of functional equations*, i.e. in order to prove the validity of a functional equation (or "a formula for a function") in a region of the complex plane, it may be sufficient to prove its validity in (say) an interval of the real axis, under the conditions specified in the lemma.

Example 3.1.8. *The permanence of functional equations.*

We know from elementary real analysis that the functional equation

$$e^{(p+q)z} = e^{pz}e^{qz}, \quad (p, q \in \mathbf{R}),$$

holds for all $z \in \mathbf{R}$. We also know that all the three functions involved are analytic for all $z \in \mathbf{C}$. Set in the lemma $D_1 = D_2 = \mathbf{C}$, and let "the interval" be any compact interval of $\mathbf{R}$. The lemma then tells us that that the displayed equation holds for all complex z.

The right and the left hand side then have identical power series. Applying the convolution formula and matching the coefficients of z^n, we obtain

$$\frac{(p+q)^n}{n!} = \sum_{j=0}^{n} \frac{p^j}{j!}\frac{q^{n-j}}{(n-j)!}, \quad \text{i.e.,} \quad (p+q)^n = \sum_{j=0}^{n} \frac{n!}{j!(n-j)!}p^j q^{n-j}.$$

This is not a very sensational result. It is more interesting to start from the following functional equation

$$(1+z)^{p+q} = (1+z)^p(1+z)^q.$$

The same argumentation holds, except that—by the discussion around Table 3.1.1—D_1, D_2 should be equal to the complex plane with a cut from -1 to $-\infty$, and that the Maclaurin series is convergent in the unit disk only.

We obtain the equations

$$\binom{p+q}{n} = \sum_{j=0}^{n} \binom{p}{j}\binom{q}{n-j}, \quad n = 0, 1, 2, \ldots. \tag{3.1.20}$$

(They can also be proved by induction, but it is not needed.) This sequence of algebraic identities, where *each identity contains a finite number of terms*, is equivalent to the above functional equation.

We shall see that this observation is useful for motivating certain "*symbolic computations*" with power series, that can provide elegant derivations of useful formulas in numerical mathematics.

Now we may consider the aggregate of values of $f_1(z)$ and $f_2(z)$ at points interior to D_1 or D_2 as a single analytic function f. Thus f is analytic in the union $D_1 \cup D_2$, and $f(z) = f_1(z)$ in D_1, $f(z) = f_2(z)$ in D_2.

The function f_2 may be considered as extending the domain in which f_1 is defined, and it is called a (single-valued) **analytic continuation** of f_1. In the same way f_1 is an analytic continuation of f_2. Analytic continuation denotes both this process of extending the definition of a given function, and the result of the process. We shall see examples of this, e.g., in Sec. 3.1.3. Under certain conditions the analytic continuation is unique.

Theorem 3.1.8.

Suppose that a region D is overlapped by regions D_1, D_2, and that $(D_1 \cap D_2) \cap D$ contains an interval. Let f be analytic in D, and let f_1 be an analytic continuation of f to D_1, and let f_2 an analytic continuation of f to D_2, so that

$$f(z) = f_1(z) = f_2(z) \quad \text{in} \quad (D_1 \cap D_2) \cap D.$$

Then either of these functions provides a single-valued analytic continuation of f to $D_1 \cap D_2$. The results of the two processes are the same.

Proof. Since $f_1 - f_2$ is analytic in $D_1 \cap D_2$, and $f_1 - f_2 = 0$ in the set $(D_1 \cap D_2) \cap D$, which contains an interval, it follows from the lemma that $f_1(z) = f_2(z)$ in $D_1 \cap D_2$, which proves the theorem. □

If the set $(D_1 \cap D_2) \cap D$ is *void*, the conclusion in the theorem *may not be valid*. We may still consider the aggregate of values as a single analytic function, but *this function can be multi-valued in* $D_1 \cap D_2$.

Example 3.1.9.
For $|x| < 1$ the important formula

$$\arctan x = \frac{1}{2i} \ln \left(\frac{1 + ix}{1 - ix} \right)$$

easily follows from the expansions in the Table 3.1.1. The function $\arctan x$ has an analytic continuation as single-valued functions in the complex plane with cuts along the imaginary axis from i to ∞ and from $-i$ to $-\infty$. It follows from the theorem that "the important formula" is valid in this set.

3.1.4 Manipulating Power Series

In some contexts, algebraic recurrence relations can be used for the computation of the coefficients in Maclaurin expansions, in particular if only a moderate number of coefficients are wanted. We shall study a few examples.

Example 3.1.10. *Expansion of a composite function.*
Let $g(x) = b_0 + b_1 x + b_2 x^2 + \ldots$, $f(z) = a_0 + a_1 z + a_2 z^2 + \ldots$, be given functions, analytic at the origin. Find the power series

$$h(x) = f(g(x)) \equiv c_0 + c_1 x + c_2 x^2 + \ldots .$$

In particular, we shall study the case $f(z) = e^z$.

The first idea we may think of is to substitute the expansion $b_0 + b_1 x + b_2 x^2 + \ldots$ for z into the power series for $f(z)$. This is, however, *no good unless* $g(0) = b_0 = 0$, because

$$(g(x))^k = b_0^k + k b_0^{k-1} b_1 x + \ldots$$

gives a contribution to, e.g., c_0, c_1, for every k, so we cannot successively compute the c_j by *finite* computation.

Now suppose that $b_0 = 0$, $b_1 = 1$, i.e. $g(x) = x + b_2 x^2 + b_3 x^3 + \ldots$. (The assumption that $b_1 = 1$ is not important, but it simplifies the writing.) Then c_j depends only on b_k, a_k, $k \leq j$, since $(g(x))^k = x^k + k b_2 x^{k+1} + \ldots$. We obtain

$$h(x) = a_0 + a_1 x + (a_1 b_2 + a_2) x^2 + (a_1 b_3 + 2 a_2 b_2 + a_3) x^3 + \ldots ,$$

and the coefficients of $h(x)$ come out recursively,

$$c_0 = a_0; \quad c_1 = a_1, \quad c_2 = a_1b_2 + a_2, \quad c_3 = a_1b_3 + 2a_2b_2 + a_3, \ldots.$$

Now consider the case $f(z) = e^z$, i.e. $a_n = 1/n!$. We first see that it is then also easy to handle the case that $b_0 \neq 0$, since

$$e^{g(x)} = e^{b_0} e^{b_1 x + b_2 x^2 + b_3 x^3 + \cdots}.$$

But there exists a more important simplification if $f(z) = e^z$. Note that h satisfies the differential equation $h'(x) = g'(x)h(x)$, $h(0) = e^{b_0}$. Hence

$$\sum_{n=0}^{\infty}(n+1)c_{n+1}x^n \equiv \sum_{j=0}^{\infty}(j+1)b_{j+1}x^j \sum_{k=0}^{\infty} c_k x^k.$$

Set $c_0 = e^{b_0}$, apply the convolution formula (3.1.8), and match the coefficients of x^n on the two sides:

$$(n+1)c_{n+1} = b_1c_n + 2b_2c_{n-1} + \ldots + (n+1)b_{n+1}c_0, \quad (n = 0, 1, 2, \ldots).$$

This recurrence relation is more easily programmed than the general procedure indicated above. Other functions that satisfy appropriate differential equations can be treated similarly; see Problem 8. More information is found in Knuth [29, Sec. 4.7].

Formulas like these are often used in packages for **symbolic differentiation** and for **automatic** or **algorithmic differentiation**. Expanding a function into a Taylor series is equivalent to finding the sequence of derivatives of the function at a given point. The goal of *symbolic* differentiation is to obtain analytic *expressions* for derivatives of functions given in analytic form. This is handled by computer algebra systems, e.g., Maple or Mathematica.

In contrast, the goal of **automatic** or **algorithmic differentiation** is to extend an algorithm (a program) for the computation of the *numerical values* of a few functions to an algorithm that also computes *the numerical values* of a few derivatives of these functions, without truncation errors. A simple example, Horner's scheme for computing values and derivatives for a polynomial was given in Sec. 1.3.1. At the time of writing, there is a lively activity about automatic differentiation—theory, software development and applications. Typical applications are in the solution of ordinary differential equations by Taylor expansion; see Example 3.1.1. Such techniques are also used in optimization for partial derivatives of low order, e.g., for the computation of Jacobian and Hessian matrices.

Sometimes power series are needed with many terms, although rarely more than 30 (say). (The ill-conditioned series are exceptions; see Sec. 3.2.5.) The determination of the coefficients can be achieved by the **Toeplitz matrix method** using floating point computation and an interactive matrix language. Computational details will be given in Problems 9–12 of this section for MATLAB . (Systems like Maple and Mathematica that include exact arithmetic and other features, are

evidently also useful here.) An alternative method, the **Cauchy–FFT method**, will be described in Sec. 3.2.2.

Both methods will be applied later in the book. See in particular Sec. 3.3.4, where they are used for deriving approximation formulas in the form of *expansions in powers of elementary difference or differential operators*. In such applications, the coefficient vector, v (say), is obtained in floating point (usually in a very short time). Very accurate rational approximations to v, often even the exact values, can be obtained (again in a very short time) by applying MATLAB function $[N, D] =$ `rat`(z, Tol) to the results, with two different values of the tolerance. This function is based on a continued fraction algorithm, given in Example 3.5.2 for finding the best rational approximation to a real number. This can be used for the *"cleaning"* of numerical results which have, for practical reasons, been computed by floating point arithmetic, although the exact results are known to be (or strongly believed to be) rather simple rational numbers. The algorithm attempts to remove the *"dirt"* caused by computational errors. In Sec. 3.5.1 you also find some comments of importance for the interpretation of the results, e.g., for judging whether the rational numbers are exact results or good approximations only.

Let

$$f(z) = \sum_{j=0}^{\infty} a_j z^j$$

be the power series of a function analytic at $z = 0$. With this power series we can associate an infinite upper triangular **semicirculant matrix**

$$C_f = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 & \dots \\ & a_0 & a_1 & a_2 & \dots \\ & & a_0 & a_1 & \dots \\ & & & a_0 & \dots \\ & & & & \ddots \end{pmatrix}, \tag{3.1.21}$$

Similarly, a truncated power series $f_N(z) = \sum_{j=0}^{N-1} a_j z^j$ is represented by the finite leading principal $N \times N$ submatrix of C_f (see Definition A.3.1), which can be written as

$$f_N(S_N) = \sum_{j=0}^{N-1} a_j S_N^j, \tag{3.1.22}$$

where S_N is a **shift matrix**. For example, with $N = 4$,

$$f_N(S_N) = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ 0 & a_0 & a_1 & a_2 \\ 0 & 0 & a_0 & a_1 \\ 0 & 0 & 0 & a_0 \end{pmatrix}, \qquad S_N = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The following properties of S_N explains the term "shift matrix":

$$S_N \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} x_2 \\ x_3 \\ x_4 \\ 0 \end{pmatrix}, \qquad (x_1, x_2, x_3, x_4) S_N = (0, x_1, x_2, x_3).$$

An $N \times N$ matrix, with a structure analogous to $f_4(S_4)$, is known as an upper triangular **Toeplitz matrix**.[8]. Matrices (not necessarily triangular), whose entries are constant along each diagonal, are called *Toeplitz matrices*.

What do the powers of S_N look like? Note that $S_N^N = 0$, i.e. S_N is a **nilpotent** matrix. This is one of the reasons why the Toeplitz matrix representation is convenient for work with truncated power series, since it follows that

$$f(S_N) = \sum_{j=0}^{\infty} a_j S_N^j = \sum_{j=0}^{N-1} a_j S_N^j = f_N(S_N).$$

It is easily verified that a **product** of upper triangular Toeplitz matrices is of the same type. Also note that the multiplication of such matrices is *commutative*. It is also evident that a **linear combination** of such matrices is of the same type. Further it holds that

$$\begin{aligned}(f \cdot g)(S_N) &= f(S_N)g(S_N) = f_N(S_N)g_N(S_N);\\ (\alpha f + \beta g)(S_N) &= \alpha f_N(S_N) + \beta g_N(S_N).\end{aligned}$$

(In general, Toeplitz matrices are not nilpotent, and the product of two *non-triangular* Toeplitz matrices is not a Toeplitz matrix. Similarly for the inverse. In this section we shall only deal with upper triangular Toeplitz matrices.)

Denote by e_1^T the first row of the unit matrix of a size appropriate in the context. An upper triangular Toeplitz matrix of order N is uniquely *determined by its first row* r by means of a simple and fast algorithm that we call t*oep*$\,(r, N)$. For example, the unit matrix of order N is $I_N =$ t*oep*$\,(e_1^T, N)$, and the shift matrix is $S_N =$ t*oep*$\,([0\ e_1^T], N)$. A MATLAB implementation is given in Problem 9.

Now it will be indicated how one can save CPU time and memory space by working on the row vector level, with the first rows instead of with the full triangular matrices.[9] We shall *denote by* $f1$, $g1$, *the row vectors with the first* N *coefficients of the Maclaurin expansions of* $f(z)$, $g(z)$. They are equal to the first rows of the matrices $f(S_N)$, $g(S_N)$, respectively. Suppose that $f1$, $g1$ are given and we shall compute $f \cdot g1$, i.e. the first row of $f(S_N) \cdot g(S_N)$ in a similar notation. Then

$$f \cdot g1 = e_1^T(f(S_N) \cdot g(S_N)) = (e_1^T f(S_N)) \cdot g(S_N) = f1 \cdot \mathrm{toep}(g_1, N). \qquad (3.1.23)$$

Note that you never have to multiply two triangular matrices, if you work with the first rows only. So, only about $N^2/2$ flops and (typically) an application of the toep(r, N) algorithm, are needed instead of about $N^3/6$ if two upper triangular matrices are multiplied; see Sec. 1.4.1, where the operation count for matrix multiplication is discussed.

Similarly the **quotient** *of two upper triangular Toeplitz matrices*, (say)

$$Q(S_N) = f(S_N) \cdot g(S_N)^{-1},$$

[8] Otto Toeplitz (1881-1940), German mathematician

[9] In interactive computations with rather short series the gain of time may sometimes be neutralized by an increased number of manual operations. See the computer exercises.

is also a matrix of the same type. (A hint to a proof is given in Problem 9.[10]) Note that $Q(S_N) \cdot g(S_N) = f(S_N)$. With similar notations as above, we obtain for the first row of this matrix equation the following triangular linear system where the *row* vector $q1$ is the unknown.

$$q1 \cdot \mathrm{toep}(g1, N) = f1. \tag{3.1.24}$$

Although the discussion in Sec. 1.3.4 is concerned with a linear system with a *column* as the unknown (instead of a row), we draw from it the conclusion that only about $N^2/2$ scalar flops (including N scalar divisions) and one application of the toep algorithm, are needed, instead of the $N^3/6$ needed in the solution of the matrix equation $Q \cdot g(S_N) = f(S_N)$.[11]

A library called toeplib consists of short MATLAB scripts mainly based on Table 3.1.2. It is given in Problem 10 (a). In Problem 10 (b), etc., the series of the library are combined by elementary operations to become interesting examples of the Toeplitz matrix method. The convenience, the accuracy and the execution time are probably much better than you expect; even the authors were surprised.

Next we shall study how a **composite function** $h(z) = f(g(z))$ can be expanded in powers of z. *Suppose that* $f(z)$ and $g(z)$ are analytic at $z = 0$, $f(z) = \sum_{j=1}^{\infty} f1(j) z^{j-1}$. An **important assumption** is that $g(0) = 0$. Then we can set $g(z) = z\bar{g}(z)$, hence $(g(z))^n = z^n(\bar{g}(z))^n$ and, because $S_N^n = 0$, $n \geq N$, we obtain

$$(g(S_N))^n = S_N^n \cdot (\bar{g}(S_N))^n = 0, \quad \text{if } n \geq N \text{ and } g(0) = 0,$$

$$h(S_N) \equiv f(g(S_N)) = \sum_{j=1}^{N} f1(j)(g(S_N))^{j-1}, \quad \text{if } g(0) = 0. \tag{3.1.25}$$

This matrix polynomial can be computed by a matrix version of Horner's scheme. The row vector version of this equation is written

$$h1 = \mathrm{comp}(f1, g1, N). \tag{3.1.26}$$

A MATLAB implementation of the function comp is listed and applied in Problem 11.

If $g(0) \neq 0$, Equation (3.1.25) still provides an "expansion", but it is **wrong**; see Problem 11 (c). Suppose that $|g(0)|$ is less than the radius of convergence of the Maclaurin expansion of $f(x)$. Then a correct expansion is obtained by a different decomposition. Set $\tilde{g}(z) = g(z) - g(0)$, $\tilde{f}(x) = f(x + g(0))$. Then $\tilde{f}$, $\tilde{g}$ are analytic

[10] In the terminology of algebra, the set of upper triangular $N \times N$ Toeplitz matrices, i.e. $\{\sum_{j=0}^{N-1} \alpha_j S_N^j\}$, $\alpha_j \in \mathbf{C}$, is a **commutative integral domain** that is isomorphic with the set of polynomials $\sum_{j=0}^{N-1} \alpha_j x^j$ modulo x^N, where x is an indeterminate.

[11] The equations (3.1.23) and (3.1.24) are mathematically equivalent to the convolution product in (3.1.8) and the procedure demonstrated in Example 3.1.6, respectively. Sometimes both procedures suffer from the growth of the effects of rounding errors when n is very large, in particular when the power series are ill- conditioned; see Sec. 3.1.11. An advantage of the Toeplitz matrix method is that the coding, in a language with convenient matrix handling, becomes easier.

at $z = 0$. $\tilde{g}(0) = 0$ and $\tilde{f}(\tilde{g}(z)) = f(g(z)) = h(z)$. So, (3.1.25) and its row vector implementations can be used if $\tilde{f}, \tilde{g}$ are substituted for f, g.

Analytic functions of matrices are defined, in terms of their Taylor series; see Sec. 9.2.5 in Vol. II. For example, the series

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^2}{2!} + \cdots,$$

converges elementwise for any matrix A. There exist several algorithms for computing e^A, $\sqrt{A}$, $\log A$, where A is a square matrix. One can make linear combinations, products, quotients and composite functions of them. For example, a "principal matrix value" of $Y = (I + A)^\alpha$ is obtained by

$$B = \log(I + A), \quad Y = e^{\alpha B}.$$

For a composite function $f(g(A))$, it is here *not* necessary that $g(0) = 0$, but it is *important* that $g(z)$ and $f(g(z))$ are analytic when z is an eigenvalue of A. We obtain *truncated power series if* $A = S_N$; note that S_N has a multiple eigenvalue at 0. The coding, and the manual handling in interactive computing, are convenient with matrix functions, but the computer has to perform more operations on full triangular matrices than with the row vector level algorithms described above. So, for *very* long expansions the earlier algorithms are notably faster.

If the given power series, $f(x)$, $g(x), \ldots$ have *rational coefficients*, then the exact results of a sequence of additions, multiplications, divisions, compositions, differentiations, integrations will have rational coefficients, because the algorithms are all formed by a finite number of scalar additions, multiplications and divisions. As mentioned above, very accurate rational approximations, often even the exact values, can be quickly obtained by applying a continued fraction algorithm that is presented in Sec. 3.5.2 to the results of a floating point computation. See also Problems 9–12.

If $f(x)$ is an even function, its power series contains only even powers of x. You gain space and time, by letting the shift matrix S_N correspond to x^2 (instead of x). Similarly, if $f(x)$ is an odd function, you can instead work with the even function $f(x)/x$, and let S_N correspond to x^2. See Problems 9–12.

Finally we consider a classical problem of mathematics, known as **power series reversion**. The task is to find the power series for the **inverse function** $x = g(y)$ of the function $y = f(x) = \sum_{j=0}^{\infty} a_j x^j$, in the particular case where $a_0 = 0$, $a_1 = 1$. Note that even if the series for $f(x)$ is finite, the series for $g(y)$ is in general infinite!

The following simple cases of power series reversion are often sufficient and useful in low order computations with paper and pencil.

$$\begin{aligned} & y = x + ax^k + \ldots, \quad (k > 1), \\ \Rightarrow\ & x = y - ax^k - \ldots = y - ay^k - \ldots; \qquad (3.1.27) \\ & y = f(x) \equiv x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots, \\ \Rightarrow\ & x = g(y) \equiv y - a_2y^2 + (2a_2^2 - a_3)y^3 - (5a_2^2 - 5a_2a_3 + a_4)y^4 + \ldots. \qquad (3.1.28) \end{aligned}$$

An application of power series reversion occurs in the derivation of a family of iterative methods of arbitrary high order for solving scalar non-linear equations; see Sec. 6.3.5.

Knuth [29] devotes Sec. 4.7 to the manipulation of power series. He presents several algorithms for power series reversion, e.g., a classical algorithm due to Lagrange 1768 that requires $O(N^3)$ operations to compute the first N terms. Knuth also includes a recent algorithm due to Brent and Kung [7]. It is based on an adaptation, to formal power series, of Newton's method (1.2.3) for solving a numerical algebraic equation. For power series reversion, the equation to be solved reads

$$f(g(y)) = y, \tag{3.1.29}$$

where the coefficients of g are the unknowns. The number of correct terms is roughly doubled in each iteration, as long as N is not exceeded. In the usual numerical application of Newton's method to a scalar non-linear equation (see Secs. 1.2 and 6.3) it is the number of significant digits that is (approximately) doubled, so-called quadratic convergence. Brent–Kung's algorithm can be implemented in about $150\,(N \log N)^{3/2}$ scalar flops.

In Problem 12, a convenient Toeplitz matrix implementation of the idea of Brent and Kung is presented. It requires about $cN^3 \log N$ scalar flops with a moderate value of c. It is thus much inferior to the original algorithm if N is very large. In some interesting interactive applications, however, N rarely exceeds 30. In such cases our implementation is satisfactory, unless (say) hundreds of series are to be reversed.

3.1.5 Formal Power Series

A power series is not only a means for numerical computation; it is also an aid for deriving formulas in numerical mathematics and in other branches of applied mathematics. Then one has another, more algebraic, aspect of power series that we shall briefly introduce. A more rigorous and detailed treatment is found in Henrici [25, Chapter 1], and in the literature quoted there.

In a **formal power series**, $\mathbf{P} = a_0 + a_1\mathbf{x} + a_2\mathbf{x}^2 + \cdots$, the coefficients a_j may be real or complex numbers (or elements in some other field), while $\mathbf{x}$ is an algebraic **indeterminate**; $\mathbf{x}$ and its powers can be viewed as place keepers. No real or complex values are assigned to $\mathbf{x}$ and $\mathbf{P}$. Convergence, divergence and remainder term have no relevance for formal power series. The coefficients of a formal power series may even be such that the series diverges for any non-zero complex value that you substitute for the indeterminate, e.g. the series

$$\mathbf{P} = 0! - 1!\mathbf{x} + 2!\mathbf{x}^2 - 3!\mathbf{x}^3 + \cdots. \tag{3.1.30}$$

In algebraic terminology, the set of formal power series is an integral domain. The sum of $\mathbf{P}$ and another formal power series, $\mathbf{Q} = b_0 + b_1\mathbf{x} + b_2\mathbf{x}^2 + \cdots$, is *defined* as

$$\mathbf{P} + \mathbf{Q} = (a_0 + b_0) + (a_1 + b_1)\mathbf{x} + (a_2 + b_2)\mathbf{x}^2 + \cdots.$$

Similarly, the *Cauchy product* is *defined* as

$$\mathbf{PQ} = c_0 + c_1\mathbf{x} + c_2\mathbf{x}^2 + \cdots,$$

where the coefficients are given by the convolution formula (3.1.8). The division of two formal power series is defined by a recurrence, as indicated in Example 3.1.5, iff the first coefficient of the the denominator is not zero.

Other operations are defined without surprises, e.g., the derivative of $\mathbf{P}$ is *defined* as $\mathbf{P}' = 1a_1 + 2a_2\mathbf{x} + 3a_3\mathbf{x}^2 + \ldots$. The limit process, by which the derivative is defined in Calculus, does not exist for formal power series. The usual rules for differentiation are still valid, and as an exercise you may verify that the formal power series defined by (3.1.30) satisfies the formal differential equation $\mathbf{x}^2\mathbf{P}' = \mathbf{x} - \mathbf{P}$. See also Sec. 3.1.11.

Formal power series can be used for deriving identities. In most applications in this book difference operators or differential operators are substituted for the indeterminates, and the identities are then used in the deriving of approximation formulas, e.g. for interpolation, numerical differentiation and integration etc.

The formal definitions of the Cauchy product, (i.e. convolution) and division are rarely used in practical calculation. It is easier to work with upper triangular $N \times N$ Toeplitz matrices, as in Sec. 3.1.5, where N is any natural number. Algebraic calculations with these matrices are isomorphic with calculations with formal power series modulo $\mathbf{x}^N$.

If you perform operations on matrices $f_M(S)$, $g_M(S), \ldots$, where $M < N$, the results are equal to the principal $M \times M$ submatrices of the results obtained with the matrices $f_N(S)$, $g_N(S), \ldots$. This fact follows directly from the equivalence with power series manipulations. It is also related to the fact that in the multiplication etc. of block upper triangular matrices, the diagonal blocks of the product equals the products of the diagonal blocks, and no new off-diagonal blocks enter.

So, we can easily *define the product of two infinite upper triangular matrices*, $C = AB$, by stating that if $i \le j \le n$ then c_{ij} has the same value that it has in the $N \times N$ submatrix $C_N = A_N B_N$ for every $N \ge n$. In particular C is upper triangular, and note that there are no conditions on the behaviour of the elements a_{ij}, b_{ij} as $i, j \to \infty$. One can show that this product is associative and distributive. For the infinite triangular Toeplitz matrices it is commutative too.[12]

Henrici [25, Sec. 1.3], calls this a representation of formal power series by *infinite* upper triangular Toeplitz matrices, (which he names *semicirculants*), and proves that *the mapping of the set of formal power series onto the set of semicirculants is an isomorphism.* If the formal power series $a_0 + a_1\mathbf{x} + a_2\mathbf{x}^2 + \cdots$, and its reciprocal series, which exists iff $a_0 \neq 0$, are represented by the semicirculants A and B, respectively, Henrici proves that $AB = BA = I$, where I is the unit matrix of infinite order. This indicates how to define the inverse of any infinite upper triangular matrix if all diagonal elements $a_{ii} \neq 0$.

If a function f of a complex variable z is analytic at the origin, then *we define*[13]

[12] For infinite *non-triangular* matrices the definition of a product generally contains conditions on the behaviour of the elements as $i, j \to \infty$, but we shall not discuss this here.

[13] Henrici, loc. cit., does not use this concept—it may not be established.

$f(\mathbf{x})$ as the formal power series with the same coefficients as the Maclaurin series for $f(z)$. In the case of a multivalued function we take the principal branch.

We do *not* consider formal power series with several indeterminates. There may occur expressions with several bold-type symbols. Only one of them is the indeterminate, and the others are shorthand notations for analytic functions of this indeterminate.

There is a kind of "permanence of functional equations" also for the generalization from a function $g(z)$ of a complex variable that is analytic at the origin, to the formal power series $g(\mathbf{x})$. We illustrate a *general principle* on an important special example that we formulate as a lemma, since we shall need it in the next section.

Lemma 3.1.9.

$$(e^{\mathbf{x}})^{\theta} = e^{\theta\mathbf{x}}, \quad (\theta \in \mathbf{R}). \tag{3.1.31}$$

Proof. Let the coefficient of $\mathbf{x}^j$ in the expansion of the left hand side be $\phi_j(\theta)$. The corresponding coefficient for the right hand side is $\theta^j/j!$. If we replace $\mathbf{x}$ by a complex variable z, the power series coefficients are the same, and we know that $(e^z)^{\theta} = e^{\theta z}$, hence $\phi_j(\theta) = \theta^j/j!$, $j = 1, 2, 3\ldots$, hence $\sum_0^\infty \phi_j(\theta)\mathbf{x}^j = \sum_0^\infty (\theta^j/j!)\mathbf{x}^j$, and the lemma follows. □

Example 3.1.11.

Find (if possible) a formal power series $\mathbf{Q} = 0 + b_1\mathbf{x} + b_2\mathbf{x}^2 + b_3\mathbf{x}^3 + \ldots$, that satisfies the equation

$$e^{-\mathbf{Q}} = 1 - \mathbf{x}, \tag{3.1.32}$$

where $e^{-\mathbf{Q}} = 1 - \mathbf{Q} + \mathbf{Q}^2/2! - \ldots$.

We can, in principle, determine an arbitrarily long sequence $b_1, b_2, b_3, \ldots b_k$ by matching the coefficients of $\mathbf{x}, \mathbf{x}^2, \mathbf{x}^3, \ldots \mathbf{x}^k$, in the two sides of the equation. We display the first three equations.

$$1 - (b_1\mathbf{x} + b_2\mathbf{x}^2 + b_3\mathbf{x}^3 + \ldots) + (b_1\mathbf{x} + b_2\mathbf{x}^2 + \ldots)^2/2 - (b_1\mathbf{x} + \ldots)^3/6 + \ldots$$

$$= 1 - 1\mathbf{x} + 0\mathbf{x}^2 + 0\mathbf{x}^3 + \ldots.$$

For any natural number k, the matching condition is of the form

$$-b_k + \phi_k(b_{k-1}, b_{k-2}, \ldots, b_1) = 0.$$

This shows that the coefficients are *uniquely* determined.

$$\begin{aligned} -b_1 = -1 &\Rightarrow\ b_1 = 1; \\ -b_2 + b_1^2/2 = 0 &\Rightarrow\ b_2 = 1/2; \\ -b_3 + b_1b_2 - b_1/6 = 0 &\Rightarrow\ b_3 = 1/3; \end{aligned}$$

There exists, however, *a much easier way* to determine the coefficients. For the analogous problem with a complex variable z, we know that the solution is unique: $q(z) = -\ln(1-z) = \sum_1^\infty z^j/j$ (the principal branch, where $b_0 = 0$), and hence $\sum_1^\infty \mathbf{x}^j/j$ is the unique formal power series that solves the problem, and we can use the notation $\mathbf{Q} = -\ln(1-\mathbf{x})$ for it.[14]

This example will be applied in Example 3.2.18 to the derivation of formulas for numerical differentiation.

The theory of formal power series can in a similar way justify many elegant "symbolic" applications of power series for deriving mathematical formulas.

Review Questions

1. (a) Formulate three general theorems that can be used for estimating the remainder term in numerical series.

(b) What can you say about the remainder term, if the nth term is $O(n^{-k})$, $k > 1$? Suppose in addition that the series is alternating. What further condition should you add, in order to guarantee that the remainder term will be $O(n^{-k})$?

2. Give, with convergence conditions, the Maclaurin series for $\ln(1+x)$, e^x, $\sin x$, $\cos x$, $(1+x)^k$, $(1-x)^{-1}$, $\ln \dfrac{1+x}{1-x}$, $\arctan x$.

3. Describe the main features of a few methods to compute the Maclaurin coefficients of, e.g., $\sqrt{2e^x - 1}$.

4. Give generating functions of the Bernoulli and the Euler numbers. Describe generally how to derive the coefficients in a quotient of two Maclaurin series.

5. If a functional equation, e.g. $4(\cos x)^3 = \cos 3x + 3\cos x$, is known to be valid for real x, how do you know that it holds also for all complex x? Explain what is meant by the statement that it holds also for formal power series, and why is this true?

6. (a) Show that multiplying two arbitrary upper triangular matrices of order N uses $\sum_{k=1}^N k(N-k) \approx N^3/6$ flops, compared to $\sum_{k=1}^N k \approx N^2/2$ for the product of a row vector and an upper triangular matrix.

(b) Show that if $g(x)$ is a power series and $g(0) = 0$, then $g(S_N)^n = 0$, $n \geq N$. Make an operation count for the evaluation of the matrix polynomial $f(g(S_N))$ by the matrix version of Horner's scheme.

(c) Consider the product $f(S_N)g(S_N)$, where $f(x)$ and $g(x)$ are two power series. Show, using rules for matrix multiplication, that for any $M < N$ the leading $M \times M$ block of the product matrix equals $f(S_M)g(S_M)$.

7. Consider a power series $y = f(x) = \sum_{j=0}^\infty a_j x^j$, where $a_0 = 0$, $a_1 = 1$. What is meant by reversion of this power series? In the Brent–Kung method the

[14] The three coefficients b_j computed above agree, of course, with $1/j$, $j = 1:3$.

problem of reversion of a power series is formulated as a nonlinear equation. Write this equation for the Toeplitz matrix representation of the series.

8. Let $\mathbf{P} = a_0 + a_1\mathbf{x} + a_2\mathbf{x}^2 + \cdots$ and $\mathbf{Q} = b_0 + b_1\mathbf{x} + b_2\mathbf{x}^2 + \cdots$ be two formal power series. Define the sum $\mathbf{P} + \mathbf{Q}$ and the Cauchy product $\mathbf{PQ}$.

Problems and Computer Exercises

1. In how large a neighborhood of $x = 0$ does one get, respectively, four and six correct decimals using the following approximations?

(a) $\sin x \approx x$; (b) $(1+x^2)^{-1/2} \approx 1-x^2/2$; (c) $(1+x^2)^{-1/2}e^{\sqrt{\cos x}} \approx e(1-\frac{3}{4}x^2)$.

Comment: The truncation error is asymptotically qx^p where you know (?) p. An alternative to an exact algebraic calculation of q, is a numerical estimation of q, by means of the actual error for a suitable value of x—neither too big nor too small (!). (Check the estimate of q for another value of x.)

2. (a) Let a, b, be the lengths of the two smaller sides of a right angle triangle, $b \ll a$. Show that the hypotenuse is approximately $a + b^2/(2a)$ and estimate the error of this approximation. If $a = 100$, how large is b allowed to be, in order that the absolute error should be less than 0.01?

(b) How large a relative error do you commit, when you approximate the length of a small circular arc by the length of the chord? How big is the error if the arc is 100 km on a great circle of the earth? (Approximate the earth by a ball of radius $40000/(2\pi)$ km.)

(c) $P(x) = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4$ is a polynomial approximation to $\cos x$ for small values of $|x|$. Estimate the errors of $P(x)$, $P'(x)$, $\frac{1}{x}\int_0^x P(t)\,dt$, and compare them, e.g., for $x = 0.1$.

(d) How accurate is the formula $\arctan x \approx \pi/2 - 1/x$ for $x \gg 1$?

3. (a) Compute $10-(999.999)^{1/3}$ to 9 significant digits, by the use of the binomial expansion. Compare with the result obtained by a computer in IEEE double precision, directly from the first expression.

(b) How many terms of the Maclaurin series for $\ln(1 + x)$ would you need in order to compute $\ln 2$ with an error less than 10^{-6} ? How many terms do you need, if you use instead the the series for $\ln(1 + x)/(1 - x)$, with an appropriate choice of x?

4. Give an approximate expression of the form $ah^b f^{(c)}(0)$ for the error of the estimate of the integral $\int_{-h}^{h} f(x)dx$ obtained by Richardson extrapolation (according to Sec. 1.2.2) from the trapezoidal values $T(h)$ and $T(2h)$.

5. Compute, by means of appropriate expansions, not necessarily in powers of t, the following integrals to (say) five correct decimals.
(This is for paper, pencil and a pocket calculator.)

$$\text{(a)}\ \int_0^{0.1} (1 - 0.1\sin t)^{1/2}\,dt; \qquad \text{(b)}\ \int_{10}^{\infty} (t^3 - t)^{-1/2}\,dt.$$

6. (a) Expand $\arcsin x$ into powers of x by the integration of the expansion of $(1-x^2)^{-1/2}$.

(b) Use the result in (a) to prove the expansion

$$x = \sinh x - \frac{1}{2}\frac{\sinh^3 x}{3} + \frac{1\cdot 3}{2\cdot 4}\frac{\sinh^5 x}{5} - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\frac{\sinh^7 x}{7} + \dots$$

7. (a) Consider the power series for

$$(1+x)^{-\alpha}, \quad x > 0, \quad 0 < \alpha < 1.$$

Show that it is equal to the hypergeometric function $F(\alpha, 1, 1, -x)$. Is it true that the expansion is alternating, and that the remainder has the same sign as the first neglected term, also if $x > 1$, where the series is divergent? What do the Theorems 3.1.3 and 3.1.4 tell you in the cases $x < 1$ and $x > 1$?

Comment: An application of the divergent case for $\alpha = \frac{1}{2}$ is found in Problem 3.2.9 (c).

(b) Express the coefficients of the power series expansions of $y\cot y$ and $\ln(\sin y/y)$ in terms of the Bernoulli numbers.

Hint: Set $x = 2iy$ into (3.1.17). Differentiate the second function.

(c) Find a recurrence relation for the Euler numbers E_n, see Example 3.1.5, and use it for showing that these numbers are integers.

(d) Show that

$$\ln\Big(\frac{z+1}{z-1}\Big) = 2\Big(\frac{1}{z} + \frac{1}{3z^3} + \frac{1}{5z^5} + \dots\Big), \quad |z| > 1.$$

Find a recurrence relation for the coefficients of the expansion

$$\Big(\ln\Big(\frac{z+1}{z-1}\Big)\Big)^{-1} = \frac{1}{2}z - \mu_1 z^{-1} - \mu_3 z^{-3} - \mu_5 z^{-5} - \dots, \quad |z| > 1.$$

Compute μ_1, μ_3, μ_5 and determine $\sum_0^\infty \mu_{2j+1}$ by letting $z \downarrow 1$. (Full rigor is not required.)

Hint: Look at Example 3.1.5.

8. The power series expansion $g(x) = b_1 x + b_2 x^2 + \dots$ is given. Find recurrence relations for the coefficients of the expansion for $h(x) \equiv f(g(x)) = c_0 + c_1 x + c_2 x^2 + \dots$ in the following cases:

(a) $h(x) = \ln(1+g(x))$, $f(x) = \ln(1+x)$.

Hint: Show that $h'(x) = g'(x) - h'(x)g(x)$. Then proceed analogously to Example 3.1.10.

Answer:

$$c_0 = 0, \quad c_n = b_n - \frac{1}{n}\sum_{j=1}^{n-1}(n-j)c_{n-j}b_j.$$

(b) $h(x) = (1+g(x))^k$, $f(x) = (1+x)^k$, $k \in \mathbf{R}$, $k \neq 1$.

Hint: Show that $g(x)h'(x) = kh(x)g'(x) - h'(x)$. Then proceed analogously to Example 3.1.10.

Answer:

$$c_0 = 1, \qquad c_n = \frac{1}{n}\sum_{j=1}^{n}\big((k+1)j - n\big)c_{n-j}b_j,$$

$n = 1, 2, \ldots$. The recurrence relation is known as the J. C. P. Miller formula.

(c) $h_1(y) = \cos g(x)$, $h_2(y) = \sin g(x)$, simultaneously.

Hint: Consider instead $h(y) = e^{ig(x)}$, and separate real and imaginary parts afterwards.

9. In Problems 9–12 we use notations and results from the Toeplitz matrix representation in Sec. 3.1.4. For example, S_N denotes the Nth order shift matrix and $f1$ is the first row vector of the matrix $f(S_N)$. We also assume that you are familiar with simple MATLAB . We now present some more advanced notions from MATLAB that are important in our context.

The solution to the linear system (3.1.24), i.e. $q1 \cdot \mathrm{toep}(g1, N) = f1$, can in MATLAB be neatly written as `q1 = f1/toep(g1,N)`. Note that this is the *vector by matrix* division of MATLAB .

If x is a vector, `cumprod(x)` is the *cumulative product* of the elements of x, e.g., `cumprod(2:5) = [2 6 24 120];` cumsum is analogously defined.

If some of the arguments of a function, in the sense of MATLAB , are optional, `nargin` is the number of input arguments actually used; `nargout` is defined analogously.

The MATLAB function `[Nu,De] = rat(v,Tol)` returns two integer vectors so that `abs(Nu./De - v) <= Tol*abs(v)`. There are several variants of the function `rat`; see the help file. This function is based on a version of the continued fraction algorithm presented in Sec. 3.5.2. Take at least two different values for TOL, and compare the results. Use the rational form of a result, only if it seems reliable and shorter than the floating form.

Choose $N = 6$ while you test that a code is correct. When you apply it or examine the properties of the algorithm, choose N in the range $[12, 24]$. (Even then the computing time may be too short to be measured by the "the stopwatch timer" tic ... toc; `tic` starts the timer; `toc`, by itself, displays the elapsed time since tic was used. (You can also *save* the elapsed time by a statement like `t = toc`.) If you choose N very large you may risk exponent underflow or overflow, or some other nuisance.

In most of the following examples, the algorithms are reasonably stable. Numerical instability can occur, however, depending on the functions $f, g, \ldots$ that they are applied to, and it is a good habit to try to compare the results of two "independent" ways to derive an expansion. In applications to ill-conditioned power series; see Sec. 3.2.5, high values of N are needed, and the results may sometimes be ruined by numerical instability, unless multiple precision is used.

(a) Convince yourself that the following function expands the row r to a triangular Toeplitz matrix. What is done if `length(r) < N` and why? What is the default value of the optional input argument `N`?

```
function T = toep(r,N);
% toep expands the row vector r into an upper
% triangular Toeplitz matrix T.
%N is an optional argument.
%
lr= length(r);
if (nargin==1 | lr > N), N = lr; end;
if lr<N,  r=[r, zeros(1,N-lr)]; end;
gs = zeros(N,N);
for i=1:N,
   gs(i,i:N) = r(1:N-i+1);
end
T = gs;
```

(b) If you want $N > 3$, terms in your results, although the number of terms in the given expression for $f(x)$, e.g., $(1+x+x^2)/(1-x+x^2)$ is smaller, you must augment this by sequences of zeros, so that the order of Toeplitz matrix becomes N. Show experimentally and theoretically that the first row of

$$(I_N + S_N + S_N^2)/(I_N - S_N + S_N^2)$$

is, e.g., obtained by the statement

```
[1, 1, 1, zeros(1,N-3)]/toep([1, -1, 1, zeros(1,N-3)]).
```

(c) Let $f(z) = -z^{-1}\ln(1-z)$. Compute the first six coefficients of the Maclaurin series for the functions $f(z)^k$, $k = 1:5$ in floating point, and convert them to rational form. (The answer and an application to numerical differentiation are given in Example 3.3.6.)

Comment: If you choose an appropriate tolerance in the MATLAB function rat you will obtain an accurate rational approximation, but it is not necessarily exact. Try to judge which of the coefficients are exact.

(d) Compute in floating point the coefficients μ_{2j-1}, $j = 1:11$, defined in Problem 7 (d), and convert them to rational form.

Hint: First seek an equivalent problem for an expansion in ascending powers.

(e) Prove that $Q = f(S_N)g(S_N)^{-1}$ is an upper triangular Toeplitz matrix.

Hint: Define $Q = \text{toep}(q1, N)$, where $q1$ is defined by (3.1.24), and show that each row of the equation $Q \cdot g(S_N) = f(S_N)$ is satisfied.

10. (a) Study the following "library" of MATLAB lines for common applications of the Toeplitz matrix method for arbitrary given values of N; the shift matrix S_N corresponds to the variable x. You are welcome to add new "cases", e.g., for some of the exercises below.

```
function y = toeplib(cas,N,par)
% cas is a string parameter;
% par is an optional real or complex scalar;
% the default value is 1.
```

```
% All series are truncated to N terms.
if  nargin == 2, par = 1; end
if  cas == 'bin',
      y=[1  cumprod(par:-1:par-N+2)./cumprod(1:N-1)];
%     y = 1st row of binomial series (1+x)^par, par in R;
elseif   cas == 'con',
  y = cumprod([1  par*ones(1,N-1)]);
%  The array multiplication y.*f1 returns the first
%  row of f(par*S_N);
%  sum(y.*f1) evaluates f(par). See also Problem~(b).
elseif cas == 'exp',
   y = [1  cumprod(par./[1:(N-1)]];
%  y = 1st row of exponential \exp(par*x).
%  Since par can be complex, circular
%  (or trigonometric) functions can also be expanded.
elseif cas == 'log',
    y=[0  1./[1:(N-1)]].*cumprod ([-1 -par*ones(1:N-1)]);
%   y= 1st row of logarithm \ln(1+par*x).
elseif cas == 'e1t',
    y=[1  zeros(1,N-1)];
%   y=e_1^T , i.e. 1st row of (eye)N.
elseif cas == 'SN1', y = [0  1  zeros(1,N-2)];
%   y=1st row of S_N .
elseif cas == 'dif',  y = [0  1:(N-1)];
% The array multiplication y.*f1 returns xf'(x).
else  cas == 'int',  y =1./[1:N].
% The array multiplication y.*f1 returns
% {1\over x}\int_0^x f(t) dt.
end
```

(b) *Evaluation of* $f(x)$. Given `N` and `f1` of your own choice, set `fterms = toeplib('con',N,x).*f1`. What is `sum(fterms)` and `cumsum(fterms)`? When can `sum(fliplr(fterms))` be useful?

(c) Write a code that, for arbitrary given N, returns the 1st rows of the Toeplitz matrices for $\cos x$ and $\sin x$, with S_N corresponding to x, and then transforms them to to 1st rows for Toeplitz matrices with S_N corresponding to x^2. Apply this, for (say) $N = 36$, to determine the errors of the coefficients of $4(\cos x)^3 - 3\cos x - \cos 3x$.

(d) Find out how a library "`toeplib2`" designed for Toeplitz matrices for *even* functions, where S_N corresponds to x^2, must be different from `toeplib`. For example how are `cas == 'dif'` and `cas == 'int'` to be changed?

(e) Unfortunately, a `toeplib` "case" has at most one parameter, namely `par`. Write a code that calls toeplib twice for finding the Maclaurin coefficients of the three parameter function

$$y = (a + bx)^\alpha, \quad a > 0,$$

b, α real. Compute the coefficients in two different ways for $N = 24$; $a = 2$; $b = -1$; $\alpha = \pm 3$, and compare the results for estimating the accuracy of the

coefficients.

(f) Compute the Maclaurin expansions for $(1-x^2)^{-1/2}$ and $\arcsin(x)$, and for $y = 2arcsinh\,(x/2)$. Expand also dy/dx and y^2. Convert the coefficients to rational numbers, as long as they seem to be reliable. Save the results, or make it easy to reproduce them, for comparisons with the results of Problem 12(b).

Comment: The last three series are fundamental for the expansions of differential operators in powers of central difference operators, which lead to highly accurate formulas for numerical differentiation.

(g) Two power series that generate the Bernoulli numbers are given in Example 3.1.5, namely

$$x \equiv \Big(\sum_{i=1}^{\infty} \frac{x^i}{i!}\Big)\Big(\sum_{j=0}^{\infty} \frac{B_j x^j}{j!}\Big); \qquad \frac{x}{2}\frac{e^{x/2}+e^{-x/2}}{e^{x/2}-e^{-x/2}} = \sum_{j=0}^{\infty} \frac{B_{2j}x^{2j}}{(2j)!}.$$

Compute B_{2j} for (say) $j \le 30$ in floating point, using each of these formulas, and compute the difference of the results, which are influenced by rounding errors. Try to find, whether one of the sequences is more accurate than the other, by means of the formula in (3.1.18) for (say) $j > 4$. Then convert the results to rational numbers. Use several tolerances in the function rat and compare with [1, Table 23.2]. Some of the results are likely to disagree. Why?

(h) The Kummer confluent hypergeometric function $M(a, b, x)$ is defined by the power series (3.1.14). Kummer's first identity, i.e.

$$M(a, b, -x) = e^{-x}M(b-a, b, x),$$

is important, e.g., because the series on the left hand side is ill-conditioned if $x \gg 1$, $a > 0$, $b > 0$, while the expression on the right hand side is well-conditioned. Check the identity experimentally by computing the difference between the series on the left hand side and on the right for a few values of a, b, The computed coefficients are afflicted by rounding errors. Are the differences small enough to convince you of the validity of the formula?

11. Read about expansions of composite functions in Sec. 3.1.4

(a) Write the Horner recurrence for the evaluation of the matrix polynomial $f(g(S_N))$ according to (3.1.25). Then show that the following MATLAB function evaluates the first row of $h(S_N) = f(g(S_N))$, if $g(0) = 0$.

```
function h1 = comp(f1,g1,N);
%
% INPUT: the integer N and the rows f1, g1, with the
% first N Maclaurin coefficients for the analytic functions
% f(z), g(z).
% OUTPUT: The row h1 with the first N Maclaurin coefficients
% for the composite function h(z)=f(g(z)), where g1(1)=g(0)=0.
% computed according to the algorithm for a composite function.
% Error message if g(0)\ne 0.
```

```
if g1(1) ~= 0,
   error('g(0) ~= 0 in a composite function f(g(z))')
end
e1t = zeros(1,N);  e1t(1)=1;
r = f1(N)*e1t;
gs = toep(g1,N);
for j = N-1:-1:1,
     r = r*gs + f1(j)*e1t;
end
h1 = r;
```

(b) Matrix functions in MATLAB : For $h(z) = e^{g(z)}$ it is convenient to use the matrix function `expm(g(SN))` or, on the vector level, `h1 = e1t*expm(g(SN))`, rather than to use `h1 = comp(f1,g1)`. If $f(0) \neq 0$, you can analogously use the functions `logm` and `sqrtm`. They may be slower and less accurate than `h1 = comp(f1,g1)`, but they are typically fast and accurate enough.
Compare computing time and accuracy in the use of `expm(k * logm(eye(N) + SN)` and `toeplib('bin',N,k)` for a few values of N and k.

Comment: *The* MATLAB *function* `funm` *should, however, not be used*, because it uses an eigenvalue method that does not work well for matrices that have multiple eigenvalues. For triangular Toeplitz matrices the diagonal elements are multiple eigenvalues. The functions `expm, logm` and `sqrtm` should not be confused with the functions `exp, log` and `sqrt`, which operate on the matrix elements.

(c) Expand $e^{\sin(z)}$ in powers of z in two ways: *first* using the function in Problem 11(a); *second* using the matrix functions of MATLAB. Show that the latter can be written

`HN = expm(imag(expm(i*SN))).`

Do not be surprised if you find a dirty imaginary part of H_N. Kill it!
Compare the results of the two procedures. If you have done the runs appropriately, the results should agree excellently.

(d) Treat the series $h(z) = \sqrt{(1 + e^z)}$ in three different ways, and compare the results, with respect to validity, accuracy and computing time.
(i) Set $ha(z) = h(z)$, and determine $f(z)$, $g(z)$, analytic at $z = 0$, so that $g(0) = 0$. Compute `ha1 = comp(f1,g1,N)`. Do you trust the result?
(ii) Set $h(z) = H(z)$. Compute `HN = sqrtm(eye(N) + expm(SN))`.
In the first test, i.e. for $N = 6$, display the matrix H_N, and check that H_N is an upper triangular Toeplitz matrix. For larger values of N, display the first row only, and compare it to $ha1$. If you have done all this correctly, the agreement should be extremely good, and we can practically conclude that both are very accurate.
(iii) Try the "natural", although "illegal', decomposition $hb(z) = f(g(z))$, with $f(x) = (1+x)^{0.5}$, $g(z) = e^z$. Remove temporarily the error stop. Demonstrate by numerical experiment that $hb1$ is very wrong. If this is a surprise, read Sec. 3.1.4 once more.

(e) Suppose that you perform matrix level operations on $f(S_M)$, $g(S_M), \ldots,$

where $M < N$. Show that the results are exactly equal to the principal $M \times M$ submatrices of the results obtained with the matrices $f(S_N)$, $g(S_N), \ldots$

12. *Reversion of series.* Let

$$y = f(x) = \sum_{j=1}^{\infty} f1(j)x^{j-1},$$

where $f1(1) = f(0) = 0$, $f1(2) = f'(0) = 1$ (with the notation used in Sec. 3.1.5 and in the previous problems). Power series reversion is to find the power series for the inverse function

$$x = g(y) = \sum_{j=1}^{\infty} g1(j)y^{j-1},$$

where $g1(1) = g(0) = 0$. Read also the last paragraphs of Sec. 3.1.5.
We work with truncated series with N terms in the Toeplitz matrix representation. The inverse function relationship gives the matrix equation $f(g(S_N)) = S_N$. Because $g(0) = 0$, we have, by (3.1.25),

$$f(g(S_N)) = \sum_{j=1}^{N} f1(j)(g(S_N)^{j-1}.$$

Now Horner's scheme can be used for computing the polynomial *and its derivative*, the latter is obtained by algorithmic differentiation; see Sec. 1.3.1.
The first row of this matrix equation is treated by Newton's method in the code breku listed below. The Horner algorithms are adapted to the first row.[15] The notations in the code is almost the same as in the theoretical description, although lower case letters are used, e.g., the matrix $g(S_N)$ is denoted gs, and $fgs1$ is the first row of the matrix $f(g(S_N))$. The equation reads $fgs1 - s1 = 0$.
(a) Convince yourself that the following MATLAB function implements power series reversion under a certain condition. Describe in a few words the main features of the method.

```
function g1 = breku(f1,N);
%
% INPUT: The row vector f1 that represents a (truncated)
% Maclaurin series
% OUTPUT: The row g1, i.e. the first N terms of the series
% x = g(y) where y = f(x).
% Note that f1(1) = 0, f1(2) = 1; if not, there will
% be an error message. The integer N, i.e. the length
% of the truncated series wanted in the output, is optional
% input; by default N = length(f1).
```

[15] The name"breku" comes from Brent and Kung, who were probably the first mathematicians to apply Newton's method to series reversion, although with a different formulation of the equation than ours (no Toeplitz matrices).

```
% If length(f1) < N, f1 is extended to length N by zeros.
% Uses the function toep(r,N) (see Problem~9) for expanding
% a row to an upper triangular Toeplitz matrix.
%
if ~ (f1(1) ~= 0|f1(2) ~= 1),
    error('wrong f1(1) or f1(2)');
end
lf1 =  length(f1);
if (nargin == 1|lf1 > N), N = lf1; end
if lf1 < N, f1 = [f1 zeros(1, N-lf1)]  end
maxiter = floor(log(N)/log(2));
e1t = [1, zeros(1,N-1)];
s1 = [0 1 zeros(1,N-2)];  gs1 = s1;
for iter = 0:maxiter
    gs  = toep(gs1,N);
% Horner's scheme for computing the first rows
% of f(gs) and f'(g(s)):
    fgs1 = f1(N)*e1t;  der1 = zeros(1,N);
    for j  =  N-1:-1:1
        ofgs1 = fgs1;   %ofgs1 means "old" fgs1
        fgs1 = ofgs1*gs + f1(j)*e1t ;
        der1 = ofgs1 + der1*gs ;
    end
    % A Newton iteration for the equation fgs1 - s1 = 0:
    gs1 = gs1 - (fgs1 - s1)/toep(der1,N);
end
g1 = gs1;
```

Comment: The radius of convergence depends on the singularities of $g(y)$, which are typically related to the singularities of $f(x)$ and to the zeros of $f'(x)$, (why?). There are other cases, e.g., if $f'(x) \to 0$ as $x \to \infty$ then $\lim f(x)$ may be a singularity of $g(y)$.

(b) Apply the code breku to the computation of $g(y)$ for $f(x) = \sin x$ and for $f(x) = 2\sinh(x/2)$. Compare with the results of Problem 10 (d). Then reverse the two computed series $g(y)$, and study how you return to the original expansion of $f(x)$, more or less accurately. Use tic toc to take the time, for a few values of N.

(c) Apply the code breku to compute $g(y)$ for $f(x) = \ln(1+x)$, $f(x) = e^x - 1$, $f(x) = x + x^2$, $f(x) = x + x^2 + x^3$.

If you know an analytic expression for $g(y)$, find the Maclaurin expansion for this, and compare with the expansions obtained from breku .

Apply breku to the computed expansions of $g(y)$, and study how accurately you return to the expansion of $f(x)$.

(d)$f(x) = xe^x$; the inverse function $g(y)$ is known as the Lambert W function.[16] Determine $g(y)$. Then reverse the power series for $g(y)$, and compare with the expansion of $f(x)$.

[16] Johann Heinrich Lambert (1728–1777), German mathematician, physicist and astronomer, and colleague of Euler and Lagrange at the Berlin Academy of Sciences. He is best known for

(e) Estimate for $f(x) = xe^x$ the radius of convergence approximately, by means of the ratios of the coefficients computed in (d), and exactly; see the comment after the code above.

(f)* Set $y = f(x)$. Suppose that $y(0) \neq 0$, $y'(0) \neq 0$. Show that the code breku can be used for expanding the inverse function in powers of $(y - y(0))/y'(0)$. Construct some good test example.

(g)* For the equation $\sin x - (1 - y)x = 0$, express $x^2 = g(y)$ (why x^2?), with $N = 12$. Then express x in the form $x \approx \pm y^{1/2} P(y)$, where $P(y)$ is a truncated power series with (say) 11 terms.

(h)* Make simple temporary changes in the code breku, so that you can follow the iterations on the screen.

3.2 More About Series

3.2.1 Laurent and Fourier Series

A **Laurent series** is a series of the form

$$\sum_{n=-\infty}^{\infty} c_n z^n. \tag{3.2.1}$$

Its convergence region is the intersection of the convergence regions of the expansions

$$\sum_{n=0}^{\infty} c_n z^n \quad \text{and} \quad \sum_{m=1}^{\infty} c_{-m} z^{-m},$$

the interior of which are determined by conditions of the form $|z| < r_2$ and $|z| > r_1$. The convergence region can be void, e.g., if $r_2 < r_1$.

If $0 < r_1 < r_2 < \infty$ the convergence region is an *annulus*, $r_1 < |z| < r_2$. The series defines an analytic function in the annulus. Conversely, if $f(z)$ is a **single-valued analytic function** in this annulus, it is there represented by a Laurent series, that converges uniformly in every closed subdomain of the annulus.

The coefficients are determined by the following formula, due to Cauchy[17]

$$c_n = \frac{1}{2\pi i} \int_{|z|=r} z^{-n-1} f(z) dz, \quad r_1 < r < r_2, -\infty < n < \infty, \tag{3.2.2}$$

and

$$|c_n| \leq r^{-n} \max_{|z|=r} |f(z)|. \tag{3.2.3}$$

The extension to the case when $r_2 = \infty$ is obvious; the extension to $r_1 = 0$ depends on whether there are any terms with negative exponents or not. In the extension of *formal* power series to *formal Laurent series*, however, only a finite number of

his illumination laws and for the continued fraction expansions of elementary functions, Sec. 3.5.3. His W function was "rediscovered" a few years ago, [14].

[17] Augustin Cauchy (1789–1857) is the father of modern analysis. He is the creator of complex analysis, a centerpiece of which this formula plays a fundamental role.

terms with negative indices are allowed to be different from zero; see Henrici loc. cit. Sec. 1.8. If you substitute z for z^{-1} an infinite number of negative indices is allowed, if the number of positive indices is finite.

Example 3.2.1.

A function may have several Laurent expansions (with different regions of convergence), e.g.,

$$(z-a)^{-1} = \begin{cases} -\sum_{n=0}^{\infty} a^{-n-1} z^n & \text{if } |z| < |a| \\ \sum_{m=1}^{\infty} a^{m-1} z^{-m} & \text{if } |z| > |a|. \end{cases}$$

The function $1/(z-1) + 1/(z-2)$ has three Laurent expansions, with validity conditions $|z| < 1$, $1 < |z| < 2$, $2 < |z|$, respectively. The series contains both positive and negative powers of z in the middle case only. The details are left for Problem 4 (a).

Lemma 3.2.2 can, with some modifications, be generalized to Laurent series (and to complex Fourier series), e.g.,(3.2.17) becomes

$$\tilde{c}_n - c_n = \dots c_{n-2N} r^{-2N} + c_{n-N} r^{-N} + c_{n+N} r^{N} + c_{n+2N} r^{2N} \dots \tag{3.2.4}$$

Remark 3.2.1. The restriction to *single-valued* analytic functions is important in this subsection. In this book we cannot entirely avoid to work with **multi-valued** functions such as $\sqrt{z}$, $\ln z, z^{\alpha}$, (α non-integer). We always work with such a function, however, in some region where one branch of it, determined by some convention, is single-valued. In the examples mentioned, the natural conventions are to require the function to be positive when $z > 1$, and to forbid z to cross the negative real axis. In other words, the complex plane has a **cut** along the negative real axis. The annulus mentioned above is in these cases incomplete; its intersection with the negative real axis is missing, and we cannot use a Laurent expansion.

For a function like $\ln\left(\dfrac{z+1}{z-1}\right)$, we can, depending on the context, cut out either the interval $[-1, 1]$ or the complement of this interval with respect to the real axis. We then use an expansion into negative or into positive powers of z, respectively.

If $r_1 < 1 < r_2$, we set $F(t) = f(e^{it})$. Note that $F(t)$ is a periodic function; $F(t+2\pi) = F(t)$. By (3.2.1) and (3.2.2), the Laurent series then becomes for $z = e^{it}$ a **Fourier series**:

$$F(t) = \sum_{n=-\infty}^{\infty} c_n e^{int}, \quad c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-int} F(t)\, dt. \tag{3.2.5}$$

Note that $c_{-m} = O(r_1^m)$ for $m \to +\infty$, and $c_n = O(r_2^{-n})$ for $n \to +\infty$. *The formulas in* (3.2.5), *however, are valid in much more general situations*, where $c_n \to 0$ much more slowly, and where $F(t)$ cannot be continued to an analytic function $f(z)$, $z = re^{it}$, in an annulus. (In such a case $r_1 = 1 = r_2$, typically.)

A Fourier series is often written in the following form,

$$F(t) = \tfrac{1}{2}a_0 + \sum_{k=1}^{\infty}(a_k \cos kt + b_k \sin kt). \tag{3.2.6}$$

Consider $c_k e^{ikt} + c_{-k}e^{-ikt} \equiv a_k \cos kt + b_k \sin kt$. Since $e^{\pm ikt} = \cos kt \pm i \sin kt$, we obtain for $k \geq 0$:

$$a_k = c_k + c_{-k} = \frac{1}{\pi}\int_{-\pi}^{\pi} F(t)\cos kt\, dt; \quad b_k = i(c_k - c_{-k}) = \frac{1}{\pi}\int_{-\pi}^{\pi} F(t)\sin kt\, dt. \tag{3.2.7}$$

Also note that $a_k - ib_k = 2c_k$. If $F(t)$ is real for $t \in \mathbf{R}$ then $c_{-k} = \bar{c}_k$.

We mention without proof the important **Riemann–Lebesgue theorem**,[18] by which the Fourier coefficients c_n *tend to zero as* $n \to \infty$ *for any function that is integrable* (in the sense of Lebesgue), a fortiori for any periodic function that is continuous everywhere. A finite number of finite jumps in each period are also allowed.

A function $F(t)$ is said to be of bounded variation in an interval if, in this interval, it can expressed in the form $F(t) = F_1(t) - F_2(t)$, where F_1 and F_2 are non-decreasing bounded functions. A finite number of jump discontinuities are allowed. The variation of F over the interval $[a, b]$ is denoted $\int_a^b |dF(t)|$. If F is differentiable the variation of F equals $\int_a^b |F'(t)|\, dt$.

Another classical result in the theory of Fourier series reads: *If* $F(t)$ *is of* **bounded variation**. *in the closed interval* $[-\pi, \pi]$ *then* $c_n = O(n^{-1})$; see Titchmarsh [45, § 13.21, §13.73]. This can be generalized. *Suppose that* $F^{(p)}$ *is of bounded variation on* $[-\pi, \pi]$, *and that* $F^{(j)}$ *is continuous everywhere for* $j < p$. *Denote the Fourier coefficients of* $F^{(p)}(t)$ *by* $c_n^{(p)}$. *Then*

$$c_n = (in)^{-p} c_n^{(p)} = O(n^{-p-1}). \tag{3.2.8}$$

This follows from the above classical result, after the integration of the formula for c_n in (3.2.2) by parts p times. Bounds for the truncation error of a Fourier series can also be obtained from this. The details are left for Problem 4 (d), together with a further generalization. A similar result is that $c_n = o(n^{-p})$ if $F^{(p)}$ is integrable, hence a fortiori if $F \in C^p$.

In particular, we find for $p = 1$ (since $\sum n^{-2}$ is convergent) that the Fourier series (3.2.2) *converges absolutely and uniformly* in $\mathbf{R}$. It can also be shown that *the Fourier series is valid*, i.e. the sum is equal to $F(t)$.

3.2.2 The Cauchy–FFT Method

An alternative method for deriving coefficients of power series, when many terms are needed is based on the following classic result. Suppose that the value $f(z)$

[18] Jean Baptist Joseph Fourier (1768–1830), French mathematician and physicist. In a pioneering publication about the flow of heat (1822), he utilized series of the type of equation (3.2.6). B. Riemann (1826-1866), German mathematician, made fundamental contributions to Analysis and Geometry. H. Lebesgue (1875-1941), French mathematician, created path-breaking general concepts of measure and integral.

of an analytic function can be computed at any point inside and on the circle $C_r = \{z : |z-a| = r\}$, and set

$$M(r) = \max|f(z)|, \quad z \in C_r, \quad z = a + re^{i\theta}, \quad z' = a + r'e^{i\theta},\ (r' < r).$$

Then the coefficients of the Taylor expansion around a are determined by Cauchy's formula,

$$a_n = \frac{1}{2\pi i}\int_{C_r} \frac{f(z)}{(z-a)^{(n+1)}}\,dz = \frac{r^{-n}}{2\pi}\int_0^{2\pi} f(a+re^{i\theta})e^{-ni\theta}\,d\theta. \tag{3.2.9}$$

For a derivation, multiply the Taylor expansion (3.1.3) by $(z-a)^{-n-1}$, integrate term by term over C_r, and note that

$$\frac{1}{2\pi i}\int_{C_r}(z-a)^{j-n-1}\,dz = \frac{1}{2\pi}\int_0^{2\pi} r^{j-n}e^{(j-n)i\theta}\,d\theta = \begin{cases} 1, & \text{if } j = n; \\ 0, & \text{if } j \neq n. \end{cases} \tag{3.2.10}$$

The following inequalities are useful consequences of the definitions and of (3.2.9).

$$|a_n| \le r^{-n}M(r), \tag{3.2.11}$$

$$|R_n(z')| \le \sum_{j=n}^{\infty}|a_j(z'-a)^j| \le \sum_{j=n}^{\infty} r^{-j}M(r)(r')^j = \frac{M(r)(r'/r)^n}{1-r'/r}, \quad 0 \le r' < r.$$

This form of the remainder term of a Taylor series is useful in theoretical studies, and also for practical purpose, if the maximum modulus $M(r)$ is easier to estimate than the nth derivative.

Set $z = a + re^{i\theta}$, $\Delta\theta = 2\pi/N$, and apply the trapezoidal rule to the second integral in (3.2.9). Then[19]

$$a_n \approx \tilde{a}_n \equiv \frac{1}{Nr^n}\sum_{k=0}^{N-1} f(a+re^{ik\Delta\theta})e^{-ink\Delta\theta}, \quad n = 0 : N-1. \tag{3.2.12}$$

The approximate Taylor coefficients $\tilde{a}_n$, or rather the numbers $a_n^\star = \tilde{a}_n N r^n$, are here expressed as a case of the (direct) **Discrete Fourier Transform**. More generally, this transform maps an *arbitrary* sequence $\{\alpha_k\}_0^{N-1}$ to a sequence $\{a_n^\star\}_0^{N-1}$, by the following equations:

$$a_n^\star = \sum_{k=0}^{N-1}\alpha_k e^{-ink\Delta\theta}, \quad n = 0 : N-1. \tag{3.2.13}$$

The transform will be studied more systematically in Sec. 4.6.

If N is a power of 2, it is shown in Sec. 4.6 that, given the N values α_k, $k = 0 : N-1$, and $e^{-i\Delta\theta}$, *no more than* $N\log_2 N$ *complex multiplications and additions are*

[19] See (1.2.6). Note that the integrand has the same value for $\theta = 2\pi$ as for $\theta = 0$. The terms $\frac{1}{2}f_0$ and $\frac{1}{2}f_N$ that appear in the general trapezoidal rule can therefore in this case be replaced by f_0.

needed for the computation of all the N coefficients $a_n^\star$, if an implementation of the discrete Fourier transform known as the **Fast Fourier Transform (FFT)** is used. This makes our theoretical considerations very practical. (Packages for interactive mathematical computation usually contain commands related to FFT.)

It is also shown in Sec. 4.6 that the **inverse** of the discrete Fourier transform (3.2.13) is given by the formulas,

$$\alpha_k = (1/N)\sum_{n=0}^{N-1} a_n^\star e^{ink\Delta\theta}, \quad k = 0 : N-1. \tag{3.2.14}$$

It looks almost like the direct Discrete Fourier Transform (3.2.13), except for the sign of i and the factor $1/N$. It can therefore also be performed by means of $O(N \log N)$ elementary operations, instead of the $O(N^3)$ operations that the most obvious approach to this task would require, (i.e. by solving the linear system (3.2.13)).

In our context, i.e. the computation of Taylor coefficients, we have, by (3.2.12) and the line after that equation,

$$\alpha_k = f(a + re^{ik\Delta\theta}), \qquad a_n^\star = \tilde{a}_n N r^n. \tag{3.2.15}$$

Set $z_k = a + re^{ik\Delta\theta}$. Using (3.2.15), the inverse transformation then becomes,[20]

$$f(z_k) = \sum_{n=0}^{N-1} \tilde{a}_n (z_k - a)^n, \quad k = 0 : N-1. \tag{3.2.16}$$

Since the Taylor coefficients are equal to $f^{(n)}(a)/n!$, this is de facto a method for the accurate *numerical differentiation of an analytic function.* If r and N are chosen appropriately, it is more well-conditioned than most methods for *numerical differentiation*, such as the difference approximations mentioned in Chapter 1; see also Sec. 3.3 and Chapter 4. It requires, however, complex arithmetic for a convenient implementation. We call this the **Cauchy–FFT method** for Taylor coefficients and differentiation.

The question arises, how to choose N and r. Theoretically, any r less than the radius of convergence ρ would do, but there may be trouble with cancellation if r is small. On the other hand, the truncation error of the numerical integration usually increases with r. "*Scylla and Charybdis situations*" [21] like this are very common with numerical methods.

It is typically the rounding error that sets the limit for the accuracy; it is usually not expensive to choose r and N, so that the truncation error becomes much smaller. A rule of thumb for this situation is to guess a value of $\hat{n}$, i.e. how

[20] One interpretation of these equations is that the polynomial $\sum_{n=0}^{N-1} \tilde{a}_n (z-a)^n$ is the solution of a special, although important, interpolation problem for the function f, analytic inside a circle in $\mathbf{C}$.

[21] According to American Heritage Dictionary Scylla is a rock on the Italian side of the Strait of Messina, opposite to the whirlpool Charybdis, personified by Homer (Ulysses) as a female sea monster who devoured sailors. The problem is to navigate safely between them.

many terms will be needed in the expansion, and then to try two values for N (powers of 2) larger than $\hat{n}$. *If ρ is finite* try $r = 0.9\rho$ and $r = 0.8\rho$, and compare the results. They may or may not indicate that some other values of N and r (and perhaps also $\hat{n}$) should also be tried. On the other hand, *if $\rho = \infty$*, try, e.g., $r = 1$ and $r = 3$, and compare the results. Again the results indicate whether or not more experiments should be made.

One can also combine numerical experimentation with a theoretical analysis of a more or less simplified model, including a few elementary optimization calculations. The authors take the opportunity to exemplify below this type of "hard analysis" on this question.

We first derive two lemmas, which are important also in many other contexts. First we have a discrete analogue of equation (3.2.10).

Lemma 3.2.1. *Let p, N be integers. Then $\sum_{k=0}^{N-1} e^{2\pi ipk/N} = 0$, unless $p = 0$ or p is a multiple of N. In these exceptional case every term equals 1, and the sum equals N.*

Proof. If p is neither 0 nor a multiple of N, the sum is a geometric series, the sum of which is equal to $(e^{2\pi ip} - 1)/(e^{2\pi ip/N} - 1) = 0$. The rest of the statement is obvious. □

Lemma 3.2.2. *Suppose that $f(z) = \sum_0^\infty a_n(z-a)^n$ is analytic in the disc $|z-a| < \rho$. Let $\tilde{a}_n$ be defined by* (3.2.12)*, where $r < \rho$. Then*

$$\tilde{a}_n - a_n = a_{n+N}\, r^N + a_{n+2N}\, r^{2N} + a_{n+3N}\, r^{3N} + \ldots, \quad 0 \le n < N. \tag{3.2.17}$$

Proof. Since $\Delta\theta = 2\pi/N$,

$$\tilde{a}_n = \frac{1}{Nr^n} \sum_{k=0}^{N-1} e^{-2\pi ink/N} \sum_{m=0}^{\infty} a_m \Big(re^{2\pi ik/N}\Big)^m = \frac{1}{Nr^n} \sum_{m=0}^{\infty} a_m r^m \sum_{k=0}^{N-1} e^{2\pi i(-n+m)k/N}.$$

By the previous lemma, the inner sum of the last expression is zero, unless $m - n$ is a multiple of N. Hence (recall that $0 \le n < N$),

$$\tilde{a}_n = \frac{1}{Nr^n} \left(a_n r^n N + a_{n+N}\, r^{n+N} N + a_{n+2N}\, r^{n+2N} N + \ldots\right),$$

from which equation (3.2.17) follows. □

Let $M(r)$ be the maximum modulus for the function $f(z)$ on the circle C_r, and denote by $M(r)U$ an upper bound for the error of a computed function value $f(z)$, $|z| = r$, where $U \ll 1$. Assume that rounding errors during the computation of $\tilde{a}_n$ are of minor importance.

Then, by (3.2.12), $M(r)U/r^n$ is a bound for the *rounding error* of $\tilde{a}_n$. (The rounding errors during the computation can be included by a redefinition of U.)

Next we shall consider the *truncation error* of (3.2.12). First we *estimate* the coefficients that occur in (3.2.17) by means of $\max|f(z)|$ on a circle with radius r'; $r' > r$, where r is the radius of the circle used in the *computation* of the first N coefficients. So, in (3.2.9) we substitute r', j for r, n, respectively, and obtain the inequality

$$|a_j| \le M(r')(r')^{-j}, \qquad 0 < r < r' < \rho.$$

The actual choice of r', strongly depends on the function f.[22] Put this inequality into (3.2.17), where we shall choose $r < r' < \rho$. Then

$$\begin{aligned}|\tilde{a}_n - a_n| &\le M(r')\left((r')^{-n-N}r^N + (r')^{-n-2N}r^{2N} + (r')^{-n-3N}r^{3N} + \ldots\right)\\ &= M(r')(r')^{-n}\left((r/r')^N + (r/r')^{2N} + (r/r')^{3N} + \ldots\right) = \frac{M(r')(r')^{-n}}{(r'/r)^N - 1}.\end{aligned}$$

We make a digression here, because *this is an amazingly good result.* The trapezoidal rule that was used in the calculation of the Taylor coefficients is typically expected to have an error that is $O\big((\Delta\theta)^2\big) = O\big(N^{-2}\big)$. (As before, $\Delta\theta = 2\pi/N$.) This application is, however, *a very special situation: a periodic analytic function is integrated over a full period.* We shall return to results like this several times. In this case, for fixed values of r, r', the truncation error is $O\big((r/r')^N\big) = O\big(e^{-\eta/\Delta\theta}\big)$, where $\eta > 0$, $\Delta\theta \to 0+$. This tends to zero faster than any power of $\Delta\theta$.

It follows that a bound for the total error of $\tilde{a}_n$, i.e. the sum of the bounds for the rounding and the truncation errors, is given by

$$UM(r)r^{-n} + \frac{M(r')(r')^{-n}}{(r'/r)^N - 1}, \qquad r < r' < \rho. \tag{3.2.18}$$

Example 3.2.2. "Scylla and Charybdis" in the Cauchy–FFT.

We shall discuss how to choose the parameters r and N, so that the *absolute error bound* of a_n, given in (3.2.18) becomes uniformly small for (say) $n = 0 : \hat{n}$. $1 + \hat{n} \gg 1$ is thus the number of Taylor coefficients requested. The parameter r' does not belong to the Cauchy–FFT method, but it has to be chosen well in order to make the *bound* for the truncation error realistic.

The discussion is rather technical, and you may omit it at a first reading. It may, however, be useful to study this example later, because similar technical subproblems occur in many serious discussions of numerical methods that contain parameters that should be appropriately chosen.

First consider the *rounding error.* By the maximum modulus theorem, $M(r)$ is an increasing function, hence, for $r > 1$, $\max_n M(r)r^{-n} = M(r) > M(1)$. On the other hand, for $r \le 1$, $\max_n M(r)r^{-n} = M(r)r^{-\hat{n}}$.[23] Let r^* be the value of r, for which this maximum is minimal. Note that $r^* = 1$ unless $M'(r)/M(r) = \hat{n}/r$ for some $r \le 1$.

Then try to determine N and $r' \in [r^*, \rho)$ so that, for $r = r^*$, the bound for the the second term of (3.2.18) becomes much smaller than the first term, i.e. *the*

[22] In rare cases we may choose $r' = \rho$.

[23] $\hat{n}$ was introduced in the beginning of this example.

truncation error is made negligible compared to the rounding error. This works well if $\rho \gg r^*$. In such cases, we may therefore choose $r = r^*$, and the total error is then just a little larger than $UM(r^*)(r^*)^{-\hat{n}}$.

For example, if $f(z) = e^z$ then $M(r) = e^r$, $\rho = \infty$. In this case $r^* = 1$ (since $\hat{n} \gg 1$). Then we shall choose N and $r' = N$, so that $e^{r'}/((r')^N - 1) \ll eU$. One can show that it is sufficient to choose $N \gg |\ln U / \ln|\ln U||$. For instance, if $U = 10^{-15}$, this is satisfied with a wide margin by $N = 32$. On a computer, the choice $r = 1$, $N = 32$, gave (with 53 bits floating point arithmetic) an error less than $2 \cdot 10^{-16}$. The results were much worse for $r = 10$, and for $r = 0.1$; the maximum error of the first 32 coefficients became $4 \cdot 10^{-4}$ and $9 \cdot 10^{13}$(!), respectively. In the latter case the errors of the first 8 coefficients did not exceed 10^{-10}, but the rounding error of a_n, due to cancellations, increase rapidly with n.

If ρ is not much larger than r^*, the procedure described above may lead to a value of N that is much larger than $\hat{n}$. In order to avoid this, we now set $\hat{n} = \alpha N$. We now confine the discussion to the case that $r < r' < \rho \leq 1$, $n = 0 : \hat{n}$. Then, with all other parameters fixed, the bound in (3.2.18) is maximal for $n = \hat{n}$. We simplify this bound; $M(r)$ is replaced by the larger quantity $M(r')$, and the denominator is replaced by $(r'/r)^N$.

Then, for given r', α, N, we set $x = (r/r')^N$ and determine x so that

$$M(r')(r')^{-\alpha N}(Ux^{-\alpha} + x)$$

is minimized. The minimum is obtained for $x = (\alpha U)^{1/(1+\alpha)}$, i.e. for $r = r' x^{1/N}$, and the minimum is equal to[24]

$$M(r')(r')^{-n} U^{1/(1+\alpha)} c(\alpha), \quad \text{where} \quad c(\alpha) = (1+\alpha)\alpha^{-\alpha/(1+\alpha)}.$$

We see that the error bound contains the factor $U^{1/(1+\alpha)}$. This is, e.g., proportional to $2U^{1/2}$ for $\alpha = 1$, and to $1.65U^{4/5}$ for $\alpha = \frac{1}{4}$. The latter case is thus much more accurate, but, for the same $\hat{n}$, one has to choose N four times as large, which leads to more than four times as many arithmetic operations. In practice, $\hat{n}$ is usually given, and the order of magnitude of U can be estimated. Then α is to be chosen to make a compromise between the requirements for a good accuracy and for a small volume of computation. *If ρ is not much larger than r^**, we may choose

$$N = \hat{n}/\alpha, \quad x = (\alpha U)^{1/(1+\alpha)}, \quad r = r' x^{1/N}.$$

Experiments were made with $f(z) = \ln(1 - z)$. Then $\rho = 1$, $M(1) = \infty$. Take $\hat{n} = 64$, $U = 10^{-15}$, $r' = 0.999$. Then $M(r') = 6.9$. For $\alpha = 1, 1/2, 1/4$, we have $N = 64, 128, 256$, respectively. The above theory suggests $r = 0.764, 0.832, 0.894$, respectively. The theoretical estimates of the absolute errors become, $10^{-9}, 2.4\,10^{-12}, 2.7\,10^{-14}$, respectively. The smallest errors obtained in experiments with these three values of α are, $6\,10^{-10}, 1.8\,10^{-12}, 1.8\,10^{-14}$, which were obtained for $r = 0.766, 0.838, 0.898$, respectively. So, the theoretical predictions of these experimental results are very satisfactory.

[24] This is a rigorous upper bound of the error for this value of r, in spite of the simplifications in the formulation of the minimization.

3.2.3 Chebyshev Polynomials

The **Chebyshev**[25] **polynomials** of the first kind are

$$T_n(z) = \cos n\phi, \quad z = \cos\phi, \tag{3.2.19}$$

Note that $T_0(z) = 1$, $T_1(z) = z$. That $T_n(z)$ is an nth degree polynomial follows, by induction, from the important **recurrence relation**,

$$T_{n+1}(z) = 2zT_n(z) - T_{n-1}(z), \quad (n \geq 1), \tag{3.2.20}$$

which follows from the well known trigonometric formula

$$\cos(n+1)\phi + \cos(n-1)\phi = 2\cos\phi\cos n\phi.$$

We obtain,

$$T_2(z) = 2z^2 - 1; \quad T_3(z) = 4z^3 - 3z; \quad T_4(z) = 8z^4 - 8z^2 + 1,$$
$$T_5(z) = 16z^5 - 20z^3 + 5z; \quad T_7(z) = 64z^7. - 112z^5 + 56z^3 - 7z$$

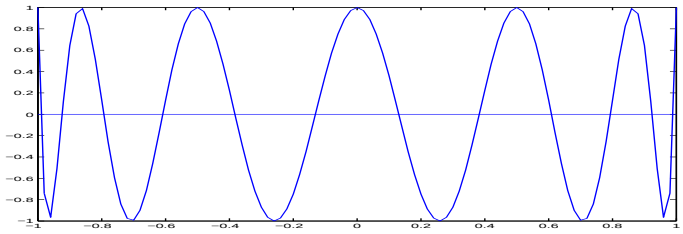


Figure 3.2.1. *The Chebyshev polynomial* $T_{12}(x), x \in [-1, 1]$.

The **Chebyshev polynomials** of the second kind are

$$U_{n-1}(z) = \frac{\sin n\phi}{\sin\phi}, \quad \text{where} \quad z = \cos\phi, \tag{3.2.21}$$

satisfies the same recurrence relation, with the initial conditions $U_{-1}(z) = 0$, $U_0(z) = 1$; its degree is $n - 1$. (When we write just Chebyshev polynomial we refer to the first kind.)

The Chebyshev polynomial $T_n(x)$ has n zeros in $[-1, 1]$ given by

$$x_k = \cos\left(\frac{2k-1}{n}\frac{\pi}{2}\right), \quad k = 1 : n, \tag{3.2.22}$$

[25] Pafnuti Lvovich Chebyshev (1821–1894), Russian mathematician, pioneer in approximation theory and the constructive theory of functions. His name has many different transcriptions, e.g., Tschebyscheff. This may explain why the polynomials that bear his name are denoted $T_n(x)$. He also gave important contributions to probability theory and number theory.

the **Chebyshev points**, and $n+1$ *extrema*

$$x'_k = \cos\Big(\frac{k\pi}{n}\Big), \quad k = 0:n. \tag{3.2.23}$$

These results follow directly from the fact that $\cos(n\phi) = 0$ for $\phi = (2k+1)\pi/(2n)$, and that $\cos(n\phi)$ has maxima for $\phi = k\pi/n$.

Note that from (3.2.19) it follows that $|T_n(x)| \le 1$ for $x \in [-1,1]$, in spite that its leading coefficient is as large as 2^{n-1}. The Chebyshev polynomials have a unique **minimax property**: (For a use of this property; see, Example 3.2.4.)

Example 3.2.3.

Figure 3.2.1 shows a plot of the Chebyshev polynomial $T_{12}(x)$ for $x \in [-1,1]$. Setting $z = 1$ in the recurrence relation (3.2.20) and using $T_0(1) = T_1(1) = 1$, it follows that $T_n(1) = 1$, $n \ge 0$. From $T'_0(1) = 0$ an $T'_1(1) = 1$ and differentiating the recurrence relation we get

$$T'_{n+1}(z) = 2(zT'_n(z) + T_n(z)) - T'_{n-1}(z), (n \ge 1).$$

It follows easily by induction that $T'_n(1) = n^2$, that is *outside the interval* $[-1,1]$ *the Chebyshev polynomials grow rapidly.*

Lemma 3.2.3.

The Chebyshev polynomials have the following **minimax property***: Of all nth degree polynomials with leading coefficient* 1*, the polynomial* $2^{1-n}T_n(x)$ *has the smallest magnitude* 2^{1-n} *in* $[-1,1]$*.*

Proof. Suppose there were a polynomial $p_n(x)$, with leading coefficient 1 such that $|| < 2^{1-n}$ for all $x \in [-1,1]$. Let x'_k, $k = 0:n$, be the abscissae of the extrema of $T_n(x)$. Then we would have

$$p_n(x_0) < 2^{1-n}T_n(x'_0), \quad p_n(x_1) > 2^{1-n}T_n(x'_1), \quad p_n(x_2) < 2^{1-n}T_n(x'_2), \ldots,$$

etc., up to x'_n. From this it follows that the polynomial

$$p_n(x) - 2^{1-n}T_n(x)$$

changes sign in each of the n intervals (x'_k, x'_{k+1}), $k = 0:n-1$. This is impossible, since the polynomial is of degree $n-1$. This proves the minimax property. □

Expansions in terms of Chebyshev polynomials are an important aid in studying functions on the interval $[-1,1]$. If one is working with a function $f(t)$, $t \in [a,b]$, then one should make the substitution

$$t = \tfrac{1}{2}(a+b) + \tfrac{1}{2}(b-a)x, \tag{3.2.24}$$

which maps the interval $[-1,1]$ onto $[a,b]$.

Consider the approximation to the function $f(x) = x^n$ on $[-1, 1]$ by a polynomial of lower degree. From the minimax property of Chebyshev polynomials it follows that the maximum magnitude of the error is minimized by the polynomial

$$p(x) = x^n - 2^{1-n}T_n(x). \tag{3.2.25}$$

From the symmetry property $T_n(-x) = (-1)^n T_n(x)$, it follows that this polynomial has in fact degree $n-2$. The error $2^{1-n}T_n(x)$ assumes its extrema 2^{1-n} in a sequence of $n+1$ points, $x_i = \cos(i\pi/n)$. The sign of the error alternates at these points.

Suppose that one has obtained, e.g., by Taylor series, a truncated power series approximation to a function $f(x)$. By repeated use of (3.2.25), the series can be replaced by a polynomial of lower degree with a moderately increased bound for the truncation error. This process, called **economization of power series** often yields a useful polynomial approximation to $f(x)$ with a considerably smaller number of terms than the original power series.

Example 3.2.4.

If the series expansion $\cos x = 1 - x^2/2 + x^4/24 - \cdots$ is truncated after the x^4-term, the maximum error is 0.0014 in $[-1, 1]$. Since $T_4(x) = 8x^4 - 8x^2 + 1$, it holds that

$$x^4/24 \approx x^2/24 - 1/192$$

with an error which does not exceed $1/192 = 0.0052$. Thus the approximation

$$\cos x = (1 - 1/192) - x^2(1/2 - 1/24) = 0.99479 + 0.45833x^2$$

has an error whose magnitude does not exceed $0.0052 + 0.0014 < 0.007$. This is less than one-sixth of the error 0.042, which is obtained if the power series is truncated after the x^2-term.

Note that for the economized approximation $\cos(0)$ is not approximated by 1. It may not be acceptable that such an exact relation is lost. In this example one could have asked for a polynomial approximation to $(1 - \cos x)/x^2$ instead.

The Chebyshev polynomials are perhaps the most important example of a family of **orthogonal polynomials**; see Sec. 4.5.5. The **Chebyshev expansion** of a function $f(z)$,

$$f(z) = \sum_{j=0}^{\infty} c_j T_j(z), \tag{3.2.26}$$

have many useful properties. Set $e^{i\phi} = w$; ϕ and z may be complex. Then

$$z = \tfrac{1}{2}(w + w^{-1}), \quad T_n(z) = \tfrac{1}{2}(w^n + w^{-n}), \tag{3.2.27}$$

$$w = z \pm \sqrt{z^2 - 1}, \quad (z + \sqrt{z^2 - 1})^n = T_n(z) + U_{n-1}(z)\sqrt{z^2 - 1}.$$

It follows that the Chebyshev expansion (3.2.26) formally corresponds to a symmetric Laurent expansion,

$$g(w) = f\left(\tfrac{1}{2}(w + w^{-1})\right) = \sum_{-\infty}^{\infty} a_j w^j; \quad a_{-j} = a_j = \begin{cases} \frac{1}{2}c_j, & \text{if } j > 0; \\ c_0, & \text{if } j = 0. \end{cases}$$

It can be shown, e.g., by the parallelogram law, that $|z+1|+|z-1| = |w|+|w|^{-1}$, Hence, if $R > 1$, $z = \frac{1}{2}(w + w^{-1})$ maps the annulus $\{w : R^{-1} < |w| < R\}$, twice onto an ellipse $\mathcal{E}_R$, determined by the relation,

$$\mathcal{E}_R = \{z : |z-1| + |z+1| \le R + R^{-1}\}, \tag{3.2.28}$$

with foci at 1 and -1. The axes are, respectively, $R+R^{-1}$ and $R-R^{-1}$, and hence R *is the sum of the semi-axes.*

Note that, as $R \to 1$, the ellipse degenerates into the interval $[-1, 1]$. As $R \to \infty$, it becomes close to the circle $|z| < \frac{1}{2}R$. It follows from (3.2.27) etc. that this family of confocal ellipses are level curves of $|w| = |z \pm \sqrt{z^2-1}|$. In fact, we can also write,

$$\mathcal{E}_R = \left\{z : 1 \le |z + \sqrt{z^2-1}| \le R\right\}. \tag{3.2.29}$$

Theorem 3.2.4.

Let $f(z)$ be real-valued for $z \in [-1, 1]$, analytic and single-valued for $z \in \mathcal{E}_R$, $R > 1$. Assume that $|f(z)| \le M$ for $z \in \mathcal{E}_R$. Then[26]

$$\left|f(x) - \sum_{j=0}^{n-1} c_j T_j(x)\right| \le \frac{2MR^{-n}}{1-1/R} \quad \textit{for } x \in [-1, 1].$$

Proof. Set as before, $z = \frac{1}{2}(w + w^{-1})$, $g(w) = f\big(\frac{1}{2}(w + w^{-1})\big)$. Then $g(w)$ is analytic in the annulus $R^{-1} + \epsilon \le |w| \le R - \epsilon$, and hence the Laurent expansion (1.2) converges there. In particular it converges for $|w| = 1$, hence the Chebyshev expansion for $f(x)$ converges when $x \in [-1, 1]$.

Set $r = R - \epsilon$. By Cauchy's formula, we obtain, for $j > 0$,

$$|c_j| = 2|a_j| = \left|\frac{2}{2\pi i}\int_{|w|=r} g(w) w^{-(j+1)} dw\right| \le \frac{2}{2\pi}\int_0^{2\pi} M r^{-j-1} r d\phi = 2Mr^{-j}.$$

We then obtain, for $x \in [-1, 1]$,

$$\left|f(x) - \sum_{j=0}^{n-1} c_j T_j(x)\right| = \left|\sum_{n}^{\infty} c_j T_j(x)\right| \le \sum_{n}^{\infty} |c_j| \le 2M \sum_{n}^{\infty} r^{-j} \le 2M \frac{r^{-n}}{1-1/r}.$$

This holds for any $\epsilon > 0$. We can here let $\epsilon \to 0$ and thus replace r by R. ∎

If a Chebyshev expansion converges rapidly, the truncation error is, by and large, determined by the first few neglected terms. As indicated by Figures 3.2.1 and 3.2.5 the error curve is oscillating with slowly varying amplitude in $[-1, 1]$. In contrast, the truncation error of a power series is proportional to a power of x.

Note that $f(z)$ is allowed to have a singularity arbitrarily close to the interval $[-1, 1]$, and the convergence of the Chebyshev expansion will still be exponential, although the exponential rate deteriorates, as $R \downarrow 1$.

[26] A generalization to complex values of x is formulated in Problem 6.

The numerical value of a truncated Chebyshev expansion can be computed by means of **Clenshaw's algorithm** which holds for any sum of the form $S = \sum_{k=1}^{n} c_k\phi_k$, where $\{\phi_k\}$ satisfies a **three term recurrence relation**

Theorem 3.2.5. Clenshaw's algorithm [13]
Suppose that a sequence $\{p_k\}$ satisfies the three term recurrence relation

$$p_{k+1} = \gamma_k p_k - \beta_k p_{k-1}, \quad k = 0 : n-1, \tag{3.2.30}$$

where $p_{-1} = 0$. Then

$$S = \sum_{k=0}^{n} c_k p_k = y_0 p_0$$

where y_0 is obtained by the recursion

$$\begin{aligned} &y_{n+1} = 0, \qquad y_n = c_n, \\ &y_k = c_k + \gamma_{k-1} y_{k+1} - \beta_k y_{k+2}, \quad k = n-1 : -1 : 0. \end{aligned} \tag{3.2.31}$$

Proof. Write the recursion (3.2.30) in matrix form as

$$\begin{pmatrix} 1 & & & & \\ -\gamma_0 & 1 & & & \\ \beta_1 & -\gamma_1 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & -\gamma_{n-1} & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} p_0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix},$$

or $Lp = g$, $g = p_0 e_1$, where L is unit lower triangular and e_1 is the first column of the unit matrix. Then

$$S = c^T p = c^T L^{-1} g = g^T (L^T)^{-1} c = g^T y,$$

where y is the solution to the upper triangular system $L^T y = c$. Solving this by backsubstitution we get the recursion (3.2.31). □

It can proved that Clenshaw's algorithm is componentwise backward stable with respect to the data γ_k and β_k; see Smoktunowicz [41].

Clenshaw's algorithm can also be applied to series of Legendre functions, Bessel functions, Coulomb wave functions etc., because they satisfy recurrence relations of this type, where the α_k, γ_k depend on x; see Abramowitz and Stegun [1] or any text on special functions. Other applications are the case when the ϕ_k are the denominators or numerators of the approximants of a *continued fraction*; see Sec. 3.5.1

Important properties of trigonometric functions and Fourier series can be reformulated in the terminology of Chebyshev polynomials. For example, they satisfy certain orthogonality relations; see Sec. 4.5.5. Also results like (3.2.8) concerning

how the rate of decrease of the coefficients or the truncation error of a Fourier series, is related to the smoothness properties of its sum, can be translated to Chebyshev expansions. So, even *if F is not analytic, a Chebyshev expansion converges under amazingly general conditions* (unlike a power series), but the convergence is much slower than exponential. A typical result reads: *if $f \in C^k[-1,1]$, $k > 0$, there exists a bound for the truncation error that decreases uniformly like $O(n^{-k}\log n)$.* Sometimes convergence acceleration can be successfully applied to such series.

3.2.4 Perturbation Expansions

In the equations of applied mathematics it is often possible to identify a small dimensionless parameter (say) ϵ, $\epsilon \ll 1$. The case when $\epsilon = 0$ is called the *reduced problem* or the unperturbed case, and one asks for a **perturbation expansion**, i.e. an expansion of the solution of the perturbed problem into powers of the perturbation parameter ϵ. In many cases it can be proved that the expansion has the form $c_0+c_1\epsilon+c_2\epsilon^2+\ldots$, but there are also important cases, where the expansion contains fractional or a few negative powers.

In this subsection, we consider an analytic equation $\phi(z,\epsilon) = 0$ and seek expansions for the roots $z_i(\epsilon)$ in powers of ϵ. This has some practical interest in its own right, but it is mainly to be considered as a preparation for more interesting applications of perturbation methods to more complicated problems. A simple perturbation example for a *differential equation* is given in Problem 10.

If $z_i(0)$ is a simple root, i.e. if $\partial\phi/\partial z \neq 0$, for $(z,\epsilon) = (z_i(0),0)$, then a theorem of complex analysis tells us that $z_i(\epsilon)$ is an analytic function in a neighborhood of the origin, hence the expansion

$$z_i(\epsilon) - z_i(0) = c_1\epsilon + c_2\epsilon^2 + \ldots$$

has a positive (or infinite) radius of convergence. We call this a **regular perturbation problem**. The techniques of power series reversion, presented in Sec. 3.1.4, can often be applied after some preparation of the equation. Computer algebra systems are also used in perturbation problems, if expansions with many terms are needed.

Example 3.2.5.

We shall expand the roots of $\phi(z,\epsilon) \equiv \epsilon z^2 - z + 1 = 0$ into powers of ϵ. The reduced problem $-z+1=0$ has only one finite root $z_1(0) = 1$. Set $z = 1+x\epsilon,\ x = c_1 + c_2\epsilon + c_3\epsilon^2 + \ldots$. Then $\phi(1+x\epsilon,\ \epsilon)/\epsilon = (1+x\epsilon)^2 - x = 0$, i.e.

$$(1 + c_1\epsilon + c_2\epsilon^2 + \ldots)^2 - (c_1 + c_2\epsilon + c_3\epsilon^2 + \ldots) = 0.$$

Matching the coefficients of $\epsilon^0,\ \epsilon^1, \epsilon^2$, we obtain the system

$$\begin{aligned} 1 - c_1 = 0 &\ \Rightarrow\ c_1 = 1; \\ 2c_1 - c_2 = 0 &\ \Rightarrow\ c_2 = 2; \\ 2c_2 + c_1^2 - c_3 = 0 &\ \Rightarrow\ c_3 = 5; \end{aligned}$$

hence $z_1(\epsilon) = 1 + \epsilon + 2\epsilon^2 + 5\epsilon^3 + \ldots$.

Now, the easiest way to obtain the expansion for the second root $z_2(\epsilon)$, is to use the fact that the sum of the roots of the quadratic equation equals ϵ^{-1}, hence $z_2(\epsilon) = \epsilon^{-1} - 1 - \epsilon - 2\epsilon^2 + \ldots$.

Note the appearance of the term ϵ^{-1}. This is due to a characteristic feature of this example. The degree of the polynomial is lower for the reduced problem than it is for $\epsilon \neq 0$; one of the roots escapes to ∞ as $\epsilon \to 0$. This is an example of a **singular perturbation** problem, an important type of problem for differential equations; see Problem 10.

If $\partial\phi/\partial z = 0$, for some z_i, the situation is more complicated; z_i is a multiple root, and the expansions look differently. If $z_i(0)$ is a k-fold root then there may exist an expansion of the form

$$z_i(\epsilon) = c_0 + c_1\epsilon^{1/k} + c_2(\epsilon^{1/k})^2 + \ldots$$

for each of the k roots of ϵ, but this is not always the case. See (3.2.32) below, where the expansions are of a different type. *If one tries to determine the coefficients in an expansion of the wrong form, one usually runs into contradictions,* but the question about the right form of the expansions still remains.

The answers are given by the classical theory of *algebraic functions*, where Riemann surfaces and Newton polygons are two of the key concepts, see, e.g., Bliss [5]. We shall, for several reasons, not use this theory here. One reason is that it seems hard to generalize some of the methods of algebraic function theory to more complicated equations, such as differential equations. We shall instead use a general **balancing procedure**, recommended in Lin and Segel [31, Sec. 9.1], where it is applied to singular perturbation problems for differential equations too.

The basic idea is very simple: each term in an equation behaves like some power of ϵ. *The equation cannot hold, unless there is a β, such that a pair of terms of the equation behave like $A\epsilon^\beta$, (with different values of A), and the ϵ-exponents of the other terms are larger than or equal to β.* (Recall that larger exponents make smaller terms.)

Let us return to the previous example. Although we have already determined the expansion for $z_2(\epsilon)$ (by a trick that may not be useful for other problems than single analytic equations), we shall use this task to illustrate the balancing procedure. Suppose that

$$z_2(\epsilon) \sim A\epsilon^\alpha, \ (\alpha < 0).$$

The three terms of the equation $\epsilon z^2 - z + 1 = 0$ then get the exponents

$$1 + 2\alpha, \ \alpha, \ 0.$$

Try the first two terms as the candidates for being the dominant pair. Then $1+2\alpha = \alpha$, hence $\alpha = -1$. The three exponents become $-1, \ -1, \ 0$. Since the third exponent is larger than the exponent of the candidates, this choice of pair seems possible, but we have not shown that it is the only possible choice.

Now try the first and the third terms as candidates. Then $1 + 2\alpha = 0$, hence $\alpha = -\frac{1}{2}$. The exponent of the non-candidate is $-\frac{1}{2} \leq 0$; this candidate pair is thus

impossible. Finally, try the second and the third terms. Then $\alpha = 0$, but we are only interested in negative values of α.

The conclusion is that we can try coefficient matching in the expansion $z_2(\epsilon) = c_{-1}\epsilon^{-1} + c_0 + c_1\epsilon + \ldots$. We don't need to do it, since we know the answer already, but it indicates how to proceed in more complicated cases.

Example 3.2.6.

First consider the equation $z^3 - z^2 + \epsilon = 0$. The reduced problem $z^3 - z^2 = 0$ has a single root, $z_1 = 1$, and a double root, $z_{2,3} = 0$. No root has escaped to ∞. By a similar coefficient matching as in the previous example we find that $z_1(\epsilon) = 1 - \epsilon - 2\epsilon^2 + \ldots$. For the double root, set $z = A\epsilon^\beta$, $\beta > 0$. The three terms of the equation obtain the exponents $3\beta,\ 2\beta, 1$. Since 3β is dominated by 2β we conclude that $2\beta = 1$, i.e.$\beta = 1/2$,

$$z_{2,3}(\epsilon) = c_0\epsilon^{1/2} + c_1\epsilon + c_2\epsilon^{3/2} + \ldots.$$

By matching the coefficients of $\epsilon,\ \epsilon^{3/2},\ \epsilon^2$, we obtain the system

$$\begin{aligned} -c_0^2 + 1 = 0 \ &\Rightarrow c_0 = \pm 1, \\ -2c_0c_1 + c_0^3 = 0 \ &\Rightarrow c_1 = \tfrac{1}{2}, \\ -2c_0c_2 - c_1^2 + 2c_0^2c_1 + c_1c_0^2 = 0 \ &\Rightarrow c_2 = \pm\tfrac{5}{8}, \end{aligned}$$

hence $z_{2,3}(\epsilon) = \pm\epsilon^{1/2} + \frac{1}{2}\epsilon \pm \frac{5}{8}\epsilon^{3/2} + \ldots$.

There are, however, equations with a double root, where the perturbed pair of roots do not behave like $\pm c_0\epsilon^{1/2}$ as $\epsilon \to 0$. In such cases the balancing procedure may help. Consider the equation

$$(1+\epsilon)z^2 + 4\epsilon z + \epsilon^2 = 0. \tag{3.2.32}$$

The reduced problem reads $z^2 = 0$, with a double root. Try $z \sim A\epsilon^\alpha$, $\alpha > 0$. The exponents of the three terms become $2\alpha,\ \alpha + 1,\ 2$. We see that $\alpha = 1$ makes the three exponents all equal to 2; this is fine. So, set $z = \epsilon y$. The equation reads, after division by ϵ^2, $(1+\epsilon)y^2 + 4y + 1 = 0$, hence $y(0) = a \equiv -2 \pm \sqrt{3}$. Coefficient matching yields the result

$$z = \epsilon y = a\epsilon + (-a + a^2/2)\epsilon^2 + \ldots, \quad a = -2 \pm \sqrt{3},$$

where all exponents are natural numbers.

If ϵ is small enough, the last term included can serve as an error estimate. A more reliable error estimate (or even an error bound) can be obtained by inserting the truncated expansion into the equation. It shows that *the truncated expansion satisfies a modified equation exactly.* The same idea was indicated for a differential equation in Example 3.1.2; see also Problem 10, and it can be applied to equations of many other types.

3.2.5 Ill-Conditioned Series

Slow convergence is not the only numerical difficulty that occurs in connection with infinite series. There are also series with oscillating terms and a complicated type of catastrophic cancellation. The size of some terms are many orders of magnitude larger than the sum of the series. Small relative errors in the computation of the large terms lead to a large relative error in the result. We call such a series **ill-conditioned**.

An important class of sequences $\{c_n\}$, are known as **completely monotonic**.

Definition 3.2.6.

A sequence $\{u_n\}$ *is completely monotonic for* $n \geq a$ *iff*

$$u_n \geq 0, \quad (-\Delta)^j u_n \geq 0, \quad \forall j \geq 0,\ n \geq a,\ \text{(integers)}.$$

Such series have not been subject to many systematic investigations. One simply tries to avoid them. For the important "special functions" of Applied Mathematics, such as Bessel Functions, confluent hypergeometric functions etc., there usually exists *expansions into descending powers of* z that can be useful, when $|z| \gg 1$ and the usual series, in *ascending* powers, are divergent or ill-conditioned. Another possibility is to use *multiple precision* in computations with ill-conditioned power series; this is relatively expensive and laborious (but the difficulties should not be exaggerated). There are, however, also other, less known, possibilities that will now be exemplified. The subject is still open for new fresh ideas, and we hope that the following pages and the related problems at the end of the section will stimulate some readers to thinking about it.

First, we shall consider power series of the form

$$\sum_{n=0}^{\infty} \frac{(-x)^n c_n}{n!}, \tag{3.2.33}$$

where $x \gg 1$, although not so large that there is risk for overflow. We assume that the coefficients c_n are positive and slowly varying (relatively to $(-x)^n/n!$). The ratio of two consecutive terms is

$$\frac{c_{n+1}}{c_n}\frac{-x}{n+1} \approx \frac{-x}{n+1}.$$

We see that the series converges for all x, and that the magnitude increases iff $n+1 < |x|$. *The term of largest magnitude is thus obtained for* $n \approx |x|$. Denote its magnitude by $M(x)$. Then, for $x \gg 1$, the following type of approximations can be used, e.g., for crude estimates of the number of terms needed, the arithmetic precision that is to be used etc. in computations related to ill-conditioned power series:

$$M(x) \approx c_x e^x (2\pi x)^{-1/2}, \quad \text{i.e., } \log_{10} M(x)/c_0 \approx 0.43x - \tfrac{1}{2}\log_{10}(2\pi x). \tag{3.2.34}$$

This follows from the classical **Stirling's formula**,

$$x! \sim (x/e)^x \sqrt{2\pi x}, \quad x \gg 1, \tag{3.2.35}$$

that gives $x!$ with a relative error that is about $\frac{1}{12x}$. You find a proof of this in most textbooks on calculus. It will often be used in the rest of this book. A more accurate and general version is given in Example 3.3.12 together with a few more facts about the gamma function, $\Gamma(z)$, an analytic function that interpolates the factorial, $\Gamma(n+1) = n!$ if n is a natural number. Sometimes the notation $z!$ is used instead of $\Gamma(z+1)$ also if z is not an integer.

There exist **preconditioners**, i.e. transformations that can convert classes of ill-conditioned power series (with accurately computable coefficients) to more well-conditioned problems. One of the most successful preconditioners known to the authors is the following:[27]

$$\sum_{n=0}^{\infty} \frac{(-x)^n c_n}{n!} = e^{-x} \sum_{m=0}^{\infty} \frac{x^m (-\Delta)^m c_0}{m!}. \tag{3.2.36}$$

This identity is proved in Example . A hint to a shorter proof is given in Problem 3.21.

Example 3.2.7.

Consider the function

$$F(x) = \frac{1}{x} \int_0^x \frac{1 - e^{-t}}{t}\, dt = 1 - \frac{x}{2^2 \cdot 1!} + \frac{x^2}{3^2 \cdot 2!} - \dots,$$

i.e. $F(x)$ is a particular case of (3.2.33) with $c_n = (n+1)^{-2}$. We shall look at three methods of computing $F(x)$ for $x = 10 : 10 : 50$, named A, B, C. $F(x)$ decreases smoothly from 0.2880 to 0.0898. The computed values of $F(x)$ are denoted $FA(x), FB(x), FC(x)$.

The coefficients c_n, $n = 0 : 119$, are given in IEEE floating point, double precision. The table of results show that, except for $x = 50$, 120 terms is much more than necessary for the rounding of the coefficients to become the dominant error source.

x	10	20	30	40	50
$F(x) \approx$	0.2880	0.1786	0.1326	0.1066	0.0898
lasttermA	$1 \cdot 10^{-82}$	$8 \cdot 10^{-47}$	$7 \cdot 10^{-26}$	$6 \cdot 10^{-11}$	$2 \cdot 10^{1}$
$M(x; A)$	$3 \cdot 10^{1}$	$1 \cdot 10^{5}$	$9 \cdot 10^{8}$	$1 \cdot 10^{13}$	$1 \cdot 10^{17}$
$\lvert FA(x) - F(x)\rvert$	$2 \cdot 10^{-15}$	$5 \cdot 10^{-11}$	$2 \cdot 10^{-7}$	$3 \cdot 10^{-3}$	$2 \cdot 10^{1}$
lasttermB	$4 \cdot 10^{-84}$	$1 \cdot 10^{-52}$	$4 \cdot 10^{-36}$	$2 \cdot 10^{-25}$	$2 \cdot 10^{-18}$
$M(x; B)$	$4 \cdot 10^{-2}$	$2 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$7 \cdot 10^{-3}$	$5 \cdot 10^{-3}$
$\lvert FC(x) - FB(x)\rvert$	$7 \cdot 10^{-9}$	$2 \cdot 10^{-14}$	$6 \cdot 10^{-17}$	0	$1 \cdot 10^{-16}$

[27] The notation $\Delta^m c_n$ for high order differences was introduced in Sec. 1.1.3.

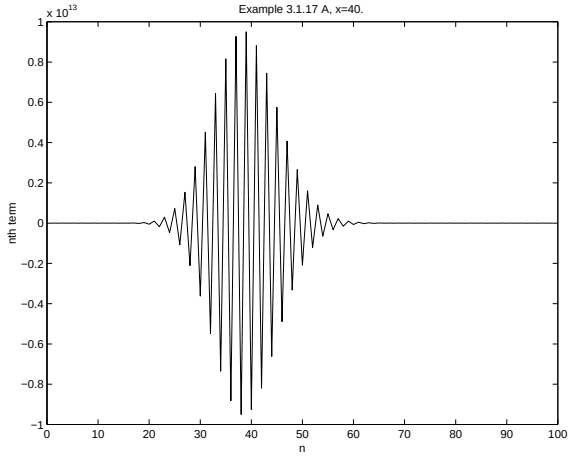


Figure 3.2.2. *Example* 3.2.5 *A: Terms of* (3.2.33), $c_n = (n+1)^{-2}$, $x = 40$, *no preconditioner. Note the scale, and look also in the table. Since the largest term is* 10^{13}, *it is no surprise that the relative error of the sum is not better than* 0.03, *in spite that double precision floating point has been used.*

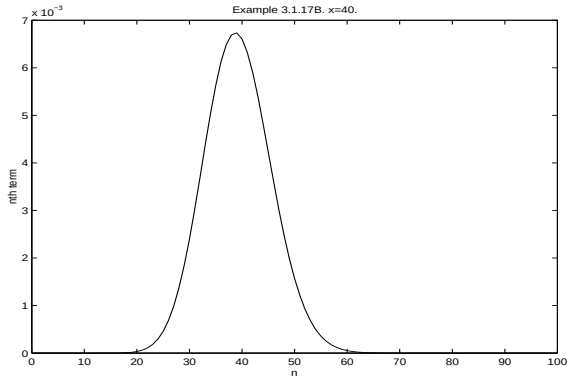


Figure 3.2.3. *Example* 3.2.5 *B:* $c_n = (n+1)^{-2}$, $x = 40$, *with the preconditioner in* (3.2.36). *The terms of the right hand side, including the factor* e^{-x}, *becomes a so-called* **bell sum**; *the largest term is about* $7\,10^{-3}$. *The computed sum is correct to 16 decimal places.*

A We use (3.2.33) without preconditioner. $M(x;A)$ is the largest magnitude of the terms of the expansion. $M(x;A)\cdot 10^{-16}$ tells the order of magnitude of the effect of the rounding errors on the computed value $FA(x)$. Similarly, the truncation error is crudely estimated by lasttermA. See also Figure 3.1.6.

B. We use the preconditioner (3.2.36). In this example $c_n = (n+1)^{-2}$. In Problem 3.2.2(c) we find the following explicit expressions, related to the series on

the right hand side of the preconditioner for this example.

$$(-\Delta)^m c_0 = (-\Delta)^m c_n|_{n=0} = c_0(-\Delta)^m x^{-2}|_{x=1} = \frac{c_0}{m+1}\sum_{k=0}^{m}\frac{1}{k+1},$$

$$F(x) = c_0 e^{-x}\sum_{m=0}^{\infty}\frac{x^m}{m!}\frac{1}{(m+1)}\sum_{k=0}^{m}\frac{1}{k+1}. \tag{3.2.37}$$

Note that $(-\Delta)^m c_0$ is positive and smoothly decreasing; (This is not a special feature only for this example, but it holds for sequences $\{c_n\}$, which are completely monotonic.)

The largest term is thus smaller than the sum, and the series (3.2.37) is **well-conditioned**. It can be shown that, if $x \gg 1$, the mth term is approximately proportional to the value at m of the normal probability density with mean x and standard deviation equal to $\sqrt{x}$; note the resemblance to a Poisson distribution. Multiple precision is not needed here.

$M(x;B)$ and lasttermB are defined analogously to $M(x;A)$ and lasttermA, The B-values are very different from the A-values. In fact they indicate that *all values of $FB(x)$, referred to in the table, give $F(x)$ to full accuracy.*

C. The following expression for $F(x)$,

$$xF(x) \equiv \sum_{n=1}^{\infty}\frac{(-x)^n}{nn!} = -\gamma - \ln x - E_1(x); \quad E_1(x) = \int_x^{\infty}\frac{e^{-t}}{t}\,dt, \tag{3.2.38}$$

is valid for all $x > 0$; see [1, **5.1.11**]. $E_1(x)$ is known as the **exponential integral**, and

$$\gamma = 0.57721\,56649\,01532\,86061\ldots$$

is the well known **Euler's constant**. In the next section, an asymptotic expansion for $E_1(x)$ for $x \gg 1$ is derived, the first two terms of which are used here in the computation of $F(x;C)$ for the table above.

$$E_1(x) \approx e^{-x}(x^{-1} - x^{-2}), \quad x \gg 1.$$

This approximation is the dominant part of the error of $F(x;C)$; it is less than $e^{-x}2x^{-4}$. $F(x;C)$ gives full accuracy for (say) $x > 25$.

More examples of sequences, for which rather simple explicit expressions for the high order differences are known, are given in Problem 3.21. The **Kummer confluent hypergeometric function** $M(a,b,x)$ was defined in (3.1.14). We have

$$M(a,b,x) = 1 + \sum_{n=1}^{\infty}\frac{(-x)^n c_n}{n!}, \quad c_n = c_n(a,b) = \frac{a(a+1)\ldots(a+n-1)}{b(b+1)\ldots(b+n-1)}.$$

In our context $b > a > 0$, $n > 0$. The oscillatory series for $M(a,b,-x)$, $x > 0$, is ill-conditioned if $x \gg 1$.

By Problem 3.21, $(-\Delta)^n c_0(a,b) = c_n(b-a,b) > 0$, $n > 0$, hence the preconditioner (3.2.36) yields the equation

$$M(a,b,-x) = e^{-x} M(b-a,b,x), \tag{3.2.39}$$

where the series on the right hand side has positive terms, because $b-a > 0$, $x > 0$, and is a well-conditioned *bell sum*. The mth term has typically a sharp maximum for $m \approx x$; compare Figure 3.2.7. Equation (3.2.39) is in the theory of the confluent hypergeometric functions known as **Kummer's first identity**. It is emphasized here, because several functions with famous names of their own are particular cases of the Kummer function. These share the numerous useful properties of Kummer's function, e.g., the above identity; see the theory in Lebedev [30, Secs. 9.9–9.14][28] and the formulas in [1, Ch. 13] in particular Table 13.6 of special cases. An important example is the error function (see Example 1.2.3) that can be expressed in terms of Kummer's confluent hypergeometric as .

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt = \frac{2x}{\sqrt{\pi}} M\left(\frac{1}{2}, \frac{3}{2}, -x^2\right). \tag{3.2.40}$$

If we cannot find explicit expressions for high order differences, we can make a *difference scheme* by the recurrence $\Delta^{m+1} c_n = \Delta^m c_{n+1} - \Delta^m c_n$. Unfortunately the computation of a difference scheme suffers from numerical instability. Suppose that the absolute errors of the c_n are bounded by ϵ. Then the absolute errors can become as large as 2ϵ in the first differences, 4ϵ in the second differences etc. More generally, the absolute errors of $(-\Delta)^m c_n$ can become as large as $2^m\epsilon$. (You find more about this in Examples 3.2.2 and 3.2.3.) In connection with ill-conditioned series, this instability is much more disturbing than in the traditional applications of difference schemes to interpolation etc., where m is seldom much larger than 10. Recall that $m \approx x$ for the largest term of the preconditioned series. So, if $x > 53$ even this term may not have any correct bit if IEEE double precision is used, and many terms are needed after this.

So, during the computation of the new coefficients, $(-\Delta)^m c_n$, (only once for the function F, and with double accuracy in the results), the old coefficients c_n must be available with multiple accuracy, and multiple precision must be used in the computation of their difference scheme. Otherwise, we cannot evaluate the series with a decent accuracy for much larger values of x than we could have done without preconditioning. Note, however, that if satisfactory coefficients have been obtained for the preconditioned series, double precision is sufficient when the series is evaluated for large values of x. (It is different for method A above.)

Let $F(x)$ be the function that we want to compute for $x \gg 1$, where it is defined by an ill-conditioned power series $F_1(x)$. A more general preconditioner can be described as follows. Try to find a power series $P(x)$ with positive coefficients such that the power series $P(x)F_1(x)$ has less severe cancellations than than $F_1(x)$.

In order to distinguish between the algebraic manipulation and the numerical evaluation of the functions defined by these series, we introduce the indeterminate

[28] Unfortunately, the formulation of Kummer's first identity in [30, Eqn. (9.11.2)] contains a serious sign error.

x and describe a **more general preconditioner** as follows:

$$\mathbf{F_2^*(x)} = \mathbf{P(x)} \cdot \mathbf{F_1(x)}; \qquad F_2(x) = F_2^*(x)/P(x). \tag{3.2.41}$$

The second statement is a usual scalar evaluation (no bold-face). Here $P(x)$ may be evaluated by some other method than the power series, if it is more practical. If $P(x) = e^x$, and $F_1(x)$ is the series defined by (3.2.33), then it can be shown that $F_2(x)$ is mathematically equivalent to the right hand side of (3.2.36); see Example 3.2.1. In these cases $F_2(x)$ has positive coefficients.

If, however, $F_1(x)$ has a positive zero, this is also a zero of $F_2^*(x)$, and hence it is impossible that all coefficients of the series $\mathbf{F_2^*(x)}$ have the same sign. Nevertheless, the following example shows that the preconditioner (3.2.41) can sometimes be successfully used in such a case too.

Table 3.2.1.

1	x	10	20	30	40	50
2	$J_0(x) \approx$	$-2 \cdot 10^{-1}$	$2 \cdot 10^{-1}$	$-9 \cdot 10^{-2}$	$7 \cdot 10^{-3}$	$6 \cdot 10^{-2}$
3	$N1(x)$	26	41	55	69	82
4	$J(x; N1) - J_0(x)$	$9 \cdot 10^{-14}$	$3 \cdot 10^{-10}$	$-2 \cdot 10^{-6}$	$-1 \cdot 10^{-1}$	$-2 \cdot 10^{2}$
5	$N2(x)$	16	26	36	46	55
6	$IJ(x; N2) \approx$	$-7 \cdot 10^{2}$	$7 \cdot 10^{6}$	$-7 \cdot 10^{10}$	$1 \cdot 10^{14}$	$2 \cdot 10^{19}$
7	$I_0(x) \approx$	$3 \cdot 10^{3}$	$4 \cdot 10^{7}$	$8 \cdot 10^{11}$	$1 \cdot 10^{16}$	$3 \cdot 10^{20}$
8	$IJ(x)/I_0(x) - J_0(x)$	$3 \cdot 10^{-17}$	$2 \cdot 10^{-14}$	$3 \cdot 10^{-13}$	$-5 \cdot 10^{-12}$	$2 \cdot 10^{-10}$

Example 3.2.8.

The two functions

$$J_0(x) = \sum_{n=0}^{\infty} (-1)^n \frac{(x^2/4)^n}{(n!)^2}, \qquad I_0(x) = \sum_{n=0}^{\infty} \frac{(x^2/4)^n}{(n!)^2},$$

are examples of Bessel functions of the first kind; I_0 is nowadays called a modified Bessel function. $J_0(x)$ is oscillatory and bounded, while $I_0(x) \sim e^x/\sqrt{2\pi x}$ for $x \gg 1$. Since all coefficients of I_0 are positive, we shall set $P = I_0$, $F_1 = J_0$, and try

$$\mathbf{F_2^*(x)} = \mathbf{IJ(x)} \equiv \mathbf{I_0(x)} \cdot \mathbf{J_0(x)}, \quad F_2(x) = F_2^*(x)/I_0(x),$$

as a preconditioner for the power series for $J_0(x)$, which is ill-conditioned if $x \gg 1$. In Table 3.2.2 line 2 and line 7 are obtained from the fully accurate built-in functions for $J_0(x)$ and $I_0(x)$. $J(x; N1)$ is computed in IEEE double precision from $N1$ terms of the above power series for $J_0(x)$. $N1 = N1(x)$ is obtained by a termination criterion that should give full accuracy or, if the estimate of the effect of the rounding error is bigger than 10^{-16}, the truncation error should be smaller than this estimate. We omit the details; see also Problem 12 (d).

The coefficients of $\mathbf{IJ(x)}$ are obtained from the second expression for γ_m given in Problem 12 (c). $N2 = N2(x)$ is the number of terms used in the expansion

of **IJ**(**x**), by a termination criterion, similar to the one described for $J(x; N1)$. Compared to line 4, line 8 is a remarkable improvement, obtained without the use of multiple precision.

For series of the form

$$\sum_{n=0}^{\infty} a_n \frac{(-x^2)^n}{(2n)!}$$

one can generate a preconditioner from $P(x) = \cosh x$. This can also be applied to $J_0(x)$ and other Bessel functions; see Problem 12 (e).

3.2.6 Divergent or Semiconvergent Series

That a series is convergent is no guarantee that it is numerically useful. In this section, we shall see examples of the reverse situation: a divergent series can be of use in numerical computations. This sounds strange, but it refers to series where the size of the terms decreases rapidly at first and increases later, and where an error bound (see Figure 3.2.4), can be obtained in terms of the first neglected term. Such series are sometimes called **semiconvergent**. An important subclass are the *asymptotic* series; see below.

Example 3.2.9.

We shall derive a semiconvergent series for the computation of Euler's function

$$f(x) = e^x E_1(x) = e^x \int_x^{\infty} e^{-t} t^{-1}\, dt = \int_0^{\infty} e^{-u} (u + x)^{-1}\, du$$

for large values of x. (The second integral was obtained from the first by the substitution $t = u + x$.) The expression $(u + x)^{-1}$ should first be expanded in a geometric series with remainder term, valid even for $u > x$,

$$(u + x)^{-1} = x^{-1}(1 + x^{-1}u)^{-1} = x^{-1} \sum_{j=0}^{n-1} (-1)^j x^{-j} u^j + (-1)^n (u + x)^{-1} (x^{-1}u)^n$$

We shall frequently use the well known formula

$$\int_0^{\infty} u^j e^{-u}\, du = j! = \Gamma(j + 1).$$

We write $f(x) = S_n(x) + R_n(x)$, where

$$S_n(x) = x^{-1} \sum_{j=0}^{n-1} (-1)^j x^{-j} \int_0^{\infty} u^j e^{-u} du = \frac{1}{x} - \frac{1!}{x^2} + \frac{2!}{x^3} - \ldots + (-1)^{n-1} \frac{(n-1)!}{x^n},$$

$$R_n(x) = (-1)^n \int_0^{\infty} (u + x)^{-1} \Big(\frac{u}{x}\Big)^n e^{-u} du.$$

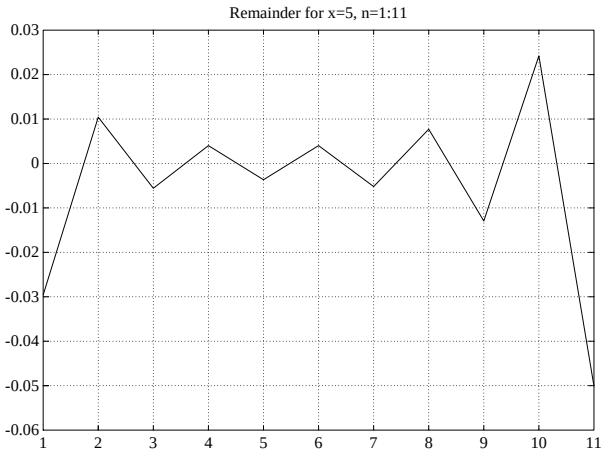


Figure 3.2.4. *The first 11 error estimates of the semiconvergent series of Example* 3.2.7*; see* (3.2.43)*. The smallest actual error is only 5% of the smallest error estimate.*

The terms in $S_n(x)$ qualitatively behave as in Figure 3.2.4. The ratio between the last term in S_{n+1} and the last term in S_n is

$$-\frac{n!}{x^{n+1}}\frac{x^n}{(n-1)!} = -\frac{n}{x}, \tag{3.2.42}$$

and since the absolute value of that ratio for fixed x is unbounded as $n \to \infty$, the sequence $\{S_n(x)\}_{n=1}^{\infty}$ *diverges for every positive* x. But since sign $R_n(x) = (-1)^n$ for $x > 0$, it follows from Theorem 3.1.4 that

$$f(x) = \frac{1}{2}\Big(S_n(x) + S_{n+1}(x)\Big) \pm \frac{1}{2}\frac{n!}{x^{n+1}}. \tag{3.2.43}$$

The idea is now to choose n so that the estimate of the remainder is as small as possible. According to (3.2.42), this happens when n is equal to the integer part of x. For $x = 5$ we choose $n = 5$,

$$S_5(5) = 0.2 - 0.04 + 0.016 - 0.0096 + 0.00768 = 0.17408,$$
$$S_6(5) = S_5(5) - 0.00768 = 0.16640,$$

which gives $f(5) = 0.17024 \pm 0.00384$. The correct value is 0.17042, so the actual error is only 5% of the error bound.

For larger values of x the accuracy attainable increases. One can show that the bound for the *relative* error using the above computational scheme decreases approximately as $(\pi \cdot x/2)^{1/2}e^{-x}$; an extremely good accuracy for large values of x, if one stops at the smallest term. It can even be improved further, by the use of

the convergence acceleration techniques presented in Sec. 3.4, notably the *repeated averages* algorithm, also known as the **Euler transformation**; see Sec. 3.4.3. The algorithms for the transformation of a power series into a rapidly convergent continued fraction, mentioned in Sec. 3.5.1, can also be successfully applied to this example and to many other divergent expansions.

One can derive the same series expansion as above by repeated integration by parts. This is often a good way to derive numerically useful expansions, convergent or semi-convergent, with a remainder in the form of an integral. For convenient reference, we formulate this as a lemma that is easily proved by induction and the mean value theorem of integral calculus. See Problem 13 for applications.

Lemma 3.2.7. *Repeated Integration by Parts.*

Let $F \in C^p(a,b)$, let G_0 be a piecewise continuous function, and let $G_0, G_1, \ldots$ be a sequence of functions such that $G'_{j+1}(x) = G_j(x)$ with suitably chosen constants of integration. Then

$$\int_a^b F(t)G_0(t)\,dt = \sum_{j=0}^{p-1}(-1)^j F^{(j)}(t)G_{j+1}(t)\Big|_{t=a}^{b} + (-1)^p \int_a^b F^{(p)}(t)G_p(t)\,dt.$$

The sum is the "expansion", and the last integral is the "remainder". If $G_p(t)$ has a constant sign in (a,b), the remainder term can also be written in the form

$$(-1)^p F^{(p)}(\xi)(G_{p+1}(b) - G_{p+1}(a)), \quad \xi \in (a,b).$$

The expansion in Lemma 3.2.7 is valid as an *infinite* series, if and only if the remainder tends to 0 as $p \to \infty$. Even if the sum converges as $p \to \infty$, it may converge to the wrong result.

The series in Example 3.2.9 is an expansion in *negative* powers of x, with the property that for all n, the remainder, when $x \to \infty$, approaches zero faster than the last included term. Such an expansion is said to **represent** $f(x)$ **asymptotically** as $x \to \infty$. Such an **asymptotic series** can be either convergent or divergent (semi-convergent). In many branches of applied mathematics, divergent asymptotic series are an important aid, though they are often needlessly surrounded by an air of mysticism.

It is important to appreciate that *an asymptotic series does not define a sum uniquely.* For example $f(x) = e^{-x}$ is asymptotically represented by the series $\sum 0x^{-j}$, as $x \to \infty$. So e^{-x}, (and many other functions), can therefore be added to the function, for which the expansion was originally obtained.

Asymptotic expansions are not necessarily expansions into negative powers of x. An expansion into *positive* powers of $x - a$,

$$f(x) \sim \sum_{\nu=0}^{n-1} c_\nu (x-a)^\nu + R_n(x),$$

represents $f(x)$ asymptotically when $x \to a$ if

$$\lim_{x \to a} (x-a)^{-(n-1)} R_n(x) = 0.$$

Asymptotic expansions of the error of a numerical method into positive powers of a step length h are of great importance in the more advanced study of numerical methods. Such expansions form the basis of simple and effective acceleration methods for improving numerical results; see Sec. 3.4.

Review Questions

1. Give the Cauchy formula for the coefficients of Taylor and Laurent series, and describe the Cauchy–FFT method. Give the formula for the coefficients of a Fourier series. For which of the functions in Table 3.1.1 does also another Laurent expansion exist?
2. Describe by an example the balancing procedure that was mentioned in the subsection about perturbation expansions.
3. Define the Chebyshev polynomials, and tell some interesting properties of these and of Chebyshev expansions. For example, what do you know about the speed of convergence of a Chebyshev expansion for various classes of functions? (The detailed expressions are not needed.)
4. Describe and exemplify, what is meant by an ill-conditioned power series and a preconditioner for such a series.
5. Define what is meant, when one says that the series $\sum_0^\infty a_n x^{-n}$

   (a) converges to a function $f(x)$ for $x \geq R$;

   (b) represents a function $f(x)$ asymptotically as $x \to \infty$.

   (c) Give an example of a series that represents a function asymptotically as $x \to \infty$, although it diverges for every finite positive x.

   (d) What is meant by semi-convergence? Say a few words about termination criteria and error estimation.

Problems and Computer Exercises

1. Some of the functions appearing in Table 3.1.1, in Problem 6, and in other examples and problems are *not single-valued* in the complex plane. Brush up your Complex Analysis, and find out how to define the branches, where these expansions are valid, and (if necessary) define cuts in the complex plane that must not be crossed. It turns out not to be necessary for these expansions. Why?

   (a) If you have access to programs for functions of complex variables (or to commands in some package for interactive computation), find out the conventions used for functions like square root, logarithm, powers, arctan etc.

If the manual does not give enough detail, invent numerical tests, both with strategically chosen values of z and with random complex numbers in some appropriate domain around the origin. For example, do you obtain

$$\ln\left(\frac{z+1}{z-1}\right) - \ln(z+1) + \ln(z-1) = 0, \quad \forall z?$$

Or, what values of $\sqrt{z^2-1}$ do you obtain for $z = \pm i$? What values should you obtain, if you want the branch which is positive for $z > 1$?

(b) What do you obtain, if you apply Cauchy's coefficient formula or the Cauchy–FFT method to find a Laurent expansion for $\sqrt{z}$? Note that $\sqrt{z}$ is analytic everywhere in an annulus, but that does not help. The expansion is likely to become weird. Why?

2. (a) Apply (on a computer) the Cauchy–FFT method to find the Maclaurin coefficients a_n of (say) e^z, $\ln(1-z)$ and $(1+z)^{1/2}$. Make experiments with different values of r and N, and compare with the exact coefficients. This presupposes that you have access to good programs for complex arithmetic and FFT.

Try to summarize your experiences of how the error of a_n depends on $r,\ N$. You may find some guidance in Example 3.2.2.

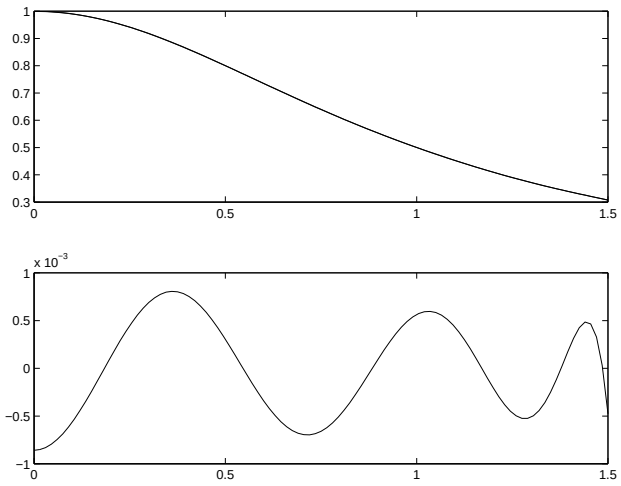


Figure 3.2.5. *Illustrations to Problem* 3 *c.* Upper part: *The function* $f(x) = \frac{1}{1+x^2}$, $x \in [0, 1.5]$. Lower part: *The error of the expansion of* $f(x)$ *in a sum of Chebyshev polynomials* $\{T_n(x/1.5)\}$, $n \le 10$. *The scale is* 10^{-3} *in the lower curve.*

3. (a) Suppose that r is located inside the unit circle; t is real. Show that

$$\frac{1-r^2}{1-2r\cos t + r^2} = 1 + 2\sum_{n=1}^{\infty} r^n \cos nt,$$

$$\frac{2r\sin t}{1-2r\cos t+r^2} = 2\sum_{n=1}^{\infty} r^n \sin nt.$$

Hint: First suppose that r is real. Set $z = re^{it}$. Show that the two series are the real and imaginary parts of $(1+z)/(1-z)$. Finally make analytic continuation of the results.

(b) Let a be positive, $x \in [-a, a]$, while w is complex, $w \notin [-a, a]$. Let $r = r(w)$, $|r| < 1$ be a root of the quadratic $r^2 - (2w/a)r + 1 = 0$. Show that (with an appropriate definition of the square root)

$$\frac{1}{w-x} = \frac{1}{\sqrt{w^2-a^2}} \cdot \left(1 + 2\sum_{n=1}^{\infty} r^n T_n\Big(\frac{x}{a}\Big)\right), \quad (w \notin [-a,\ a],\ x \in [-a, a]).$$

(c) Find the expansion of $1/(1+x^2)$ for $x \in [-1.5, 1.5]$ into the polynomials $T_n(x/1.5)$. Explain the order of magnitude of the error and the main features of the error curve in Figure 3.2.5.

Hint: Set $w = i$, and take the imaginary part. Note that r becomes imaginary.

4. (a) Find the Laurent expansions for

$$f(z) = 1/(z-1) + 1/(z-2).$$

(b) How do you use the Cauchy–FFT method for finding Laurent expansions? Test your ideas on the function in the previous subproblem (and on a few other functions). There may be some pitfalls with the interpretation of the output from the FFT program, related to so-called **aliasing**; see Sec. 4.6.4 and Strang [44].

(c) As in Sec. 3.2.1, suppose that $F^{(p)}$ is of bounded variation in $[-\pi, \pi]$ and denote the Fourier coefficients of $F^{(p)}$ by $c_n^{(p)}$. Derive the following generalization of (3.2.8):

$$c_n = \frac{(-1)^{n-1}}{2\pi} \sum_{j=0}^{p-1} \frac{F^{(j)}(\pi) - F^{(j)}(-\pi)}{(in)^{j+1}} + \frac{c_n^{(p)}}{(in)^p},$$

and show that if we add the condition that $F \in C^j[-\infty, \infty]$, $j < p$, then the asymptotic results given in (and after) (3.2.8) hold.

(d) Let $z = \frac{1}{2}(w + w^{-1})$. Show that $|z-1| + |z+1| = |w| + |w|^{-1}$.

Hint: Use the parallelogram law, $|p-q|^2 + |p+q|^2 = 2(|p|^2 + |q|^2)$.

5. (a) The expansion of $\operatorname{arcsinh} t$ into powers of t, truncated after t^7, is obtained from Problem 1.6 (b). Using economization of a power series construct from this a polynomial approximation of the form $c_1 t + c_3 t^3$ in the interval $-\frac{1}{2} \le t \le \frac{1}{2}$. Give bounds for the truncation error for the original truncated expansion and for the economized expansion.

(b) The graph of $T_{12}(x)$ for $x \in [-1, 1]$ is shown in Figure 3.2.1. Draw the graph of $T_{12}(x)$ for (say) $x \in [-1.1, 1.1]$.

6. Show the following generalization of Theorem 3.2.4. Assume that $|f(z)| \leq M$ for $z \in \mathcal{E}_R$. Let $|\zeta| \in \mathcal{E}_\rho$, $1 < \rho < r \leq R - \epsilon$. Then the Chebyshev expansion of $f(\zeta)$ satisfies the inequality

$$\left| f(\zeta) - \sum_{j=0}^{n-1} c_j T_j(\zeta) \right| \leq \frac{2M(\rho/R)^n}{1 - \rho/R}.$$

Hint: Set $\omega = \zeta + \sqrt{\zeta^2 - 1}$, and show that $|T_j(\zeta)| = |\frac{1}{2}(\omega^j + \omega^{-j})| \leq \rho^j$.

7. Compute a few terms of the expansions into powers of ϵ or k of each of the roots of the following equations, so that the error is $O(\epsilon^2)$ or $O(k^{-2})$ (ϵ is small and positive; k is large and positive). Note that some terms may have fractional or negative exponents. Also try to fit an expansion of the wrong form in some of these examples, and see what happens.

(a) $(1+\epsilon)z^2 - \epsilon = 0$; (b) $\epsilon z^3 - z^2 + 1 = 0$; (c) $\epsilon z^3 - z + 1 = 0$;

(d) $z^4 - (k^2+1)z^2 - k^2 = 0$, $(k^2 \gg 1)$.

8. Modify Clenshaw's algorithm to a formula for the derivative of an orthogonal expansion.

9. (a) Let α_j, $j = 1 : n$ be the zeros of the Chebyshev polynomial $T_n(x)$, $n \geq 1$. (There are, of course, simple trigonometric expressions for them.) Apply Clenshaw's algorithm to compute $\sum_{m=0}^{n-1} T_m(\alpha_1)T_m(x)$, for $x = \alpha_j$, $j = 1 : n$. It turns out that the results are remarkably simple. (An explanation to this will be found in Sec. 4.5.

(b) Show that $S = \sum_{k=0}^{n-1} c_k \phi_k$ can be computed by a forward version of Clenshaw's algorithm that reads

$$\begin{aligned}
&y_{-2} = 0; \quad y_{-1} = 0;\\
&\textbf{for } k = 0 : n-1,\\
&\quad y_k = (-y_{k-2} + \alpha_k y_{k-1} + c_k)/\gamma_{k+1};\\
&\textbf{end}\\
&S = c_n\phi_n + \gamma_n y_{n-1}\phi_{n-1} - y_{n-2}\phi_n.
\end{aligned}$$

Add this version as an option to your program, and study Numerical Recipes [36, Sec. 5.4], from which this formula is quoted (with adaptation to our notation etc.). Make some test example of your own choice.

10. The solution of the boundary value problem

$$(1+\epsilon)y'' - \epsilon y = 0, \quad y(0) = 0, \quad y(1) = 1,$$

has an expansion of the form $y(t;\epsilon) = y_0(t) + y_1(t)\epsilon + y_2(t)\epsilon^2 + \ldots$.

(a) By coefficient matching, set up differential equations and boundary conditions for y_0, y_1, y_2, and solve them. You naturally use the boundary conditions of the original problem for y_0. Make sure you use the right boundary conditions for y_1, y_2.

(b) Set $R(t) = y_0(t) + \epsilon y_1(t) - y(t;\epsilon)$. Show that $R(t)$ satisfies the (modified) differential equation

$$(1+\epsilon)R'' - \epsilon R = \epsilon^2(7t - t^3)/6, \quad R(0) = 0, \quad R(1) = 0.$$

11. (a) Apply Kummer's first identity (3.2.39) to the error function $\operatorname{erf}(x)$, to show that

$$\operatorname{erf}(x) = \frac{2x}{\sqrt{\pi}} e^{-x^2} M\Big(1, \frac{3}{2}, x^2\Big) = \frac{2x}{\sqrt{\pi}} e^{-x^2}\Big(1 + \frac{2x^2}{3} + \frac{(2x^2)^2}{3\cdot 5} + \frac{(2x^2)^3}{3\cdot 5\cdot 7} + \dots\Big).$$

Why is this series well conditioned? (Note that it is a bell sum; compare Figure 3.2.7.) Investigate the largest term, rounding errors, truncation errors and termination criterion etc. in the same way as in (a).

(b) $\operatorname{erfc}(x)$ has a semi-convergent expansion for $x \gg 1$ that begins

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\, dt = \frac{e^{-x^2}}{x\sqrt{\pi}}\Big(1 - \frac{1}{2x^2} + \frac{3}{4x^4} - \frac{15}{8x^6} + \dots\Big).$$

Give an explicit expression for the coefficients, and show that the series diverges for every x. Where is the smallest term? Estimate its size.

Hint: Set $t^2 = x^2 + u$, and proceed analogously to Example 3.2.8. See Problem 3.1.7 (c), $\alpha = \frac{1}{2}$, about the remainder term. Alternatively, apply repeated integration by parts. It may be easier to find the remainder in this way.

12. Other notations for series, with application to Bessel functions.

(a) Set

$$f(x) = \sum_{n=0}^\infty \frac{a_n x^n}{n!}; \qquad g(x) = \sum_{n=0}^\infty \frac{b_n x^n}{n!}; \qquad h(x) = \sum_{n=0}^\infty \frac{c_n x^n}{n!};$$

$$\phi(w) = \sum_{n=0}^\infty \frac{\alpha_n w^n}{n!n!}; \qquad \psi(w) = \sum_{n=0}^\infty \frac{\beta_n w^n}{n!n!}; \qquad \chi(w) = \sum_{n=0}^\infty \frac{\gamma_n w^n}{n!n!}.$$

Let $h(x) = f(x)\cdot g(x)$; $\chi(w) = \phi(w)\cdot\psi(w)$. Show that

$$c_n = \sum_{j=0}^n \binom{n}{j} a_j b_{n-j}; \qquad \gamma_n = \sum_{j=0}^n \binom{n}{j}^2 \alpha_j \beta_{n-j}.$$

Derive analogous formulas for series of the form $\sum_{n=0}^\infty a_n w^n/(2n)!$ etc.. Suggest how to divide two power series in these notations.

(b) Let $a_j = (-1)^j a'_j$; $g(x) = e^x$. Show that

$$c_n = \sum_{j=0}^n \binom{n}{j} (-1)^j a'_j.$$

Comment: By (3.2.1), this can can also be written $c_n = (-1)^n \Delta^n a_0$. This proves the mathematical equivalence of the preconditioners (3.1.55) and (3.1.59)

if $P(x) = e^x$.

(c) Set, according to Example 3.2.8 and (a) (of this problem), $w = -x^2/4$,

$$J_0(x) = \sum_{n=0}^{\infty} \frac{(-1)^n w^n}{n!n!}; \quad I_0(x) = \sum_{n=0}^{\infty} \frac{w^n}{n!n!}; \quad IJ(x) \equiv I_0(x)J_0(x) = \sum_{n=0}^{\infty} \frac{\gamma_n w^n}{n!n!}.$$

Show that

$$\gamma_n = \sum_{j=0}^{n} (-1)^j \binom{n}{j}\binom{n}{n-j} = \begin{cases} (-1)^m \binom{2m}{m}, & \text{if } n = 2m; \\ 0, & \text{if } n = 2m+1. \end{cases}$$

Hint: The first expression for γ_n follows from (a). It can be interpreted as the coefficient of t^n in the product $(1-t)^n(1+t)^n$. The second expression for γ_n is the same coefficient in $(1-t^2)^n$.

(d) The second expression for γ_n in (c) is used in Example 3.2.8.[29] Reconstruct and extend the results of that example. Design a termination criterion. Where is the largest modulus of a term of the preconditioned series, and how large is it approximately? Make a crude guess in advance of the rounding error in the preconditioned series.

*(e) Show that the power series of $J_0(x)$ can be written in the form

$$\sum_{n=0}^{\infty} a_n \frac{(-x^2)^n}{(2n)!},$$

where a_n is positive and decreases slowly and smoothly.

Hint: Compute a_{n+1}/a_n.

Try preconditioning with $P(x) = \cosh x$. At the time of writing the authors do not know whether this is useful without multiple precision or not.

*(f) It is known; see Lebedev [30, (9.13.11)], that

$$J_0(x) = e^{-ix} M\left(\tfrac{1}{2}, 1; 2ix\right),$$

where $M(a, b, c)$ is Kummer's confluent hypergeometric function, this time with an imaginary argument. Show that Kummer's first identity is unfortunately of no use here for preconditioning the power series.

Comment: Most of the formulas and procedures in this problem can be generalized to the series for the Bessel functions of the first kind of general integer order, $(z/2)^{-n} J_n(x)$. These belong to the most studied functions of Applied Mathematics, and there exist more efficient methods for computing them; see, e.g., [36, Chapter 6]. This problem shows, however, that *preconditioning can work well* for a non-trivial power series, and it is worth to be tried, e.g., for other power series that may occur in connection with new applications.

[29] It is much better conditioned than the first expression. This may be one reason why multiple precision is not needed here.

13. (a) Derive the expansion of Example 3.2.5 by repeated integration by parts.
(b) Derive the Maclaurin expansion with the remainder according to (3.1.5) by the application of repeated integration by parts to the equation

$$f(z) - f(0) = z \int_0^1 f'(zt)\, d(t-1).$$

3.3 Difference Operators and Operator Expansions

3.3.1 Properties of Difference Operators

Difference operators are handy tools for the derivation, analysis, and practical application of numerical methods for many problems for interpolation, differentiation, and quadrature of a function in terms of its values at equidistant arguments. The simplest notations for difference operators and applications to derivatives, were mentioned in Sec. 1.2.3.

Let y denote a sequence $\{y_n\}$. Then we define the **shift operator** E (or translation operator) and the **forward difference operator** Δ by the relations

$$Ey = \{y_{n+1}\}, \qquad \Delta y = \{y_{n+1} - y_n\},$$

(see Sec. 1.2). E and Δ are thus operators which map one sequence to another sequence. Note, however, that if y_n is defined for $a \le n \le b$ only, then Ey_b is not defined, and the sequence Ey has fewer elements than the sequence y. (It is therefore sometimes easier to extend the sequences to infinite sequences, e.g., by adding zeros in both directions outside the original range of definition.)

These operators are **linear**, i.e. if α, β are real or complex constants and if y, z are two sequences, then $E(\alpha y + \beta z) = \alpha Ey + \beta Ez$, and similarly for Δ.

Powers of E and Δ are defined recursively, i.e.

$$E^k y = E(E^{k-1}y), \qquad \Delta^k y = \Delta(\Delta^{k-1}y).$$

By induction, the first relation yields $E^k y = \{y_{n+k}\}$. We extend the validity of this relation to $k = 0$ by setting $E^0 y = y$ and to negative values of k. $\Delta^k y$ is called the kth difference of the sequence y. We make the convention that $\Delta^0 = 1$. There will be little use of Δ^k for negative values of k in this book, although Δ^{-1} can be interpreted as a summation operator.

Note that $\Delta y = Ey - y$, and $Ey = y + \Delta y$ for any sequence y. It is therefore convenient to express these as equations between operators:

$$\Delta = E - 1, \qquad E = 1 + \Delta.$$

The identity operator is in this context traditionally denoted by 1. It can be shown that all formulas derived from the axioms of commutative algebra can be used for these operators, for example, the binomial theorem for positive integral k.

$$\Delta^k = (E-1)^k = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} E^j, \qquad E^k = (1+\Delta)^k = \sum_{j=0}^{k} \binom{k}{j} \Delta^j, \quad (3.3.1)$$

(3.3.2)

$$(\Delta^k y)_n = \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} y_{n+j}, \qquad y_{n+k} = (E^k y)_n = \sum_{j=0}^{k} \binom{k}{j} (\Delta^j y)_n. \quad (3.3.3)$$

We abbreviate the notation further and write, for example, $Ey_n = y_{n+1}$ instead of $(Ey)_n = y_{n+1}$, and $\Delta^k y_n$ instead of $(\Delta^k y)_n$. However, it is important to remember that Δ *operates on sequences* and not on elements of sequences. Thus, strictly speaking, this abbreviation is incorrect, though convenient. The formula for E^k will, in next subsection, be extended to an infinite series for non-integral values of k, but that is beyond the scope of algebra.

A **difference scheme** consists of a sequence and its difference sequences, arranged in the following way:

$$\begin{array}{lllll}
y_0 & & & & \\
 & \Delta y_0 & & & \\
y_1 & & \Delta^2 y_0 & & \\
 & \Delta y_1 & & \Delta^3 y_0 & \\
y_2 & & \Delta^2 y_1 & & \Delta^4 y_0 \\
 & \Delta y_2 & & \Delta^3 y_1 & \\
y_3 & & \Delta^2 y_2 & & \\
 & \Delta y_3 & & & \\
y_4 & & & &
\end{array}$$

A difference scheme is best computed by successive subtractions; the formulas in (3.3.1) are used mostly in theoretical contexts.

In many applications the quantities y_n are computed in increasing order $n = 0, 1, 2, \ldots$, and it is natural that a difference scheme is constructed by means of the quantities previously computed. One therefore introduces the **backward difference operator** $\nabla y_n = y_n - y_{n-1} = (1 - E^{-1})y_n$. For this operator we have

$$\nabla^k = (1 - E^{-1})^k, \qquad E^{-k} = (1 - \nabla)^k. \quad (3.3.4)$$

Note the **reciprocity** in the relations between ∇ and E^{-1}.

Any linear combination of the elements y_n, $y_{n-1}, \ldots$ y_{n-k} can also be expressed as a linear combination of y_n, $\nabla y_n, \ldots, \nabla^k y_n$, and vice versa [30] . For example, $y_n + y_{n-1} + y_{n-2} = 3y_n - 3\nabla y_n + \nabla^2 y_n$, because
$1 + E^{-1} + E^{-2} = 1 + (1 - \nabla) + (1 - \nabla)^2 = 3 - 3\nabla + \nabla^2$. By the reciprocity, we also obtain $y_n + \nabla y_n + \nabla^2 y_n = 3y_n - 3y_{n-1} + y_{n-2}$.

[30] An analogous statement holds for the elements $y_n, y_{n+1}, \ldots, y_{n+k}$ and forward differences.

In this notation the difference scheme reads

$$
\begin{array}{lllll}
y_0 & & & & \\
 & \nabla y_1 & & & \\
y_1 & & \nabla^2 y_2 & & \\
 & \nabla y_2 & & \nabla^3 y_3 & \\
y_2 & & \nabla^2 y_3 & & \nabla^4 y_4 \\
 & \nabla y_3 & & \nabla^3 y_4 & \\
y_3 & & \nabla^2 y_4 & & \\
 & \nabla y_4 & & & \\
y_4 & & & &
\end{array}
$$

In the backward difference scheme the subscripts are constant along diagonals directed upwards (backwards) to the right, while, in the forward difference scheme, subscripts are constant along diagonals directed downwards (forwards). Note, e.g., that $\nabla^k y_n = \Delta^k y_{n-k}$. In a computer, a backward difference scheme is preferably stored as a lower triangular matrix.

Example 3.3.1.
Part of the difference scheme for the sequence $y = \{\ldots, 0, 0, 0, 1, 0, 0, 0, \ldots\}$ is given below.

$$
\begin{array}{ccccccccc}
 & 0 & & 0 & & 1 & & -7 & \\
0 & & 0 & & 1 & & -6 & & 28 \\
 & 0 & & 1 & & -5 & & 21 & \\
0 & & 1 & & -4 & & 15 & & -56 \\
 & 1 & & -3 & & 10 & & -35 & \\
1 & & -2 & & 6 & & -20 & & 70 \\
 & -1 & & 3 & & -10 & & 35 & \\
0 & & 1 & & -4 & & 15 & & -56 \\
 & 0 & & -1 & & 5 & & -21 & \\
0 & & 0 & & 1 & & -6 & & 28 \\
 & 0 & & 0 & & -1 & & 7 &
\end{array}
$$

This example shows the *effect of a disturbance in one element* on the sequence of the higher differences. Because the effect broadens out and grows quickly, difference schemes are useful in the investigation and correction of computational and other errors, so-called **difference checks**. Notice that, since the differences are *linear* functions of the sequence, a **superposition principle** holds. The effect of errors can thus be estimated by studying simple sequences such as the one above.

Example 3.3.2.
The following is a difference scheme for a 5 decimal table of the function $f(x) = \tan x$, $x \in [1.30, 1.36]$, with step $h = 0.01$. The differences are given with

10^{-5} as unit.

x	y	∇y	$\nabla^2 y$	$\nabla^3 y$	$\nabla^4 y$	$\nabla^5 y$	$\nabla^6 y$
1.30	3.60210						
		14498					
1.31	3.74708		1129				
		15627		140			
1.32	3.90335		1269		26		
		16896		166		2	
1.33	4.07231		1435		28		9
		18331		194		11	
1.34	4.25562		1629		39		
		19960		233			
1.35	4.45522		1862				
		21822					
1.36	4.67344						

We see that the differences decrease roughly by a factor of 0.1—that indicates that the step size has been chosen suitably for the purpose of interpolation, numerical quadrature etc.—until the last two columns, where the rounding errors of the function values have a visible effect.

Example 3.3.3.

For the sequence $y_n = (-1)^n$ one finds easily that

$$\nabla y_n = 2y_n, \nabla^2 y_n = 4y_n, \ldots, \nabla^k y_n = 2^k y_n.$$

If the error in the elements of the sequence are bounded by ϵ, it follows that the errors of the kth differences are bounded by $2^k\epsilon$. A rather small reduction of this bound is obtained if the errors are assumed to be independent random variables (Problem 3.3.26).

It is natural also to consider *difference operations on functions* not just on sequences. E and Δ map the function f onto functions whose values at the point x are

$$E\,f(x) = f(x+h), \qquad \Delta f(x) = f(x+h) - f(x), \tag{3.3.5}$$

where h is the *step size*. Of course, Δf depends on h; in some cases this should be indicated in the notation. One can, for example, write $\Delta_h f(x)$, or $\Delta f(x;h)$. If we set $y_n = f(x_0 + nh)$, the difference scheme of the function with step size h is the same as for the sequence $\{y_n\}$. Again it is important to realize that, in this case, the operators act on *functions*, not on the values of functions. It would be more correct to write $f(x_0 + h) = (Ef)(x_0)$. Actually, the notation $(x_0)Ef$ would be even more logical, since the insertion of the value of the argument x_0 is the last operation to be done, and the convention for the order of execution of operators proceeds from right to left, but this notation would be too revolutionary.[31]

[31]The notation $[x_0]f$ occurs, however, naturally in connection with divided differences, Sec. 4.2.

Note that *no new errors are introduced during the computation of the differences, but the effects of the original irregular errors of y grow exponentially.* We emphasize the word **irregular errors**, e.g., rounding errors in y, since systematic errors, e.g., the truncation errors in the numerical solution of a differential equation, often have a smooth difference scheme. For example, if the values of y have been produced by the iterative solution of an equation, where x is a parameter, with the same number of iterations for every x and y and the same algorithm for the first approximation, then the truncation error of y is likely to be a smooth function of x.

Difference operators are in many respects similar to differentiation operators. Let f be a polynomial. By Taylor's formula,

$$\Delta f(x) = f(x+h) - f(x) = hf'(x) + \frac{1}{2}h^2 f''(x) + \dots.$$

We see from this that $deg\,\Delta f = deg\, f - 1$. Similarly for differences of higher order; *if f is a polynomial of degree less than k, then*

$$\Delta^{k-1} f(x) = constant, \quad \Delta^p f(x) = 0, \ \forall p \geq k.$$

The same holds for backward differences.

The following important result can be derived directly from Taylor's theorem with the integral form of the remainder. Assume that all derivatives of f up to kth order are continuous. If $f \in C^k$,

$$\Delta^k f(x) = h^k f^{(k)}(\zeta), \quad \zeta \in [x, x+kh]. \tag{3.3.6}$$

Hence $h^{-k}\Delta^k f(x)$ is an approximation to $f^{(k)}(x)$; the error of this approximation approaches zero as $h \to 0$ (i.e. as $\zeta \to x$). As a rule, the error is approximately proportional to h. We postpone the proof to Chapter 4, where it appears as a particular case of a theorem concerning divided differences.

Even though difference schemes do not have the same importance today that they had in the days of hand calculations or calculation with desk calculators, they are still important conceptually, and we shall also see how they are still useful also in practical computing. In a computer it is more natural to store a difference scheme as an array, e.g. with y_n, ∇y_n, $\nabla^2 y_n, \dots$, $\nabla^k y_n$ in a row (instead of along a diagonal).

Many formulas for differences are analogous to formulas for derivatives, though usually more complicated. The following results are among the most important.

Lemma 3.3.1.

It holds that

$$\Delta^k(a^x) = (a^h - 1)^k a^x, \qquad \nabla^k(a^x) = (1 - a^{-h})^k a^x. \tag{3.3.7}$$

For sequences, i.e. if h=1,

$$\Delta^k\{a^n\} = (a-1)^k\{a^n\}, \qquad \Delta^k\{2^n\} = \{2^n\}. \tag{3.3.8}$$

Proof. Let c be a given constant. For $k = 1$ we have

$$\Delta(ca^x) = ca^{x+h} - ca^x = ca^x a^h - ca^x = c(a^h - 1)a^x$$

The general result follows easily by induction. The backward difference formula is derived in the same way. □

Lemma 3.3.2. *Summation by Parts*

$$\sum_{n=0}^{N-1} u_n \Delta v_n = u_N v_N - u_0 v_0 - \sum_{n=0}^{N-1} \Delta u_n \, v_{n+1}. \tag{3.3.9}$$

Proof. (Compare the rule for integration by parts and its proof!) Notice that

$$\sum_{n=0}^{N-1} \Delta w_n = (w_1 - w_0) + (w_2 - w_1) + \ldots + (w_N - w_{N-1}) = w_N - w_0.$$

Use this on $w_n = u_n v_n$. From the result in Lemma 4.5.2 one gets after summation,

$$u_N v_N - u_0 v_0 = \sum_{n=0}^{N-1} u_n \Delta v_n + \sum_{n=0}^{N-1} \Delta u_n v_{n+1},$$

and the result follows. (For an extension; see Problem 1d.) □

Lemma 3.3.3. *The Difference of a Product*

$$\Delta(u_n v_n) = u_n \Delta v_n + \Delta u_n \, v_{n+1}. \tag{3.3.10}$$

Proof. We have

$$\Delta(u_n v_n) = u_{n+1} v_{n+1} - u_n v_n = u_n(v_{n+1} - v_n) + (u_{n+1} - u_n) v_{n+1}.$$

□

Compare the above result with the formula for differentials, $d(uv) = udv + vdu$. Note that we have v_{n+1} (not v_n) on the right-hand side.

3.3.2 The Calculus of Operators

Formal calculations with operators, using the rules of algebra and analysis, are often an elegant means of assistance in *finding approximation formulas that are exact for all polynomials of degree less than (say) k*, and they should therefore be

useful for functions that can be accurately approximated by such a polynomial. Our calculations often lead to divergent (or semi-convergent) series, but the way we handle them can usually be justified by means of the theory of formal power series, of which a brief introduction was given at the end of Sec. 3.1.5. The operator calculations also provide error estimates, asymptotically valid as the step size $h \to 0$. Strict error bounds can be derived by means of Peano's remainder theorem, Sec. 3.3.3.

Operator techniques are sometimes successfully used (see, e.g., Sec. 3.3.4) in a way that it is hard, or even impossible, to justify by means of formal power series. It is then not trivial to formulate appropriate conditions for the success and to derive satisfactory error bounds and error estimates, but it can sometimes be done.

We make a digression about terminology. More generally, *the word* **operator** *is in this book used for a function that maps a linear space $\mathcal{S}$ into another linear space $\mathcal{S}'$.* $\mathcal{S}$ can, for example, be a space of functions, a coordinate space, or a space of sequences. The dimension of these spaces can be finite or infinite. For example, the differential operator D maps the infinite-dimensional space $C^1[a,b]$ of functions with a continuous derivative, defined on the interval $[a,b]$, into the space $C[a,b]$ of continuous functions on the same interval.

In the following we denote by $\mathcal{P}_k$ the set of polynomials of degree *less than* k. [32] Note that $\mathcal{P}_k$ is a k-dimensional linear space, for which $\{1,\, x,\, x^2, \ldots,\, x^{k-1}\}$ is a basis called the *power basis*; the coefficients $(c_1,\, c_2, \ldots,\, c_k)$ are then the *coordinates* of the polynomial p defined by $p(x) = \sum_{i=1}^{k} c_i x^{i-1}$.

For simplicity, we shall assume that the space of functions on which the operators are defined is $C^\infty(-\infty,\infty)$, i.e. the functions are infinitely differentiable on $(-\infty,\infty)$. This sometimes requires (theoretically) a modification of a function outside the bounded interval where it is interesting. There are techniques for achieving this, but they are beyond the scope of this book. Just imagine that they have been applied.

We define the following operators:

$Ef(x) = f(x+h)$	Shift (or translation) operator
$\Delta f(x) = f(x+h) - f(x)$	Forward difference operator
$\nabla f(x) = f(x) - f(x-h)$	Backward difference operator
$Df(x) = f'(x)$	Differentiation operator
$\delta f(x) = f(x+\frac{1}{2}h) - f(x-\frac{1}{2}h)$	Central difference operator
$\mu f(x) = \frac{1}{2}\big(f(x+\frac{1}{2}h) + f(x-\frac{1}{2}h)\big)$	Averaging operator

Suppose that the values of f are given on an equidistant grid only, e.g., $x_j = x_0 + jh$, $j = -M : N$, (j is integer). Set $f_j = f(x_j)$. Note that δf_j, $\delta^3 f_j \ldots$, (odd powers) and μf_j *cannot* be exactly computed; they are available halfway between the grid points. (A way to get around this is given later; see (3.3.47)) The even powers $\delta^2 f_j$, $\delta^4 f_j \ldots$, and $\mu\delta f_j$, $\mu\delta^3 f_j \ldots$, *can* be exactly computed. This follows from the

[32] Some authors use similar notations to denote the set of polynomials of degree less than or equal to k. We regret that.

formulas

$$\mu\delta f(x) = \frac{1}{2}\big(f(x+h) - f(x-h)\big), \quad \mu\delta = \tfrac{1}{2}(\Delta + \nabla), \quad \delta^2 = \Delta - \nabla. \tag{3.3.11}$$

Several other notations are in use, e.g., at the study of difference methods for partial differential equations D_{+h}, D_{0h}, D_{-h} are used instead of $\Delta, \mu\delta, \nabla$, respectively.

An operator P is said to be a **linear operator** if

$$P(\alpha f + \beta g) = \alpha P f + \beta P g$$

holds for arbitrary complex constants α, β and arbitrary functions f, g. The above six operators are all linear. The operation of multiplying by a constant α, is also a linear operator.

If P and Q are two operators, then their sum, product, etc., can be defined in the following way:

$$\begin{aligned}(P+Q)f &= Pf + Qf,\\ (P-Q)f &= Pf - Qf,\\ (PQ)f &= P(Qf),\\ (\alpha P)f &= \alpha(Pf),\\ P^n f &= P\cdot P\cdots Pf, \quad n \text{ factors.}\end{aligned}$$

Two operators are equal, $P = Q$ if $Pf = Qf$, for all f in the space of functions considered. Notice that $\Delta = E - 1$. One can show that the following rules hold for all linear operators:

$$\begin{aligned} P + Q = Q + P, \qquad & P + (Q + R) = (P + Q) + R,\\ P(Q + R) = PQ + PR, \qquad & P(QR) = (PQ)R.\end{aligned}$$

The above six operators, E, Δ, ∇, hD, δ, and μ, and the combinations of them by these algebraic operations make a *commutative ring*, so $PQ = QP$ holds for these operators, and any algebraic identity that is generally valid in such rings can be used.

If $\mathcal{S} = \mathbf{R^n}$, $\mathcal{S}' = \mathbf{R^m}$, and the elements are *column* vectors, then the linear operators are matrices of size $[m, n]$. They do generally not commute.

If $\mathcal{S}' = \mathbf{R}$ or $\mathbf{C}$, the operator is called a **functional**. Examples of functionals are, if x_0 denotes a fixed (though arbitrary) point,

$$Lf = f(x_0), \quad Lf = f'(x_0), \quad Lf = \int_0^1 e^{-x} f(x)dx, \quad \int_0^1 |f(x)|^2 dx;$$

all except the last one are **linear functionals**.

There is a subtle distinction here. For example, E is a linear operator that maps a function to a function. Ef is the function whose value at the point x is $f(x + h)$. If we consider a fixed point, e.g. x_0, then $(Ef)(x_0)$ is a scalar. This is therefore a *linear functional*. We shall allow ourselves to simplify the notation and

to write $Ef(x_0)$, but it must be understood that E operates on the function f, not on the function value $f(x_0)$. This was just one example; simplifications like this will be made with other operators than E, and similar simplifications in notation were suggested earlier in this chapter. There are, however, situations, where it is, for the sake of clarity, advisable to return to the more specific notation with a larger number of parentheses.

If we represent the vectors in $\mathbf{R}^n$ by *columns* y, the linear functionals in $\mathbf{R}^n$ are the scalar products $a^T x = \sum i = 1^n a_i y_i$; every *row* a^T thus defines a linear functional.

Examples of linear functionals in $\mathcal{P}_k$ are linear combinations of a finite number of function values, $Lf = \sum a_j f(x_j)$. If $x_j = x_0 + jh$ the same functional can be expressed in terms of differences, e.g., $\sum a'_j \Delta^j f(x_0)$; see Problem 3. The main topic of this section is to show how operator methods can be used for finding approximations of this form to linear functionals in more general function spaces. First, we need a general theorem.

Theorem 3.3.4.

Let x_1, $x_2, \ldots$, x_k be k distinct real (or complex) numbers. Then no non-trivial relation of the form

$$\sum_{j=1}^{k} a_j f(x_j) = 0 \tag{3.3.12}$$

can hold for all $f \in \mathcal{P}_k$. If we add one more point (x_0), there exists only one non-trivial relation of the form $\sum_{j=0}^{k} a'_j f(x_j) = 0$, (except that it can be multiplied by an arbitrary constant). In the equidistant case, i.e. if $x_j = x_0 + jh$, then

$$\sum_{j=0}^{k} a'_j f(x_j) \equiv c\Delta^k f(x_0), \quad c \neq 0.$$

Proof. If (3.3.12) were valid for all $f \in \mathcal{P}_k$, then the linear system

$$\sum_{j=1}^{k} x_j^{i-1} a_j = 0, \quad i = 1, \ldots, k,$$

would have a non-trivial solution (a_1, $a_2, \ldots$, a_k). The matrix of the system, however, is a Vandermonde matrix; its determinant is thus equal to the product of all differences $(x_i - x_j)$, $i > j$, $1 < i \leq k$, which is nonzero.

Now we add the point x_0. Suppose that there exist two relations,

$$\sum_{j=0}^{k} b_j f(x_j) = 0, \qquad \sum_{j=0}^{k} c_j f(x_j) = 0.$$

with linearly independent coefficient vectors. Then we can find a (non-trivial) linear combination, where x_0 has been eliminated, but this contradicts the result that we

have just proved. Hence the hypothesis is wrong; the two coefficient vectors must be proportional. We have seen above that, in the equidistant case, $\Delta^k f(x_0) = 0$ is such a relation. More generally, we shall see in Chapter 4 that, for $k+1$ arbitrary distinct points, the kth order *divided difference* is zero for all $f \in \mathcal{P}_k$. □

Corollary 3.3.5.

Suppose that a formula for interpolation, numerical differentiation or integration etc. has been derived, for example by an operator technique. If it is a linear combination of the values of $f(x)$ at k given *distinct points x_j, $j = 1:k$, and is exact for all $f \in \mathcal{P}_k$, this formula is unique. (If it is exact for all $f \in \mathcal{P}_m$, $m < k$, only, it is not unique.)*

In particular, for any $\{c_j\}_{j=1}^k$, a unique polynomial $P \in \mathcal{P}_k$ is determined by the interpolation conditions $P(x_j) = c_j$, $j = 1:k$.

Proof. The difference between two formulas that use the same function values would lead to a relation that is impossible, by the theorem. □

Now we shall go outside of polynomial algebra and consider also *infinite series of operators*. The Taylor series

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \dots$$

can be written symbolically as

$$Ef = \Big(1 + hD + \frac{(hD)^2}{2!} + \frac{(hD)^3}{3!} + \dots\Big) f.$$

We can here treat hD like an algebraic indeterminate, and consider the series inside the parenthesis (without the operand) as a *formal power series*[33]

For a formal power series the concepts of convergence and divergence do not exist. When the operator series acts on a function f, and is evaluated at a point c, we obtain an ordinary numerical series, related to the linear functional $Ef(c) = f(c+h)$. We know that this Taylor series may converge or diverge, depending on f, c, and h.

Roughly speaking, the last part of Sec. 3.1.5 tells that, with some care, "analytic functions" of one indeterminate can be handled with the same rules as analytic functions of one complex variable.

Theorem 3.3.6.

$$\begin{aligned} e^{hD} = E = 1 + \Delta, \qquad & e^{-hD} = E^{-1} = 1 - \nabla, \\ 2\sinh \tfrac{1}{2}hD = e^{hD/2} - e^{-hD/2} &= \delta, \\ (1+\Delta)^\theta = (e^{hD})^\theta = e^{\theta hD}, \quad & (\theta \in \mathbf{R}). \end{aligned}$$

[33] We now abandon the bold-type notation for indeterminates and formal power series used in Sec. 3.1.5 for the function e^{hD}, which is defined by this series. The reader is advised to take a look again at the last part of Sec. 3.1.5.

Proof. The first formula follows from the previous discussion. The second and the third formulas are obtained in a similar way. (Recall the definition of δ.) The last formula follows from the first formula together with Lemma 3.1.9 (in Sec. 3.1.3). □ It follows from the power series expansion that

$$(e^{hD})^\theta f(x) = e^{\theta hD} f(x) = f(x + \theta h),$$

when it converges. Since $E = e^{hD}$ it is natural to *define*

$$E^\theta f(x) = f(x + \theta h),$$

and we extend this definition also to such values of θ that the power series for $e^{\theta hD} f(x)$ is divergent. Note that, e.g., the formula $E^{\theta_2} E^{\theta_1} f(x) = E^{\theta_2 + \theta_1} f(x)$, follows from this definition.

When one works with operators or functionals it is advisable to avoid notations like Δx^n, $De^{\alpha x}$, where the variables appear in the operands. For two important functions we therefore set

$$F_\alpha : F_\alpha(x) = e^{\alpha x}; \qquad f_n : f_n(x) = x^n. \tag{3.3.13}$$

Let P be any of the operators mentioned above. When applied to F_α it acts like a scalar that we shall call **the scalar of the operator** [34] and denote it by $\mathrm{sc}(P)$,

$$PF_\alpha = \mathrm{sc}(P) F_\alpha.$$

We may also write $\mathrm{sc}(P; h\alpha)$ if it is desirable to emphasize its dependence on $h\alpha$. (We normalize the operators so that this is true, e.g., we work with hD instead of D.) Note that

$$\mathrm{sc}(\beta P + \gamma Q) = \beta \mathrm{sc}(P) + \gamma \mathrm{sc}(Q), \ (\beta, \gamma \in \mathbf{C}), \quad \mathrm{sc}(PQ) = \mathrm{sc}(P)\mathrm{sc}(Q),$$

For our most common operators we obtain

$$(E^\theta) = e^{\theta h\alpha}; \qquad \mathrm{sc}(\nabla) = \mathrm{sc}(1 - E^{-1}) = 1 - e^{-h\alpha}; \tag{3.3.14}$$

$$\mathrm{sc}(\Delta) = \mathrm{sc}(E - 1) = e^{h\alpha} - 1; \tag{3.3.15}$$

$$\mathrm{sc}(\delta) = \mathrm{sc}(E^{1/2} - E^{-1/2}) = e^{h\alpha/2} - e^{-h\alpha/2}.$$

Let Q_h be one of the operators hD, Δ, δ, ∇. It follows from the last formulas that

$$\mathrm{sc}(Q_h) \sim h\alpha, \ (h \to 0); \quad |\mathrm{sc}(Q_h)| \le |h\alpha| e^{|h\alpha|}$$

The main reason for grouping these operators together is that each of them has the important property (3.3.6), i.e. $Q_h^k f(c) = h^k f^{(k)}(\zeta)$, where ζ lies in the smallest interval that contains all the arguments used in the computation of $Q_h^k f(c)$. Hence,

$$f \in \mathcal{P}_k \quad \Rightarrow \quad Q_h^n f = 0, \quad \forall n \ge k. \tag{3.3.16}$$

[34] In applied Fourier analysis this scalar is, for $\alpha = i\omega$, often called the *symbol of the operator*.

This property [35] makes each of these four operators well suited to be the indeterminate in a formal power series that, hopefully, will be able to generate a sequence of approximations, L_1, L_2, $L_3 \ldots$, to a given linear operator L. L_n is the nth partial sum of a formal power series for L. Then

$$f \in \mathcal{P}_k \quad \Rightarrow \quad L_n f = L_k f, \quad \forall n \geq k. \tag{3.3.17}$$

We shall see in the next theorem that, for expansion into powers of Q_h,

$$\lim_{n\to\infty} L_n f(x) = Lf(x)$$

if f is a polynomial. This is not quite self-evident, because it is not true for all functions f, and we have seen in Sec. 3.1.5 Sec. 3.1.3 that it can happen that an expansion converges to a "wrong result". We shall see more examples of that later. Convergence does not necessarily imply validity.

Suppose that z is a complex variable, and that $\phi(z)$ is analytic at the origin, i.e. $\phi(z)$ is equal to its Maclaurin series, (say) $\phi(z) = a_0 + a_1 z + a_2 z^2 + \ldots$, if $|z| < \rho$ for some $\rho > 0$. For multivalued functions we always refer to the principal branch. The operator function $\phi(Q_h)$ is usually defined by the *formal* power series,

$$\phi(Q_h) = a_0 + a_1 Q_h + a_2 Q_h^2 + \ldots,$$

where Q_h is treated like an algebraic indeterminate.

The operators E, hD, Δ, δ, ∇ and μ are related to each others. See Table 3.3.2 that is adapted from an article by the eminent blind British mathematician W. G. Bickley (1948). Some of these formulas follow almost directly from the definitions, others are derived in this section, and the rest are left for Problem 5e. We find the value $\mathrm{sc}(\cdot)$ for each of these operators by *substituting α for D in the last column of the table*. (Why?)

Table 3.3.1. *Bickley's table of relations between difference operators*

	E	Δ	δ	∇	hD
E	E	$1+\Delta$	$1+\frac{1}{2}\delta^2+\delta\sqrt{1+\frac{1}{4}\delta^2}$	$\frac{1}{1-\nabla}$	e^{hD}
Δ	$E-1$	Δ	$\delta\sqrt{1+\frac{1}{4}\delta^2}+\frac{1}{2}\delta^2$	$\frac{\nabla}{1-\nabla}$	$e^{hD}-1$
δ	$E^{1/2}-E^{-1/2}$	$\Delta(1+\Delta)^{-1/2}$	δ	$\nabla(1-\nabla)^{-1/2}$	$2\sinh\frac{1}{2}hD$
∇	$1-E^{-1}$	$\frac{\Delta}{1+\Delta}$	$\delta\sqrt{1+\frac{1}{4}\delta^2}-\frac{1}{2}\delta^2$	∇	$1-e^{-hD}$
hD	$\ln E$	$\ln(1+\Delta)$	$2\sinh^{-1}\frac{1}{2}\delta$	$-\ln(1-\nabla)$	hD
μ	$\frac{1}{2}(E^{1/2}+E^{-1/2})$	$\frac{1+\frac{1}{2}\Delta}{(1+\Delta)^{1/2}}$	$\sqrt{1+\frac{1}{4}\delta^2}$	$\frac{1-\frac{1}{2}\nabla}{(1-\nabla)^{1/2}}$	$\cosh\frac{1}{2}hD$

[35] The operators E and μ do *not* possess this property.

Example 3.3.4. *Express E in terms of ∇.*

The definition of ∇ reads in operator form $E^{-1} = 1 - \nabla$. This can be looked upon as a formal power series (with only two non-vanishing terms) for the reciprocal of E with ∇ as the indeterminate. By the rules for formal power series mentioned in Sec. 3.1.5, we obtain *uniquely*

$$E = (E^{-1})^{-1} = (1 - \nabla)^{-1} = 1 + \nabla + \nabla^2 + \dots.$$

We find in the table an equivalent expression containing a fraction line. Suppose that we have proved the last column of the table. So, $\mathrm{sc}(\nabla) = 1 - e^{-h\alpha}$, hence

$$\mathrm{sc}((1 - \nabla)^{-1}) = (e^{-h\alpha})^{-1} = e^{h\alpha} = \mathrm{sc}(E).$$

Example 3.3.5.

Suppose that we have proved the first and the last columns of Bickley's table (except for the equation $hD = \ln E$). We shall prove one of the formulas in the second column, namely the equation $\delta = \Delta(1 + \Delta)^{-1/2}$. By the first column, the right hand side is equal to $(E - 1)E^{-1/2} = E^{1/2} - E^{-1/2} = \delta$, Q.E.D.

We shall also compute $\mathrm{sc}(\Delta(1 + \Delta)^{-1/2})$. Since $\mathrm{sc}(\Delta) = e^{h\alpha} - 1$ we obtain

$$\begin{aligned}\mathrm{sc}(\Delta(1 + \Delta)^{-1/2}) &= (e^{h\alpha} - 1)(e^{h\alpha})^{-1/2} = e^{h\alpha/2} - e^{-h\alpha/2} \\ &= 2\sinh \tfrac{1}{2}h\alpha = \mathrm{sc}(\delta).\end{aligned}$$

By the aid of Bickley's table, we are in a position to transform L into the form $\phi(Q_h)R_h$. (A sum of several such expressions with different indeterminates can also be treated.)

- Q_h is the one of the four operators, hD, Δ, δ, ∇, which we have chosen to be the "indeterminate".

  $$Lf \simeq \phi(Q_h)f = (a_0 + a_1Q_h + a_2Q_h^2 + \dots)f. \tag{3.3.18}$$

  The coefficients a_j are the same as the Maclaurin coefficients of $\phi(z)$, $z \in \mathbf{C}$ if $\phi(z)$ is analytic at the origin. They can be determined by the techniques described in Sec. 3.1.5 and Sec. 3.1.4. The meaning of the relation $\simeq$ will hopefully be clear from the following theorem.

- R_h is, e.g., $\mu\delta$ or E^k, k integer, or more generally any linear operator with the properties that $R_hF_\alpha = \mathrm{sc}(R_h)F_\alpha$, and that the values of $R_hf(x_n)$ on the grid $x_n = x_0 + nh$, n integer, are determined by the values of f on the same grid.

Theorem 3.3.7. *Recall the notation Q_h for either of the operators Δ, δ, ∇, hD, and the notations $F_\alpha(x) = e^{\alpha x}$, $f_n(x) = x^n$. Note that*

$$F_\alpha(x) = \sum_{n=0}^{\infty} \frac{\alpha^n}{n!} f_n(x). \tag{3.3.19}$$

Also recall the scalar of an operator and its properties, e.g.,

$$LF_\alpha = \mathrm{sc}(L)F_\alpha, \qquad Q_h^j F_\alpha = (\mathrm{sc}(Q_h))^j F_\alpha;$$

for the operators under consideration the scalar depends on $h\alpha$.
Assumptions:

(i) *A formal power series equation $L = \sum_{j=0}^{\infty} a_j Q_h^j$ has been derived.*[36] *Furthermore, $|\mathrm{sc}(Q_h)| < \rho$, where ρ is the convergence radius of the series $\sum a_j z^j$, $z \in \mathbf{C}$, and*

$$\mathrm{sc}(L) = \sum_{j=0}^{\infty} a_j (\mathrm{sc}(Q_h))^j. \tag{3.3.20}$$

(ii)

$$L\frac{\partial^n}{\partial \alpha^n} F_\alpha(x) = \frac{\partial^n}{\partial \alpha^n}(LF_\alpha)(x)$$

at $\alpha = 0$, or equivalently,

$$L\int_C \frac{F_\alpha(x)\,d\alpha}{\alpha^{n+1}} = \int_C \frac{(LF_\alpha)(x)\,d\alpha}{\alpha^{n+1}}. \tag{3.3.21}$$

where C is any circle with the origin as center.

(iii) *The domain of x is a bounded interval I_1 in $\mathbf{R}$.*

Then

$$LF_\alpha = \Big(\sum_{j=0}^{\infty} a_j Q_h^j\Big) F_\alpha, \quad \text{if } |\mathrm{sc}(Q_h)| < \rho, \tag{3.3.22}$$

$$Lf(x) = \sum_{j=0}^{k-1} a_j Q_h^j f(x), \quad \text{if } f \in \mathcal{P}_k, \tag{3.3.23}$$

for any positive integer k.

A **strict error bound** *for* (3.3.23)*, if $f \notin \mathcal{P}_k$, is obtained in Peano's Theorem* 3.3.8.

An **asymptotic error estimate** *(as $h \to 0$ for fixed k) is given by the first neglected non-vanishing term $a_r Q_h^r f(x) \sim a_r (hD)^r f(x)$, $r \geq k$, if $f \in C^r[I]$, where the interval I must contain all the points used in the evaluation of $Q_h^r f(x)$.*

Proof. By Assumption 1,

$$LF_\alpha = \mathrm{sc}(L)F_\alpha = \lim_{J\to\infty} \sum_{j=0}^{J-1} a_j \mathrm{sc}(Q_h^j) F_\alpha = \lim_{J\to\infty} \sum_{j=0}^{J-1} a_j Q_h^j F_\alpha = \lim_{J\to\infty} \Big(\sum_{j=0}^{J-1} a_j Q_h^j\Big) F_\alpha,$$

[36] To simplify the writing, the operator R_h is temporarily neglected. See one of the comments below.

hence $LF_\alpha = (\sum_{j=0}^{\infty} Q_h^j)F_\alpha$. This proves the first part of the theorem.
By (3.3.19), Cauchy's formula (3.2.9) and Assumption 2,

$$\frac{2\pi i}{n!}Lf_n(x) = L\int_C \frac{F_\alpha(x)\,d\alpha}{\alpha^{n+1}} = \int_C \frac{(LF_\alpha)(x)\,d\alpha}{\alpha^{n+1}}$$
$$= \int_C \sum_{j=0}^{J-1} \frac{a_j Q_h^j F_\alpha(x)\,d\alpha}{\alpha^{n+1}} + \int_C \sum_{j=J}^{\infty} \frac{a_j \mathrm{sc}(Q_h)^j F_\alpha(x)\,d\alpha}{\alpha^{n+1}}.$$

Let ϵ be any positive number. Choose J so that the modulus of the last term becomes $\epsilon\theta_n 2\pi/n!$, where $|\theta_n| < 1$. This is possible, since $|\mathrm{sc}(Q_h)| < \rho$; see Assumption 1. Hence, for every $x \in I_1$,

$$Lf_n(x) - \epsilon\theta_n = \frac{n!}{2\pi i}\sum_{j=0}^{J-1} a_j Q_h^j \int_C \frac{F_\alpha(x)\,d\alpha}{\alpha^{n+1}} = \sum_{j=0}^{J-1} a_j Q_h^j f_n(x) = \sum_{j=0}^{k-1} a_j Q_h^j f_n(x).$$

The last step holds if $J \geq k > n$, because, by (3.3.16), $Q_h^j f_n = 0$ for $j > n$. It follows that $|Lf_n(x) - \sum_{j=0}^{k-1} a_j Q_h^j f_n(x)| < \epsilon$ for every $\epsilon > 0$, hence $Lf_n = \sum_{j=0}^{k-1} a_j Q_h^j f_n$.

If $f \in \mathcal{P}_k$, f is a linear combination of f_n, $n = 0 : k-1$. Hence $Lf = \sum_{j=0}^{k-1} a_j Q_h^j f$ if $f \in \mathcal{P}_k$. This proves the second part of the theorem.

The error bound is derived in Sec. 3.3.1. Recall the important formula (3.3.6) that expresses the kth difference as the value of the k'th derivative in a point located in an interval that contains all the points used in the in the computation of the k'th difference. i.e. the ratio of the error estimate $a_r(hD)^r f(x)$ to the true truncation error tends to 1, as $h \to 0$. □

Remark 3.3.1. This theorem is concerned with series of powers of the four operators collectively denoted Q_h. One may try to use operator techniques also to *find* a formula involving, e.g., an infinite expansion into powers of the operator E. Then one should try afterwards to find sufficient conditions for the validity of the result. This procedure will be illustrated in connection with Euler–Maclaurin's formula in Sec. 3.4.4.

Sometimes, operator techniques which are not covered by this theorem can, after appropriate restrictions, be justified (or even replaced) by *transform methods*, e.g., z-transforms, Laplace or Fourier transforms.

The operator R_h that was introduced just before the theorem, was neglected in the proof, in order to simplify the writing. We now have to multiply the operands by R_h in the proof and in the results. This changes practically nothing for F_α, since $R_h F_\alpha = \mathrm{sc}(R_h)F_\alpha$. In (3.3.23) there is only a trivial change, because the polynomials f and $R_h f$ may not have the same degree. For example, if $R_h = \mu\delta$ and $f \in P_k$ then $R_h f \in P_{k-1}$. The verification of the assumptions typically offers no difficulties.

It follows from the linearity of (3.3.22) that *it is satisfied also if F_α is replaced by a linear combination of exponential functions F_α with different α*, provided that $|\mathrm{sc}(Q_h)| < \rho$ for all the occurring α. With some care, one can let the linear combination be an infinite series or an integral.

There are two things to note in connection with the asymptotic error estimates. First, the step size should be small enough; this means in practice that, in the beginning, the magnitude of the differences should decrease rapidly, as their order increases. When the order of the differences becomes large, it often happens that the moduli of the differences also become increasing. This can be due to two causes: semi-convergence (see the next comment) and/or rounding errors.

The *rounding errors* of the data may have so large effects on the high order differences[37] that the error estimation does not make sense. One should then use a smaller value of the order k, where the rounding errors have a smaller influence. An advantage with the use of a difference scheme is that it is relatively easy to choose the order k adaptively, and sometimes also the step size h.

This comment is of particular importance for numerical differentiation. Numerical illustrations and further comments are given below in Example 3.3.6 and Problem 6b, and in several other places.

The sequence of approximations to Lf may converge or diverge, depending on f and h. It is also often *semiconvergent*, recall Sec. 3.2.6, but in practice the rounding errors mentioned in the previous comment, have often, though not always, taken over already, when the truncation error passes its minimum. See Problem 6b.

Example 3.3.6. *The Backwards Differentiation Formula.*

By Theorem 3.3.6, $e^{-hD} = 1 - \nabla$. We look upon this as a formal power series; the indeterminate is $Q_h = \nabla$. By Example 3.1.11,

$$L = hD = -\ln(1-\nabla) = \nabla + \frac{1}{2}\nabla^2 + \frac{1}{3}\nabla^3 + \dots \tag{3.3.24}$$

Verification of the assumptions of Theorem 3.3.7: [38]

(i) $\operatorname{sc}(\nabla) = 1 - e^{-h\alpha}$; the convergence radius is $\rho = 1$.

$$\operatorname{sc}(L) = \operatorname{sc}(hD) = h\alpha; \quad \sum_{j=1}^{\infty} \operatorname{sc}(\nabla)^j/j = -\ln(1-(1-e^{-h\alpha})) = h\alpha.$$

The convergence condition $|\operatorname{sc}(\nabla)| < 1$ reads $h\alpha > -\ln 2 = -0.69$ if α is real, $|h\omega| < \pi/3$ if $\alpha = i\omega$.

(ii) For $\alpha = 0$, $D\dfrac{\partial^n}{\partial \alpha^n}(e^{\alpha x}) = Dx^n = nx^{n-1}$. By Leibniz' rule:

$$\frac{\partial^n}{\partial \alpha^n}(\alpha e^{\alpha x}) = 0x^n + nx^{n-1}.$$

By the theorem, we now obtain *a formula for numerical differentiation that is exact for all $f \in \mathcal{P}_k$.*

$$hf'(x) = \Big(\nabla + \frac{1}{2}\nabla^2 + \frac{1}{3}\nabla^3 + \dots + \frac{1}{k-1}\nabla^{k-1}\Big) f(x) \tag{3.3.25}$$

[37] Recall Example 3.3.2

[38] Recall the definition of the scalar $\operatorname{sc}(\cdot)$, after (3.3.13).

By Theorem 3.3.4, this is the *unique* formula of this type that uses the values of $f(x)$ at the k points $x_n : -h : x_{n-k+1}$. The same approximation can be derived in many other ways, perhaps with a different appearance; see Chapter 4. This derivation has several advantages; the same expansion yields approximation formulas for every k, and *if $f \in C^k$, $f \notin \mathcal{P}_k$, the first neglected term, i.e. $\frac{1}{k}\nabla_h^k f(x_n)$, provides an* **asymptotic error estimate**, if $f^{(k)}(x_n) \neq 0$.

We now apply this formula to the table in Example 3.3.2, where $f(x) = \tan x$, $h = 0.01$, $k = 6$,

$$0.01f'(1.35) \approx 0.1996 + \frac{0.0163}{2} + \frac{0.0019}{3} + \frac{0.0001}{4} - \frac{0.0004}{5},$$

i.e. we obtain a sequence of approximate results,

$$f'(1.35) \approx 19.96, \quad 20.78, \quad 20.84, \quad 20.84, \quad 20.83.$$

The correct value to 3D is $(\cos 1.35)^{-2} = 20.849$. Note that the last result is worse than the next to last. Recall the last comments to the theorem. In this case this is due to the rounding errors of the data. Upper bounds for their effect of the sequence of approximate values of $f'(1.35)$ is, by Example 3.3.3, shown in the series

$$10^{-2}\Big(1 + \frac{2}{2} + \frac{4}{3} + \frac{8}{4} + \frac{16}{5} + \dots\Big).$$

A larger version of this problem was run on a computer with the machine unit $2^{-53} \approx 10^{-16}$; $f(x) = \tan x$, $x = 1.35 : -0.01 : 1.06$. In the beginning the error decreases rapidly, but after 18 terms the rounding errors take over, and the error then grows almost exponentially (with constant sign). The eighteenth term and its rounding error have almost the same modulus (but opposite sign). The smallest error equals $5\,10^{-10}$, and is obtained after 18 terms; after 29 terms the actual error has grown to $2\,10^{-6}$. Such a large number of terms is seldom used in practice, unless a very high accuracy is demanded. See also Problem 6b, a computer exercise that offers both similar and different experiences.

Equation (3.3.24)—or its variable step size variant in Chapter 4—is called the **Backwards Differentiation Formula**. It is the basis of the important BDF method for the numerical integration of ordinary differential equations.

Coefficients for *Backwards differentiation formulas for higher derivatives*, are obtained from the equations

$$(hD/\nabla)^k = (-\ln(1-\nabla)/\nabla)^k.$$

The following formulas were computed by means of the matrix representation of a truncated power series:

$$\begin{pmatrix} hD/\nabla \\ (hD/\nabla)^2 \\ (hD/\nabla)^3 \\ (hD/\nabla)^4 \\ (hD/\nabla)^5 \end{pmatrix} = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1 & 1 & 11/12 & 5/6 & 137/180 \\ 1 & 3/2 & 7/4 & 15/8 & 29/15 \\ 1 & 2 & 17/6 & 7/2 & 967/240 \\ 1 & 5/2 & 25/6 & 35/6 & 1069/144 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \nabla \\ \nabla^2 \\ \nabla^3 \\ \nabla^4 \end{pmatrix}. \qquad (3.3.26)$$

The rows of the matrix are the first rows taken from the matrix representation of each of the expansions $(hD/\nabla)^k$, $k = 1:5$.

When the effect of the *irregular errors* of the data on a term becomes larger in magnitude than the term itself, the term should, of course, be neglected; it does more harm than good. This happens relatively early for the derivatives of high order; see Problem 6. When these formulas are to be used inside a program (rather than during an interactive post-processing of results of an automatic computation), some rules for *automatic truncation* have to be designed; an interesting kind of detail in scientific computing.

The *forwards differentiation formula*, which is analogously based on the operator series,

$$hD = \ln(1+\Delta) = \Delta - \frac{1}{2}\Delta^2 + \frac{1}{3}\Delta^3 \pm \dots \tag{3.3.27}$$

is sometimes useful too. We obtain the coefficients for derivatives of higher order by inserting minus signs in the second and fourth columns of the matrix in (3.3.26).

A grid (or a table) may be too sparse to be useful for numerical differentiation and for the computation of other linear functionals. For example, we saw above that the successive backward differences of $e^{i\omega x}$ increase exponentially if $|\omega h| > \pi/3$. In such a case the grid, where the values are given, gives insufficient information about the function. One also says that "the grid does not *resolve* the function". This is often indicated by a strong variation in the higher differences. However, even this indication can sometimes be absent. An extreme example is, $f(x) = \sin(\pi x/h)$, on the grid $x_j = jh$, $j = 0, \pm 1, \pm 2, \dots$. All the values, all the higher differences, and thus the estimates of $f'(x)$ at all grid points are zero, but the correct values of $f'(x_j)$ are certainly not zero. So, this is an example where the expansion (trivially) converges, but it is not valid! (Recall the discussion of a Maclaurin expansion for a non-analytic function at the end of Sec. 3.1.3. Now a similar trouble can occur also for an analytic function.)

A less trivial example is given by the functions

$$f(x) = \sum_{n=1}^{20} a_n \sin(2\pi n x), \quad g(x) = \sum_{n=1}^{10} (a_n + a_{10+n}) \sin(2\pi n x).$$

$f(x) = g(x)$ on the grid, hence they have the same difference scheme, but $f'(x) \neq g'(x)$ on the grid, and typically $f(x) \neq g(x)$ between the grid points.

3.3.3 The Peano Theorem

One can often, by a combination of theoretical and numerical evidence, rely on asymptotic error estimates. Since there are exceptions, it is interesting that there are two general methods for deriving strict error bounds. We call one of them the **norms and distance formula**. It is not restricted to polynomial approximation, and it is typically easy to use, but it requires some advanced concepts and it often overestimates the error. We therefore postpone the presentation of that method to a later chapter.

We shall now present another method, due to Peano[39]. Consider a linear functional $\tilde{L}$, e.g., $\tilde{L}f = \sum_{j=1}^{p} b_j f(x_j)$, suggested for the approximate computation of another linear functional L, e.g., $Lf = \int_0^1 \sqrt{x} f(x)dx$. Suppose that it is exact, when it is applied to any polynomial of degree less than k: In other words, $\tilde{L}f = Lf$, for all $f \in \mathcal{P}_k$. The remainder is then itself a linear functional, $R = L - \tilde{L}$, with the special property that

$$Rf = 0 \quad \text{if} \quad f \in \mathcal{P}_k.$$

Next theorem gives a representation for such functionals, which provides a universal device for deriving error bounds for approximations of the type that we are concerned with. Let $f \in C^n[a,b]$. In order to make the discussion less abstract we confine it to functionals of the following form, $0 \le m < n$,

$$Rf = \int_a^b \phi(x)f(x)dx + \sum_{j=1}^{p} \big(b_{j,0}f(x_j) + b_{j,1}f'(x_j) + \ldots + b_{j,m}f^{(m)}(x_j)\big), \quad (3.3.28)$$

where the function ϕ is integrable, and the points x_j lie in the bounded real interval $[a,b]$, and $b_{j,m} \neq 0$ for at least one value of j. Moreover, *we assume that*

$$Rp = 0 \quad \text{for all} \quad p \in \mathcal{P}_k. \quad (3.3.29)$$

We define the function[40]

$$t_+ = \max(t,0); \quad t_+^j = \big(t_+\big)^j; \quad t_+^0 = \frac{1+\operatorname{sign} t}{2}; \quad (3.3.30)$$

The function t_+^0 is often denoted $H(t)$ an is known as the **Heaviside**[41] **unit step function**. The function sign is defined in Definition def3.1.sign. Note that $t_+^j \in C^{j-1}$, $(j \ge 1)$. The **Peano kernel** $K(u)$ of the functional R is defined by the equation,

$$K(u) = \frac{1}{(k-1)!} R_x (x-u)_+^{k-1}, \quad x \in [a,b], \quad u \in (-\infty, \infty). \quad (3.3.31)$$

The subscript in R_x indicates that R acts on the variable x (not u).

The function $K(u)$ *vanishes outside* $[a,b]$, because:

- if $u > b$ then $u > x$, hence $(x-u)_+^{k-1} = 0$ and $K(u) = 0$,
- if $u < a$ then $x > u$. It follows that $(x-u)_+^{k-1} = (x-u)^{k-1} \in \mathcal{P}_k$, hence $K(u) = 0$, by (3.3.31) and (3.3.29).

If $\phi(x)$ is a polynomial then $K(u)$ becomes a piecewise polynomial; the points x_j are the joints of the pieces. In this case $K \in C^{k-m-2}$; the order of differentiability may be lower, if ϕ has singularities.

We are now in a position to prove an important theorem.

[39] Giuseppe Peano (1858-1932), Italian mathematician and logician.

[40] We use the neutral notation t here for the variable, to avoid to tie up the function too closely with the variables x and u which play a special role in the following.

[41] Oliver Heaviside (1850–1925) English physicist.

Theorem 3.3.8. *Peano's Remainder Theorem.*

Suppose that $Rp = 0$ for all $p \in \mathcal{P}_k$. Then [42] *, for all $f \in C^k[a,b]$,*

$$Rf = \int_{-\infty}^{\infty} f^{(k)}(u) K(u) du. \tag{3.3.32}$$

The definition and some basic properties of the Peano kernel $K(u)$ were given above.

Proof. By Taylor's formula,

$$f(x) = \sum_{j=1}^{k-1} \frac{f^{(j)}(a)}{j!}(x-a)^j + \int_a^x \frac{f^{(k)}(u)}{(k-1)!}(x-u)^{k-1} du.$$

This follows from putting $n = k$, $z = x - a$, $t = (u-a)/(x-u)$ into (3.1.5). We rewrite the last term as $\int_a^\infty f^{(k)}(u)(x-u)_+^{k-1} du$. Then apply the functional $R = R_x$ to both sides. Since we can allow the interchange of the functional R with the integral, for the class of functionals that we are working with, this yields

$$Rf = 0 + R\int_a^\infty \frac{f^{(k)}(u)(x-u)_+^{k-1}}{(k-1)!} du = \int_a^\infty \frac{f^{(k)}(u) R_x (x-u)_+^{k-1}}{(k-1)!} du,$$

The theorem then follows from (3.3.31). □

Corollary 3.3.9.

Suppose that $Rp = 0$ for all $p \in \mathcal{P}_k$. Then

$$R_x(x-a)^k = k! \int_{-\infty}^{\infty} K(u) du. \tag{3.3.33}$$

For any $f \in C^k[a,b]$, $Rf = \frac{f^{(k)}(\xi)}{k!} R_x((x-a)^k)$, holds for some $\xi \in (a,b)$, if and only if $K(u)$ does not change its sign.

If $K(u)$ changes its sign, the best possible error bound *reads*

$$|Rf| \le \sup_{u\in[a,b]} |f^{(k)}(u)| \int_{-\infty}^{\infty} |K(u)| du;$$

a formula with $f^{(k)}(\xi)$ is not generally true in this case.

Proof. First suppose that $K(u)$ does not change sign. Then, by (3.3.32) and the mean value theorem of Integral Calculus, $Rf = f^{(k)}(\xi)\int_{-\infty}^{\infty} K(u)du$, $\xi \in [a,b]$. For $f(x) = (x-a)^k$ this yields (3.3.33). The "if" part of the corollary follows from the combination of these formulas for Rf and $R(x-a)^k$.

If $K(u)$ changes its sign, the "best possible bound" is approached by a sequence of functions f chosen so that (the continuous functions) $f^{(k)}(u)$ approach (the discontinuous function) sign $K(u)$. The "only if" part follows. □

[42] The definition of $f^{(k)}(u)$ for $u \notin [a,b]$ is arbitrary.

Example 3.3.7.

The remainder of the *trapezoidal rule* (one step of length h) reads $Rf = \int_0^h f(x)dx - \frac{h}{2}(f(h)+f(0))$. We know that $Rp = 0$ for all $p \in \mathcal{P}_2$. The Peano kernel is zero for $u \notin [0,h]$, while for $u \in [0,h]$,

$$K(u) = \int_0^h (x-u)_+ dx - \frac{h}{2}((h-u)_+ + 0)) = \frac{(h-u)^2}{2} - \frac{h(h-u)}{2} = \frac{-u(h-u)}{2} < 0$$

We also compute

$$\frac{Rx^2}{2!} = \int_0^h \frac{x^2}{2}dx - \frac{h \cdot h^2}{2 \cdot 2} = \frac{h^3}{6} - \frac{h^3}{4} = -\frac{h^3}{12}.$$

Since the Peano kernel does not change sign, we conclude that

$$Rf = -\frac{h^3}{12}f''(\xi), \quad \xi \in (0,h).$$

Example 3.3.8. *Peano kernels for difference operators.*

Let $Rf = \Delta^3 f(a)$, and set $x_i = a + ih$, $i = 0:3$. Note that $Rp = 0$ for $p \in \mathcal{P}_3$. Then

$$\begin{aligned} Rf &= f(x_3) - 3f(x_2) + 3f(x_1) - f(x_0), \\ 2K(u) &= (x_3-u)_+^2 - 3(x_2-u)_+^2 + 3(x_1-u)_+^2 - (x_0-u)_+^2, \end{aligned}$$

i.e.

$$2K(u) = \begin{cases} 0, & \text{if } u > x_3; \\ (x_3-u)^2, & \text{if } x_2 \le u \le x_3; \\ (x_3-u)^2 - 3(x_2-u)^2, & \text{if } x_1 \le u \le x_2; \\ (x_3-u)^2 - 3(x_2-u)^2 + 3(x_1-u)^2 \equiv (u-x_0)^2, & \text{if } x_0 \le u \le x_1; \\ (x_3-u)^2 - 3(x_2-u)^2 + 3(x_1-u)^2 - (x_0-u)^2 \equiv 0, & \text{if } u < x_0. \end{cases}$$

For the simplification of the last two lines we used that $\Delta_u^3 (x_0-u)^2 \equiv 0$. Note that $K(u)$ is a piecewise polynomial in $\mathcal{P}_3$ and that $K''(u)$ is discontinuous at $u = x_i$, $i = 0:3$.

It can be shown (numerically or analytically) that $K(u) > 0$ in the interval (u_0, u_3). This is no surprise, for, by (3.3.6), $\Delta^n f(x) = h^n f^{(n)}(\xi)$ for any integer n, and, by the above corollary, this could not be generally true if $K(u)$ changes its sign. These calculations can be generalized to $\Delta^k f(a)$ for an arbitrary integer k. This example will be generalized in Sec. 4.2.5 to divided differences of non-equidistant data.

In general it is rather laborious to determine a Peano kernel. Sometimes one can show that the kernel is piecewise a polynomial, that it has a symmetry, and that has a simple form in the intervals near the boundaries. All this can simplify the computation, and might have been used in these examples.

It is usually much easier to compute $R((x-a)^k)$, and an *approximate error estimate* is often given by

$$Rf \sim \frac{f^{(k)}(a)}{k!} R\big((x-a)^k\big), \quad f^{(k)}(a) \neq 0. \tag{3.3.34}$$

For example, suppose that $x \in [a,b]$, where $b-a$ is of the order of magnitude of a step size parameter h, and that f is analytic in $[a,b]$. By Taylor's formula,

$$f(x) = p(x) + \frac{f^{(k)}(a)}{k!}(x-a)^k + \frac{f^{(k+1)}(a)}{(k+1)!}(x-a)^{k+1} + \ldots, \quad f^{(k)}(a) \neq 0,$$

where $p \in \mathcal{P}_k$, hence $Rp = 0$. Most of the common functionals can be applied term by term. Then

$$Rf = 0 + \frac{f^{(k)}(a)}{n!} R_x(x-a)^k + \frac{f^{(k+1)}(a)}{(k+1)!} R_x(x-a)^{k+1} + \ldots.$$

Assume that, for some c, $R_x(x-a)^k = O(h^{k+c})$, for $k = 1, 2, 3, \ldots$. (This is often the case.) Then (3.3.34) becomes an **asymptotic error estimate** as $h \to 0$. It was mentioned above that for formulas derived by operator methods, an asymptotic error estimate is directly available anyway, but if a formula is derived by other means (see Chapter 4) this error estimate is important.

Asymptotic error estimates are frequently used in computing, because they are often much easier to to derive and apply than strict error bounds. The question is, however, how to know (or feel), that "the computation is in the asymptotic regime", where an asymptotic estimate is practically reliable. Much can be said about this central question of Applied Mathematics. Let us her just mention that a difference scheme displays well the quantitative properties of a function needed for the judgment. If $Rp = 0$ for $p \in \mathcal{P}_k$, then a fortiori $Rp = 0$ for $p \in \mathcal{P}_{k-i}$, $i = 0:k$. We may thus obtain a Peano kernel for each i, which is temporarily denoted by $K_{k-i}(u)$. They are obtained by integration by parts,

$$R_k f = \int_{-\infty}^{\infty} K_k(u) f^{(k)}(u)\, du = \int_{-\infty}^{\infty} K_{k-1}(u) f^{(k-1)}(u)\, du \tag{3.3.35}$$

$$= \int K_{k-2}(u) f^{(k-2)}(u)\, du \ldots, \tag{3.3.36}$$

where $K_{k-i} = (-D)^i K_k$, $i = 1, 2, \ldots$, *as long as* K_{k-i} *is integrable. The lower order kernels are useful, e.g., if the actual function* f *is not as smooth as the usual remainder formula requires.*

For the trapezoidal rule we obtained in Example 3.3.7

$$K_1(u) = \frac{h}{2} u_+^0 + \frac{h}{2} - u + \frac{h}{2}(u-h)_+^0.$$

A second integration by parts can only be performed within the framework of Dirac's delta functions (distributions); K_0 is not integrable. A reader, who is familiar with

these generalized functions, may enjoy the following formula:

$$Rf = \int_{-\infty}^{\infty} K_0(u) f(u) du \equiv \int_{-\infty}^{\infty} \Big(-\frac{h}{2}\delta(u) + 1 - \frac{h}{2}\delta(u-h) \Big) f(u) du.$$

This is for one step of the trapezoidal rule, but many functionals can be expressed analogously.

3.3.4 Approximation Formulas by Operator Methods

We shall now demonstrate how operator methods can be applied for deriving approximation formulas.

In order to find interpolation formulas by operator methods we consider the operator expansion

$$f(b-\gamma h) = E^{-\gamma} f(b) = (1-\nabla)^{\gamma} f(b) = \sum_{j=0}^{\infty} \binom{\gamma}{j} (-\nabla)^j f(b).$$

The verification of the assumptions of Theorem 3.3.7 offers no difficulties, and we omit the details. Truncate the expansion before $(-\nabla)^k$. By the theorem we obtain, for every γ:

- an approximation formula for $f(b-\gamma h)$ that uses the function values $f(b-jh)$ for $j = 0 : k-1$; it is exact if $f \in \mathcal{P}_k$, and is unique in the sense of Theorem 3.3.4;
- an asymptotic error estimate if $f \notin \mathcal{P}_k$, namely the first neglected term of the expansion, i.e.

$$\binom{\gamma}{k} (-\nabla)^k f(b) \sim \binom{\gamma}{k} (-h)^k f^{(k)}(b)$$

Note that the binomial coefficients are polynomials in the variable γ, and hence also in the variable $x = b - \gamma h$. It follows that the approximation formula yields **a unique polynomial** $P_B \in \mathcal{P}_k$, that solves the **interpolation problem**: $P_B(b-hj) = f(b-hj),\ j = 0 : k-1$; ($B$ stands for Backward). If we set $x = b - \gamma h$, we obtain

$$\begin{aligned} P_B(x) &= E^{-\gamma} f(b) = (1-\nabla)^{\gamma} f(a) \\ &= \sum_{j=0}^{k-1} \binom{\gamma}{j} (-\nabla)^j f(b) = f(b-\gamma h) + O(h^k f^{(k)}). \end{aligned} \tag{3.3.37}$$

Due to the uniqueness; see the corollary of Theorem 3.3.4, the approximation to $f'(b)$ obtained by the first $k-1$ terms in Example 3.2.4 for $x_n = b$ is exactly the derivative $P_B'(b)$.

Similarly, the interpolation polynomial $P_F \in \mathcal{P}_k$ that uses *forward* differences based on the values of f at $a, a+h, \ldots, a+(k-1)h$, reads, if we set $x = a+\theta h$,

$$P_F(x) = E^\theta f(a) = (1+\Delta)^\theta f(a) = \sum_{j=0}^{k-1} \binom{\theta}{j} \Delta^j f(a) = f(a+\theta h) + O(h^k f^{(k)}). \tag{3.3.38}$$

These formulas are known as **Newton's interpolation formulas for constant step size**, backwards and forwards. The generalization to variable step size will be found in Sec. 4.2.

There exists a similar expansion for *central differences*. Set

$$\phi_0(\theta) = 1, \quad \phi_1(\theta) = \theta, \quad \phi_j(\theta) = \frac{\theta}{j}\binom{\theta + \frac{1}{2}j - 1}{j-1}, \quad (j > 1). \tag{3.3.39}$$

ϕ_j is an even function if j is even, and an odd function if j is odd. It can be shown that $\delta^j \phi_k(\theta) = \phi_{k-j}(\theta)$, and $\delta^j \phi_k(0) = \delta_{j,k}$, (Kronecker's delta). The functions ϕ_k have thus an analogous relation to the operator δ as, e.g., the functions $\theta^j/j!$ and $\binom{\theta}{j}$ have to the operators D and Δ, respectively. We obtain the following expansion, analogous to Taylor's formula and Newton's forward interpolation formula. The proof is left for Problem 4(b). Then

$$E^\theta f(a) = \sum_{j=0}^{k-1} \phi_j(\theta) \delta^j f(a) = f(a+\theta h) + O(h^k f^{(k)}). \tag{3.3.40}$$

The direct practical importance of this formula is small, since $\delta^j f(a)$ cannot be expressed as a linear combination of the given data when j is odd. There are several formulas, where this drawback has been eliminated by various transformations. They were much in use before the computer age; each formula had its own group of fans. We shall derive only one of them, by a short break-neck application of the formal power series techniques.[43] Note that

$$\begin{aligned}
E^\theta &= e^{\theta h D} = \cosh \theta h D + \sinh \theta h D, \\
\delta^2 &= e^{hD} - 2 + e^{-hD}, \qquad e^{hD} - e^{-hD} = 2\mu\delta, \\
\cosh \theta h D &= \tfrac{1}{2}(E^\theta + E^{-\theta}) = \sum_{j=0}^{\infty} \phi_{2j}(\theta) \delta^{2j}, \\
\sinh \theta h D &= \frac{1}{\theta} \frac{d(\cosh \theta h D)}{d(hD)} = \sum_{j=0}^{\infty} \phi_{2j}(\theta) \frac{1}{\theta} \frac{d\delta^{2j}}{d\delta^2} \frac{d\delta^2}{d(hD)} \\
&= \sum_{j=0}^{\infty} \phi_{2j}(\theta) \frac{j\delta^{2(j-1)}}{\theta} (e^{hD} - e^{-hD}) = \sum_{j=0}^{\infty} \phi_{2j}(\theta) \frac{2j}{\theta} \mu \delta^{2j-1}.
\end{aligned}$$

[43] Differentiation of a formal power series with respect to an indeterminate has a purely algebraic definition. See the last part of Sec. 3.1.5.

Hence,

$$f(x_0+\theta h) = f_0 + \theta\mu\delta f_0 + \frac{\theta^2}{2!}\delta^2 f_0 + \sum_{j=2}^{\infty} \phi_{2j}(\theta)\Big(\frac{2j}{\theta}\mu\delta^{2j-1} f_0 + \delta^{2j} f_0\Big). \quad (3.3.41)$$

This is known as **Stirling's interpolation formula**. [44] The first three terms have been taken out from the sum, in order to show their simplicity and their resemblance to Taylor's formula. They yield the most practical formula for quadratic interpolation; it is easily remembered and worth to be remembered. An approximate error bound for this quadratic interpolation reads $|0.016\delta^3 f|$ if $|\theta| < 1$.

Note that

$$\phi_{2j}(\theta) = \theta^2(\theta^2-1)(\theta^2-4)\cdots(\theta^2-(j-1)^2)/(2j)!.$$

The expansion yields a true interpolation formula if it is truncated after an *even* power of δ. For $k = 1$ you see that $f_0+\theta\mu\delta f_0$ is not a formula for linear interpolation; it uses three data points instead of two. It is similar for all odd values of k.

Strict error bounds can be found by means of Peano's theorem, but the remainder terms given in Sec. 4.2 for Newton's general interpolation formula (that does not require equidistant data) typically give the answer easier. Both are typically of the form $c_{k+1}f^{(k+1)}(\xi)$ and require a bound for a derivative of high order. The assessment of such a bound typically costs much more work than performing interpolation in one point.

A more practical approach is to estimate a bound for this derivative by means of a bound for the differences of the same order. (Recall the important formula in (3.3.6).) This is not a rigorous *bound*, but it typically yields a quite reliable error *estimate*, in particular if you put a moderate safety factor on the top of it. There is much more to be said about the choice of step size and order; we shall return to this kind of questions in later chapters.

You can make error estimates during the run; it can happen sooner or later that it does not decrease, when you increase the order. You may just as well stop there, and accept the most recent value as the result. This event is most likely due to the influence of irregular errors, e.g. rounding errors, but it can also indicate that the interpolation process is semi-convergent only.

The attainable accuracy of polynomial interpolation applied to a table with n equidistant values of an analytic function, depends strongly on θ; the results are much poorer near the boundaries of the data set than near the center. This question will be illuminated in Sec. 4.8 by means of complex analysis.

Example 3.3.9.

The continuation of the difference scheme of a polynomial is a classical application of a difference scheme for obtaining a smooth extrapolation of a function outside its original domain. Given the values $y_{n-i} = f(x_n - ih)$ for $i = 1 : k$ and

[44] James Stirling (1692–1770), British mathematician, perhaps most famous for his amazing approximation to $n!$.

the backward differences, $\nabla^j y_{n-1},\ j = 1 : k-1$. Recall that $\nabla^{k-1} y$ is a constant for $y \in \mathcal{P}_k$. Consider the algorithm

$$\begin{aligned}
&\nabla^{k-1} y_n = \nabla^{k-1} y_{n-1};\\
&\textbf{for } j = k-1 : -1 : 1,\\
&\quad \nabla^{j-1} y_n = \nabla^{j-1} y_{n-1} + \nabla^j y_n;\\
&\textbf{end}\\
&y_n = \nabla^0 y_n;
\end{aligned} \tag{3.3.42}$$

It is left for Problem 7a to show that the result y_n is the value at $x = x_n$ of the interpolation polynomial which is determined by $y_{n-i},\ i = 1 : k$. This is a kind of inverse use of a difference scheme; there are additions from right to left along a diagonal, instead of subtractions from left to right.

This algorithm, which needs additions only, was used long ago for the production of mathematical tables, e.g., for logarithms. Suppose that one knows, e.g., by means of a series expansion, a relatively complicated polynomial approximation to (say) $f(x) = \ln x$, that is accurate enough in (say) the interval $[a, b]$, and that this has been used for the computation of k very accurate values $y_0 = f(a),\ y_1 = f(a+h), \ldots y_{k-1}$, needed for starting the difference scheme. The algorithm is then used for $n = k,\ k+1,\ k+2, \ldots,\ (b-a)/h$. $k-1$ additions only are needed for each value y_n. Some analysis must have been needed for the choice of the step h to make the tables useful with (say) linear interpolation, and for the choice of k to make the basic polynomial approximation accurate enough over a substantial number of steps. The precision used was higher, when the table was produced than when it was used. When $x = b$ was reached, a new approximating polynomial was needed for continuing the computation over an other interval; at least a new value of $\nabla^{k-1} y_n$.

This procedure was the basis of the unfinished Difference Engine project of the great 19th century British computer pioneer Charles Babbage. He abandoned it after a while in order to spend more time on his huge "Analytic Engine" project, which was also unfinished, but he documented a lot of ideas, where he was (say) 100 years ahead of his time. "Difference engines" based on Babbage's ideas were, however, constructed in Babbage's own time, by the Swedish inventors Scheutz (father and son) 1834 and by Wiberg 1876, and they were applied, among other things, to the automatic calculation and printing of tables of logarithms. See, e.g., Goldstine [21].

The algorithm in (3.3.42) can be generalized to the non-equidistant with the use of divided differences; see Sec. 4.2.1.

We now derive some central difference formulas for numerical differentiation. From the definition and from Bickley's table (Table 3.2.1)

$$\delta \equiv E^{1/2} - E^{-1/2} = 2 \sinh\Big(\frac{1}{2} hD\Big). \tag{3.3.43}$$

We may therefore put $x = \frac{1}{2} hD$, $\sinh x = \frac{1}{2}\delta$ into the following expansion (see

Problem 3.1.7),

$$x = \sinh x - \frac{1}{2}\frac{\sinh^3 x}{3} + \frac{1\cdot 3}{2\cdot 4}\frac{\sinh^5 x}{5} - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\frac{\sinh^7 x}{7} \pm \dots,$$

with the result

$$hD = 2\text{arcsinh}\frac{\delta}{2} = \delta - \frac{\delta^3}{24} + \frac{3\delta^5}{640} - \frac{5\delta^7}{7,168} + \frac{35\delta^9}{294,912} - \frac{63\delta^{11}}{2,883,584} \pm \dots. \quad (3.3.44)$$

The verification of the assumptions of Theorem 3.3.7 follows the pattern of Example 3.3.6, and we omit the details. Since $\text{arcsinh} z$, $z \in \mathbf{C}$ has the same singularities as its derivative $(1+z^2)^{-1/2}$, namely $z = \pm i$, it follows that the expansion in (3.3.44), if $\text{sc}(\delta/2)$ is substituted for $\delta/2$, converges if $\text{sc}(\delta/2) < 1$, hence $\rho = 2$.

By squaring the above relation, we obtain

$$(hD)^2 = \delta^2 - \frac{\delta^4}{12} + \frac{\delta^6}{90} - \frac{\delta^8}{560} + \frac{\delta^{10}}{3,150} - \frac{\delta^{12}}{16,632} \pm \dots,$$
$$f''(x_0) \approx \left(1 - \frac{\delta^2}{12} + \frac{\delta^4}{90} - \frac{\delta^6}{560} + \frac{\delta^8}{3,150} - \frac{\delta^{10}}{16,632} \pm \dots\right)\frac{\delta^2 f_0}{h^2}. \quad (3.3.45)$$

By Theorem 3.3.7 (3.3.45) holds for all polynomials. Since the first neglected nonvanishing term of (3.3.45) when applied to f, is (asymptotically) $c\delta^{12} f''(x_0)$, the formula for $f''(x)$ is exact if $f'' \in \mathcal{P}_{12}$, i.e. if $f \in \mathcal{P}_{14}$, although only 13 values of $f(x)$ are used. We thus gain one degree and, in the application to other functions than polynomials, one order of accuracy, compared to what we may have expected by counting unknowns and equations only; see Theorem 3.3.4. *This is typical for a problem that has a symmetry with respect to the hull of the data points.*

Suppose that the values $f(x)$ are given on the grid $x = x_0 + nh$, n integer. Since (3.3.44) contains odd powers of δ, it cannot be used to compute f'_n on the same grid. as pointed out in the beginning of Sec. 3.3.2. This difficulty can be overcome by means of another formula given in Bickley's table, namely

$$\mu = \sqrt{1 + \delta^2/4}. \quad (3.3.46)$$

This is derived as follows. The formulas

$$\mu = \cosh\frac{hD}{2}, \qquad \frac{\delta}{2} = \sinh\frac{hD}{2}$$

follow rather directly from the definitions; the details are left for Problem 5a. The formula $(\cosh hD)^2 - (\sinh hD)^2 = 1$ holds also for formal power series. Hence

$$\mu^2 - \frac{1}{4}\delta^2 = 1, \quad \text{or} \quad \mu^2 = 1 + \frac{1}{4}\delta^2,$$

from which the relation (3.3.46) follows.

If we now multiply the right hand side of equation (3.3.44) by the expansion

$$1 = \mu\Big(1 + \frac{1}{4}\delta^2\Big)^{-1/2} = \mu\Big(1 - \frac{\delta^2}{8} + \frac{3\delta^4}{128} - \frac{5\delta^6}{1,024} + \frac{35\delta^8}{32,768} + \dots\Big). \quad (3.3.47)$$

we obtain

$$hD = \Big(1 - \frac{\delta^2}{6} + \frac{\delta^4}{30} - \frac{\delta^6}{140} \pm \ldots\Big)\mu\delta. \tag{3.3.48}$$

This leads to a useful central difference formula for the first derivative (where we have used more terms than we displayed in the above derivation).

$$f'(x_0) = \Big(1 - \frac{\delta^2}{6} + \frac{\delta^4}{30} - \frac{\delta^6}{140} + \frac{\delta^8}{630} - \frac{\delta^{10}}{2772} \pm \ldots\Big)\frac{f_1 - f_{-1}}{2h}. \tag{3.3.49}$$

If you truncate the operator expansion in (3.3.49) after the δ^{2k} term, you obtain exactly the derivative of the interpolation polynomial of degree $2k+1$ for $f(x)$ that is determined by the $2k+2$ values f_i, $i = \pm1, \pm2, \ldots, \pm(k+1)$. Note that all the neglected terms in the expansion vanish when $f(x)$ is any polynomial of degree $2k+2$, independent of the value of f_0. (Check the statements first for $k = 0$; you will recognize a familiar property of the parabola.) So, although we search for a formula that is exact in $\mathcal{P}_{2k+2}$, we actually find a formula that is exact in $\mathcal{P}_{2k+3}$.

By the multiplication of the expansions in (3.3.45) and (3.3.48), we obtain the following formulas, which have applications in other sections

$$\begin{aligned}(hD)^3 &= (1 - \frac{1}{4}\delta^2 + \frac{7}{120}\delta^4 + \ldots)\mu\delta^3\\ (hD)^5 &= (1 - \frac{1}{3}\delta^2 + \ldots)\mu\delta^5\\ (hD)^7 &= \mu\delta^7 + \ldots\end{aligned} \tag{3.3.50}$$

Another valuable feature typical for δ^2-*expansions*, i.e. for expansions in powers of δ^2, is the rapid convergence. It was mentioned earlier that $\rho = 2$, hence $\rho^2 = 4$, (while $\rho = 1$ for the backwards differentiation formula). The error constants of the differentiation formulas obtained by (3.3.45) and (3.3.49) are thus relatively small.

All this is typical for the symmetric approximation formulas which are based on central differences; see, e.g., the above formula for $f''(x_0)$, or the next example. In view of this, can we forget the forward and backward difference formulas altogether? Well, this is not quite the case, since one must often deal with data that are unsymmetric with respect to the point where the result is needed. For example, given f_{-1}, f_0, f_1, how would you compute $f'(x_1)$? Asymmetry is also typical for the application to *initial value problems* for differential equations; see Volume III. In such applications methods based on symmetric rules for differentiation or integration have sometimes inferior properties of numerical stability.

When a problem has a symmetry around some point x_0, you are advised to try to derive a δ^2-expansion. The first step is to express the relevant operator in the form $\Phi(\delta^2)$, where the function Φ is analytic at the origin.

To find a δ^2-expansion for $\Phi(\delta^2)$ is algebraically the same thing as expanding $\Phi(z)$ into powers of a complex variable z. So, the methods for the manipulation of power series mentioned in Sec. 3.2.2 and Problem 3.1.8 are available, and so is the Cauchy–FFT method (Sec. 3.1.4). *For suitably chosen r, N you evaluate*

$$\Phi(re^{2\pi ik/N}), \quad k = 0 : N-1,$$

and obtain the coefficients of the δ^2-expansion by the FFT! You can therefore derive a long expansion, and later truncate it as needed. You also obtain error estimates for all these truncated expansions for free.

Suppose that you have found a truncated δ^2-expansion, (say)

$$A(\delta^2) \equiv a_1 + a_2\delta^2 + a_3\delta^4 + \ldots + a_{k+1}\delta^{2k},$$

but you want instead an equivalent symmetric expression of the form

$$B(E) \equiv b_1 + b_2(E + E^{-1}) + b_3(E^2 + E^{-2}) + \ldots + b_{k+1}(E^k + E^{-k}).$$

Note that $\delta^2 = E - 2 + E^{-1}$. The transformation $A(\delta^2) \mapsto B(E)$ can be performed in several ways. Since it is linear it can be expressed by a matrix multiplication of the form $b = M_{k+1}a$, where $a,\ b$ are column vectors for the coefficients, and M_{k+1} is the $k+1 \times k+1$ upper triangular submatrix in the northwest corner of a matrix M that turns out to be

$$M = \begin{pmatrix} 1 & -2 & 6 & -20 & 70 & -252 & 924 & -3432 \\ & 1 & -4 & 15 & -56 & 210 & -792 & 3003 \\ & & 1 & -6 & 28 & -120 & 495 & -2002 \\ & & & 1 & -8 & 45 & -220 & 1001 \\ & & & & 1 & -10 & 66 & -364 \\ & & & & & 1 & -12 & 91 \\ & & & & & & 1 & -14 \\ & & & & & & & 1 \end{pmatrix}. \tag{3.3.51}$$

Note that the matrix elements are binomial coefficients that can be generated recursively (Problem 13). It is therefore easy to extend the matrix; this 8×8 matrix is sufficient for a δ^2-expansion up to the term $a_8\delta^{14}$.

The operator D^{-1} is defined by the relation $(D^{-1}f)(x) = \int^x f(t)\,dt$. The lower limit is not fixed, so $D^{-1}f$ contains an arbitrary integration constant. Note that $DD^{-1}f = f$, while $D^{-1}Df = f + C$, where C is the integration constant. A difference expression like $D^{-1}f(b) - D^{-1}f(a) = \int_a^b f(t)\,dt$ is uniquely defined. So is also $\delta D^{-1}f$, but $D^{-1}\delta f$ has an integration constant.

A right-hand inverse can be defined also for the operators Δ, ∇, δ. For example, $(\nabla^{-1}u)_n = \sum^{j=n} u_j$ has an arbitrary summation constant but, e.g., $\nabla\nabla^{-1} = 1$, and $\Delta\nabla^{-1} = E\nabla\nabla^{-1} = E$ are uniquely defined.

One can make the inverses unique by restricting the class of sequences (or functions). For example, if we require that $\sum_{j=0}^{\infty} u_j$ is convergent, and make the convention that $(\Delta^{-1}u)_n \to 0$ as $n \to \infty$, then $\Delta^{-1}u_n = -\sum_{j=n}^{\infty} u_j$; notice the minus sign. Also notice that this is consistent with the following formal computation: $(1 + E + E^2 + \ldots)u_n = (1 - E)^{-1}u_n = -\Delta^{-1}u_n$. We recommend, however, some extra care with infinite expansions into powers of operators like E that is not covered by Theorem 3.3.7, but the finite expansion

$$1 + E + E^2 + \ldots + E^{n-1} = (E^n - 1)(E - 1)^{-1} \tag{3.3.52}$$

is valid.

In Chapter 5 we will use operator methods together with the Cauchy–FFT method for finding the **Newton–Cotes'** formulas for symmetric numerical integration.

Operator techniques can also be extended to *functions of several variables*. The basic relation is again the operator form of Taylor's formula, which in the case of two variables reads,

$$u(x_0+h,y_0+k) = \exp\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)u(x_0,y_0)$$
$$= \exp\left(h\frac{\partial}{\partial x}\right)\exp\left(k\frac{\partial}{\partial y}\right)u(x_0,y_0). \tag{3.3.53}$$

There will be applications in some problems of Chapter 4.

3.3.5 Single Linear Difference Equations

Historically, the term **difference equation** was probably first used in connection with an equation of the form

$$b_0\Delta^k y_n + b_1\Delta^{k-1}y_n + \ldots b_{k-1}\Delta y_n + b_k y_n = 0, \quad n = 0,1,2,\ldots$$

which reminds of a linear homogeneous differential equation. It follows, however, from the discussion after (3.3.1) and (3.3.4) that this equation can also be written in the form

$$y_{n+k} + a_1 y_{n+k-1} + \ldots + a_k y_n = 0, \tag{3.3.54}$$

and nowadays this is what one usually means by a single homogeneous linear difference equation of kth order with *constant coefficients*; a difference equation without differences. More generally, if we let the coefficients a_i depend on n; we have a linear difference equation with *variable coefficients*. If we replace the zero on the right hand side with some known quantity r_n, we have a *nonhomogeneous* linear difference equation.

These types of equations are the main topic of this section. The coefficients and the unknown are real or complex numbers. We shall occasionally see examples of more general types of difference equations, e.g., a nonlinear difference equation $F(y_{n+k}, y_{n+k-1}, \ldots, y_n) = 0$), and we shall, in Chapter 13, deal with *first order systems* of difference equations, i.e. $y_{n+1} = A_n y_n + r_n$, where r_n, y_n, etc. are vectors while A_n is a square matrix. Finally, *partial difference equations* where you have two (or more) subscripts in the unknown, occur often as numerical methods for partial differential equations, but they have many other important applications too.

A difference equation can be viewed as a *recurrence relation*. With given values of $y_0,\ y_1, \ldots,\ y_{k-1}$, called the **initial values** or the **seed** of the recurrence, we can successively compute $y_k,\ y_{k+1},\ y_{k+2}, \ldots$; we see that *the general solution of a k'th order difference equation contains k arbitrary constants*, just like the general solution of the k'th order differential equation.

There are other important similarities between difference and differential equations, for example the following superposition result.

Lemma 3.3.10. *The general solution of a nonhomogeneous linear difference equation (also with variable coefficients) is the sum of one particular solution of it, and the general solution of the corresponding homogeneous difference equation.*

In practical computing, the recursive computation of the solution of a difference equations is most common. It was mentioned at the end of Sec. 3.1 that many important functions, e.g., Bessel functions and orthogonal polynomials, satisfy second order linear difference equations with variable coefficients, (although this terminology was not used there). Other important applications are the multistep methods for ordinary differential equations.

In such an application you are usually interested in the solution for one particular initial condition, but due to rounding errors in the initial values you obtain another solution. It is therefore of interest to know the behaviour of the solutions of the corresponding homogeneous difference equation. The questions are:

- *Can we use a recurrence to find the wanted solution accurately?*
- *How shall we use a recurrence, forward or backward?*

Forward recurrence is the type we described above. In backward recurrence we choose some large integer N, and give (almost) arbitrary values of y_{N+i}, $i = 0 : k-1$ as seed, and compute y_n for $n = N-1 : -1 : 0$.

We have seen this already in Example 1.3.3 (and in Problem 10a of Sec. 1.3) for an inhomogeneous first order recurrence relation. It was there found that the forward recurrence was useless, while backward recurrence, with a rather naturally chosen seed, gave satisfactory results; (see Example 1.3.4 and Problem 10b).

It is often like this, though not always. In Problem 9 of Sec. 1.3 it is the other way around: the forward recurrence is useful, and the backward recurrence is useless.

Sometimes **boundary values** are prescribed for a difference equation instead of initial values, (say) p values at the beginning and $q = k-p$ values at the end, e.g., the values of y_0, y_1,..., y_{p-1}, and $y_{N-q}, \ldots, _{N-1}$, y_N are given. Then the difference equation can be treated as a *linear system* with $N-k$ unknown. This also holds for a difference equation with variable coefficients and for an inhomogeneous difference equation. *From the point of view of numerical stability, such a treatment can be better than either recurrence.* The amount of work is somewhat larger, not very much though, for the matrix is a band matrix. We have sees in Example 1.3.2 that for a fixed number of bands *the amount of work to solve such a linear system is proportional to the number of unknown.* An important particular case is when $k = 2$, $p = q = 1$; the linear system is then tridiagonal. An algorithm for such linear systems is described in Example 1.3.2.

Another similarity for differential and difference equations, is that the general solution of a linear equation with constant coefficients has a simple closed form. Although, in most cases, the real world problems have variable coefficients (or are nonlinear), one can often formulate a class of model problems with constant coefficients, with similar features. The analysis of such model problems can give hints, e.g., whether forward or backward recurrence should be used, or other questions re-

lated to the design and the analysis of the numerical stability of a numerical method for a more complicated problem.

We shall therefore now study how to solve a *single homogeneous linear difference equation with constant coefficients* (3.3.54), i.e.

$$y_{n+k} + a_1 y_{n+k-1} + \ldots + a_k y_n = 0.$$

It is satisfied by the sequence $\{y_j\}$, where $y_j = cu^j$, $(u \neq 0,\ c \neq 0)$, if and only if $u^{n+k} + a_1 u^{n+k-1} + \ldots + a_k u^n = 0$, that is when

$$\phi(u) \equiv u^k + a_1 u^{k-1} + \ldots + a_k = 0. \tag{3.3.55}$$

Equation (3.3.55) is called the **characteristic equation** of (3.3.54); $\phi(u)$ is called the **characteristic polynomial**.

Theorem 3.3.11.

If the characteristic equation has k different *roots,* $u_1, \ldots, u_k$*, then the general solution of equation* (3.3.54) *is given by the sequences* $\{y_n\}$*, where*

$$y_n = c_1 u_1^n + c_2 u_2^n + \cdots + c_k u_k^n, \tag{3.3.56}$$

where $c_1, c_2, \ldots, c_k$ *are arbitrary constants.*

Proof. That $\{y_n\}$ satisfies equation (3.3.54) follows from the previous comments and from the fact that the equation is linear. The parameters $c_1, c_2, \ldots, c_k$ can be adjusted to arbitrary initial conditions $y_0, y_1, \ldots, y_{k-1}$ by solving the system of equations

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ u_1 & u_2 & \cdots & u_k \\ \vdots & \vdots & & \vdots \\ u_1^{k-1} & u_2^{k-1} & \cdots & u_k^{k-1} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{k-1} \end{pmatrix}.$$

The matrix is a Vandermonde matrix and its determinant is thus equal to the product of all differences $(u_i - u_j)$, $i \geq j$, $1 < i \leq k$, which is nonzero. □

Example 3.3.10.

Consider the difference equation $y_{n+2} - 5y_{n+1} + 6y_n = 0$ with initial conditions $y_0 = 0$, $y_1 = 1$. Forward recurrence yields $y_2 = 5$, $y_3 = 19$, $y_4 = 65, \ldots$.

The characteristic equation $u^2 - 5u + 6 = 0$ has roots $u_1 = 3$, $u_2 = 2$. Hence, the general solution is $y_n = c_1 3^n + c_2 2^n$. The initial conditions give the system of equations

$$c_1 + c_2 = 0, \qquad 3c_1 + 2c_2 = 1,$$

with solution $c_1 = 1$, $c_2 = -1$, hence $y_n = 3^n - 2^n$.

As a check we find $y_2 = 5,\ y_3 = 19$ in agreement with the results found by using forward recurrence.

Example 3.3.11.

Consider the difference equation

$$T_{n+1}(x) - 2xT_n(x) + T_{n-1}(x) = 0, \quad n \geq 1, \quad -1 < x < 1,$$

with initial conditions $T_0(x) = 1$, $T_1(x) = x$. We obtain $T_2(x) = 2x^2 - 1$, $T_3(x) = 4x^3 - 3x$, $T_4(x) = 8x^4 - 8x^2 + 1$, ... By induction, $T_n(x)$ is an nth degree polynomial in x.

We can obtain a simple formula for $T_n(x)$ by solving the difference equation. The characteristic equation is $u^2 - 2xu + 1 = 0$, with roots $u = x \pm i\sqrt{1-x^2}$. Set $x = \cos\phi$, $0 < x < \pi$. Then $u = \cos\phi \pm i\sin\phi$, and thus

$$u_1 = e^{i\phi}, \qquad u_2 = e^{-i\phi}, \quad u_1 \neq u_2.$$

The general solution is $T_n(x) = c_1 e^{in\phi} + c_2 e^{-in\phi}$, and the initial conditions give

$$c_1 + c_2 = 1, \qquad c_1 e^{i\phi} + c_2 e^{-i\phi} = \cos\phi,$$

with solution $c_1 = c_2 = 1/2$. Hence, $T_n(x) = \cos(n\phi)$, $x = \cos\phi$.

These polynomials are thus identical to the important Chebyshev polynomials that were introduced in (3.2.19), and were there in fact denoted by $T_n(x)$.

We excluded the cases $x = 1$ and $x = -1$, i.e. $\phi = 0$ and $\phi = \pi$, respectively. For the particular initial values of this example, there are no difficulties; the solution $T_n(x) = \cos n\phi$ depends continuously on ϕ, and as $\phi \to 0$ or $phi \to \pi$, $T_n(x) = \cos n\phi$ converges to 1 $\forall n$ or $(-1)^n$ $\forall n$, respectively.

When we ask for the general solution of the difference equation, the matters are a little more complicated, because the characteristic equation has in these cases a double root; $u = 1$ for $x = 1$, $u = -1$ for $x = -1$. Although they are thus covered by the next theorem, we shall look at them directly, because they are easy to solve, and they give a good preparation for the general case.

If $x = 1$, the difference equation reads $T_{n+1} - 2T_n + T_{n-1} = 0$, i.e. $\Delta^2 T_n = 0$. We know from before (see, e.g., Theorem 3.3.4) that this is satisfied iff $T_n = an + b$. The solution is no longer built up by exponentials; a linear term is there too.

If $x = -1$, the difference equation reads $T_{n+1} + 2T_n + T_{n-1} = 0$. Set $T_n = (-1)^n V_n$. The difference equation becomes, after division by $(-1)^{n+1}$, $V_{n+1} - 2V_n + V_{n-1} = 0$, with the general solution, $V_n = an + b$, hence $T_n = (-1)^n(an + b)$.

Theorem 3.3.12.

When u_i is an m_i-fold root of the characteristic equation, then the difference equation (12.3.3) *is satisfied by the sequence $\{y_n\}$, where*

$$y_n = P_i(n)u_i^n,$$

and P_i is an arbitrary polynomial in $\mathcal{P}_{m_i}$. The general solution of the difference equation is a linear combination of solutions of this form using all the distinct roots of the characteristic equation.

Proof. We can write the polynomial $P \in \mathcal{P}_{m_i}$ in the form

$$P_i(n) = b_1 + b_2 n + b_3 n(n-1) + \cdots + b_{m_i} n(n-1)\cdots(n-m_i+2).$$

Thus it is sufficient to show that equation (3.3.54) is satisfied when

$$y_n = n(n-1)\cdots(n-p+1)u_i^n = (u^p \partial^p (u^n)/\partial u^p)_{u=u_i}, \qquad p = 1, 2, \ldots, m_i - 1. \tag{3.3.57}$$

Substitute this in the left-hand side of equation (3.3.54):

$$\begin{aligned} u^p \frac{\partial^p}{\partial u^p}\Big(u^{n+k} + a_1 u^{n+k-1} + \cdots + a_k u^n\Big) &= u^p \frac{\partial^p}{\partial u^p}\big(\phi(u)u^n\big) \\ = u^p\Big(\phi^{(p)}(u)u^n + \binom{p}{1}\phi^{(p-1)}(u) n u^{n-1} + \cdots + \binom{p}{p}\phi(u)\frac{\partial^p}{\partial u^p}(u^n)\Big). \end{aligned}$$

The last manipulation was made using Leibniz's rule.

Now ϕ and all the derivatives of ϕ which occur in the above expression are 0 for $u = u_i$, since u_i is an m_i-fold root. Thus the sequences $\{y_n\}$ in equation (3.3.57) satisfy the difference equation. We obtain a solution with $\sum m_i = k$ parameters by the linear combination of such solutions derived from the different roots of the characteristic equation.

It can be shown (see, e.g., Henrici [23, p. 214]) that these solutions are linearly independent. (This also follows from a different proof given in Chapter. 13, where a difference equation of higher order is transformed to a system of first order difference equations. This transformation also leads to other ways of handling inhomogeneous difference equations than those which are presented in this section.) □

Note that the double root cases discussed in the previous example are completely in accordance with this theorem. We take one more example.

Example 3.3.12.

Consider the difference equation $y_{n+3} - 3y_{n+2} + 4y_n = 0$. The characteristic equation is $u^3 - 3u^2 + 4 = 0$ with roots $u_1 = -1$, $u_2 = u_3 = 2$. Hence, the general solution reads

$$y_n = c_1(-1)^n + (c_2 + c_3 n)2^n.$$

For a **nonhomogeneous** linear difference equation of order k, one can often find a *particular solution* by the use of an *"Ansatz"* with undetermined coefficients; thereafter, by Lemma 3.3.10 one can get the general solution by adding the general solution of the homogeneous difference equation.

Example 3.3.13.

Consider the difference equation $y_{n+1} - 2y_n = a^n$, with initial condition $y_0 = 1$. Try the "Ansatz" $y_n = ca^n$. One gets $ca^{n+1} - 2ca^n = a^n$, $c = 1/(a-2)$, $a \neq 2$. Thus the general solution is $y_n = a^n/(a-2) + c_1 2^n$. By the initial condition, $c_1 = 1 - 1/(a-2)$, hence

$$y_n = \frac{a^n - 2^n}{a-2} + 2^n. \tag{3.3.58}$$

When $a \to 2$, l'Hospital's rule gives $y_n = 2^n + n2^{n-1}$. Notice how the "Ansatz" must be modified when a is a root of the characteristic equation.

The general rule when the right hand side is of the form $P(n)a^n$ (or a sum of such terms), where P is a polynomial, is that the contribution of this term to y_n is $Q(n)a^n$, where Q is a polynomial. If a does not satisfy the characteristic equation then $\deg Q = \deg P$; if a is a single or a double root of the characteristic equation, then $\deg Q = \deg P + 1$ or $\deg Q = \deg P + 2$, respectively, etc. The coefficients of Q are determined by the insertion of $y_n = Q(n)a^n$ on the left hand side of the equation and matching the coefficients with the right hand side.

Another way to find a particular solution is based on the calculus of operators. Suppose that an inhomogeneous difference equation is given in the form $\psi(Q)y_n = b_n$, where Q is one of the operators Δ, δ and ∇, or an operator easily derived from these, e.g., $\frac{1}{6}\delta^2$, see Problem 27.

In § 3.2.2 $\psi(Q)^{-1}$ was defined by the formal power series with the same coefficients as the Maclaurin series for the function $1/\psi(z)$, $z \in \mathbf{C}$, $\psi(0) \neq 0$. In simple cases, e.g., if $\psi(Q) = a_0 + a_1 Q$, these coefficients are easily found. Example 3.1.6 indicates how to find the coefficients in more general cases. Then $\psi(Q)^{-1}b_n$ *is a particular solution* of the difference equation $\psi(Q)y_n = b_n$; the truncated expansions approximate this. Note that if $Q = \delta$ or ∇, the infinite expansion demands that b_n is defined also if $n < 0$.

Note that a similar technique, with the operator D, can also be applied to linear differential equations. Today this technique has to a large extent been replaced by the Laplace transform, that yields essentially the same algebraic calculations as operator calculus.

In some branches of applied mathematics it is popular to treat nonhomogeneous difference equations by means of a **generating function**, also called the **z-transform**, since both the definition and the practical computations are analogous to the Laplace transform. The z-transform of the sequence $y = \{y_n\}_0^\infty$ is

$$Y(z) = \sum_{n=0}^{\infty} y_n z^{-n}. \tag{3.3.59}$$

Note that the sequence $\{Ey\} = \{y_{n+1}\}$ has the z-transform $zY(z) - y_0$, $\{E^2 y\} = \{y_{n+2}\}$ has the z-transform $z^2 Y(z) - y_0 z - y_1$, etc.

If $Y(z)$ is available in *analytic* form, it can often be brought to a sum of functions, whose inverse z-transforms are known, by means of various analytic techniques, notably expansion into partial fractions, e.g., if $Y(z)$ is a rational function. On the other hand, if *numerical values* of $Y(z)$ have been computed for complex values of z on some circle in $\mathbf{C}$ by means of an algorithm, then y_n can be determined by an obvious modification of the Cauchy–FFT method described in Sec. 3.1.3 (for expansions into negative powers of z). More information about the z-transform can be found in Strang [44, Sec. 6.3].

We are now in a position to exemplify in more detail the use of linear difference equations to studies of numerical stability, of the type mentioned above.

Theorem 3.3.13.

Necessary and sufficient for boundedness (stability) of all solutions of the difference equation (3.3.54) *for all positive* n *is the following* **root condition***: (We shall say either that a difference equation or that a characteristic polynomial satisfies the root condition; the meaning is the same.)*

i. *All roots of characteristic equation* (3.3.55) *should be located inside or on the unit circle* $|z| \leq 1$*;*

ii. *The roots on the unit circle should be simple.*

Proof. Follows directly from Theorem 3.3.12. □

This root condition corresponds to cases, where it is the absolute error that matters. It is basic in the theory of linear multistep methods for ordinary differential equations. Computer Graphics and an algebraic criterion due to Schur are useful for investigations of the root condition in particular if the recurrence relation under investigation contains parameters.

There are important applications of single linear difference equations to the study of the stability of numerical methods. When a recurrence is used one is usually interested in the solution for one particular initial condition, but a rounding error in an initial value produces a different solution, and it is therefore of interest to know the behaviour of other solutions of the corresponding homogeneous difference equation. We have seen this already in Sec. 1.3.3 for an inhomogeneous first order recurrence relation, but it is even more important for recurrence relations of higher order.

The following example is based on a study by J. Todd[45] in (1950) (see [46]).

Example 3.3.14.

Consider the initial-value problem

$$y''(x) = -y, \quad y(0) = 0, \quad y'(0) = 1, \tag{3.3.60}$$

with the exact solution $y(x) = \sin x$. To compute an approximate solution $y_k = y(x_k)$ at equidistant points $x_k = kh$, where h is a step length, we approximate the second derivative according to (3.3.45),

$$y_k'' = h^{-2}\left(\delta^2 y_k + \frac{\delta^4 y_k}{12} + \frac{\delta^6 y_k}{90} + \dots\right). \tag{3.3.61}$$

We first use the first term only; the second term shows that the truncation error of this approximation of y_k'' is asymptotically $h^2 y^{(4)}/12$. We then obtain the difference equation $h^{-2}\delta^2 y_k = -y_k$ or, in other words,

$$y_{k+2} = (2 - h^2)y_{k+1} - y_k, \quad y_0 = 0, \tag{3.3.62}$$

[45] John Todd, Irish-American numerical analyst that was one of the first studies of the numerical stability of an algorithm for the approximate solution of ordinary differential equations.

where a suitable value of y_1 is to be assigned. In the third column of Table 3.4.2 we show the results obtained using this recursion formula with $h = 0.1$ and $y_1 = \sin 0.1$. We obtain about 3 digits accuracy at the end, $x = 1.5$.

Table 3.3.2. *Integrating $y'' = -y$, $y(0) = 0$, $y'(0) = 1$; the letters U and S in the headlines of the last two columns refer to "Unstable" and "Stable'.*

x_k	$\sin x_k$	2nd order	4th order U	4th order S
0.1	0.0998334166	0.0998334	0.0998334166	0.0998334166
0.2	0.1986693308	0.1986685	0.1986693307	0.1986693303
0.3	0.2955202067	0.2955169	0.2955202067	0.2955202050
0.4	0.3894183423	0.3894101	0.3894183688	0.3894183382
0.5	0.4794255386	0.4794093	0.4794126947	0.4794255305
0.6	0.5646424734	0.5646143	0.5643841035	0.5646424593
0.7	0.6442176872	0.6441732	0.6403394433	0.6442176650
0.8	0.7173560909	0.7172903	0.6627719932	0.7173560580
0.9	0.7833269096	0.7832346	0.0254286676	0.7833268635
1.0	0.8414709848	0.8413465	−9.654611899	0.8414709226
1.1	0.8912073601	0.8910450	−144.4011267	0.8912072789
1.2	0.9320390860	0.9318329	−2010.123761	0.9320389830
1.3	0.9635581854	0.9633026	−27834.59620	0.9635580577
1.4	0.9854497300	0.9851393	−385277.6258	0.9854495749
1.5	0.9974949866	0.9971245	−5332730.260	0.9974948015

Since the algorithm was based on a second order accurate approximation of y'' one may expect that the solution of the differential equation is also second order accurate. This turns out to be correct in this case, e.g., if we divide the step size by 2, the errors will be divided by 4, approximately. We shall, however; see that we cannot always draw conclusions of this kind; we also have to take the numerical stability into account.

In the hope to obtain a more accurate solution, we shall now use one more term in the expansion (3.3.61); the third term then shows that the truncation error of this approximation is asymptotically $h^4 y^{(6)}/90$. The difference equation now reads

$$\delta^2 y_k - \frac{1}{12}\delta^4 y_k = -h^2 y_k \tag{3.3.63}$$

or, in other words,

$$y_{k+2} = 16y_{k+1} - (30 - 12h^2)y_k + 16y_{k-1} - y_{k-2}, \quad k \geq 2, \quad y_0 = 0, \tag{3.3.64}$$

where starting values for y_1, y_2, and y_3 need to be assigned. You see them in Table 3.4.2, where the results from this algorithm are shown in the fourth column. We see that disaster has struck—the recursion is severely unstable! Already for $x = 0.6$ the results are less accurate than the second order scheme. For $x \geq 0.9$ the errors dominate completely.

We shall now look at these difference equations from the point of view of the root condition. The characteristic equation for (3.3.62) reads $u^2-(2-h^2)u+1=0$, and since $|2-h^2|<2$, direct computation shows that it has simple roots of unit modulus. The root condition is satisfied. By Example 3.3.11, the solution of (3.3.62) is $y_n = T_n(1-h^2/2)$.

For (3.3.64) the characteristic equation reads $u^4-16u^3+(30-12h^2)u^2-16u+1=0$. We see immediately that *the root condition cannot be satisfied*. Since the sum of the roots equals 16, it is impossible that all roots are inside or on the unit circle. In fact, the largest root equals 13.94. So, a tiny error at $x=0.1$ has been multiplied by $13.94^{14}\approx 10^{16}$ at the end.

It is easy to construct a stable fourth order accurate method. Just replace the term $\delta^4 y_k$ in (3.3.63) by $h^2\delta^2 y_k'' = -h^2\delta^2 y_k$. This leads to the recursion formula

$$y_{k+1} = \left(2 - \frac{h^2}{1+h^2/12}\right) y_k - y_{k-1}, \quad y_0 = 0. \tag{3.3.65}$$

This difference equation satisfies the root condition if $h^2<6$ (Problem 20(b)). This algorithm can be generalized to differential equations of the form $y''=f(x,y)$. It is known under several names, e.g., Numerov's method.[46] It requires y_0, $y_1\approx y(h)$ as seed. $y(h)$ can be obtained, for a small value of h, from a few terms of the Taylor expansion of the solution, the coefficients of which can be computed in the style of Example 3.1.1; see also Sec. 3.4 (Problem 28).

In the fifth column of Table 3.4.2 we show the results obtained using this recursion formula with $h=0.1$ and $y_1=\sin 0.1$. The error at the end is about $2\cdot 10^{-7}$.

Remark 3.3.2. If the solution of the original problem is itself strongly decreasing or strongly increasing, one should consider the location of the characteristic roots with respect to a circle in the complex plane that corresponds to the interesting solution. For example, if the interesting root is 0.8 then a root equal to -0.9 causes oscillations that may eventually become disturbing, if one is interested in *relative* accuracy also in a long run, even if the oscillating solution is small in the beginning.

Many problems contain homogeneous or nonhomogeneous linear difference equations with variable coefficients, for which the solutions are not known in a simple closed form.

We now confine the discussion to the cases where the original problems are to compute a particular solution of a *second order difference equation with variable coefficients*; several interesting problems of this type were mentioned above, and we formulated the questions: *can* we use a recurrence to find the wanted solution accurately, and *how* shall we use a recurrence, forwards or backwards. Typically the original problem contains some parameter, and one usually wants to make a study for an interval of parameter values.

Such questions are sometimes studied with *frozen coefficients*, i.e. the model problems are in the class of difference equations with constant coefficients in the

[46] The method can be traced back at least to B. Numerov 1924.

range of the actual coefficients of the original problem; if one of the types of recurrence is satisfactory (i.e. numerically stable in some sense) for all model problems, one would like to conclude that they are satisfactory also for the original problem, but *the conclusion is not always valid* without further restrictions on the coefficients—see a counterexample in Problem 26c.

The technique with *frozen coefficients provides just a hint* that should always be *checked by numerical experiments* on the original problem. It is beyond the scope of this text to discuss what restrictions are needed. *If the coefficients of the original problem are slowly varying, however, there is a good chance that the numerical tests will confirm the hint*—but again: how slowly is "slowly"? A warning against the use of one of the types of recurrence may also be a valuable result of a study, although it is negative.

The following lemma exemplifies a type of tool that may be useful in such cases. The proof is left for Problem 25a. Another useful tool is presented in Problem 26a and applied in Problem 26b.

Lemma 3.3.14. *Suppose that the wanted sequence y_n^* satisfies a difference equation (with constant coefficients),*

$$\alpha y_{n+1} + \beta y_n - \gamma y_{n-1} = 0, \quad (\alpha > \gamma > 0,\ \beta > 0),$$

and that y_n^* is known to be positive for all sufficiently large n. Then the characteristic roots can be written $0 < u_1 < 1$, $u_2 < 0$ and $|u_2| > u_1$. Then y_n^* is unique apart from a positive factor c; $y_n^* = cu_1^n$, $c > 0$.

A solution $\bar{y}_n$, called the trial solution *that is approximately of this form can be computed for $n = N : -1 : 0$ by* backward *recurrence starting with the "seed" $y_{N+1} = 0$ $y_N = 1$. If an accurate value of y_0^* is given, the wanted solution is*

$$y_n^* = \bar{y}_n y_0^* / \bar{y}_0,$$

with a relative error approximately proportional to $(u_2/u_1)^{n-N}$. (neglecting a possible error in y_0^*).[47]

The *forward* recurrence is not recommended for finding y_n^* in this case, since the positive term $c_1 u_1^n$ will eventually be drowned by the oscillating term $c_2 u_2^n$ that will be introduced by the rounding errors. The proof is left for Problem 26c. Even if y_0 (in the use of the forward recurrence) has no rounding errors, such errors committed at later stages will yield similar contributions to the numerical results.

Example 3.3.15.

The "original problem" is to compute the parabolic cylinder function $U(a, x)$ which satisfies the difference equation

$$(a + \tfrac{1}{2})U(a + 1, x) + xU(a, x) - U(a - 1, x) = 0,$$

see Handbook of mathematical functions [1, Ch. 19]; see in particular Example 19.28.1.

[47] If y_n^* is defined by some other condition, one can proceed analogously.

To be more precise, we consider the case $x = 5$. Given $U(3,5) = 5.2847\,10^{-6}$ (obtained from a table in Handbook, p. 710), we want to determine $U(a,5)$ for integer values of a, $a > 3$, as long as $|U(a,5)| > 10^{-15}$. We guess (a priori) that the discussion can be restricted to the interval (say) $a = [3,15]$. The above lemma then gives the hint of a backward recurrence, for $a = a' - 1 : -1 : 3$ for some appropriate a' (see below), in order to obtain a trial solution $\bar{U}_a$ with the seed $\bar{U}_{a'} = 1$, $\bar{U}_{a'+1} = 0$. Then the wanted solution becomes, by the Lemma, (with changed notation),

$$U(a,5) = \bar{U}_a U(3,5)/\bar{U}_3.$$

The positive characteristic root of the frozen difference equation varies from 0.174 to 0.14 for $a = 5 : 15$; while the modulus of the negative root is between 6.4 and 3.3 times as large. This motivates a choice of $a' \approx 4 + (-9 - \log 5.3)/\ln 0.174 \approx 17$ for the backward recursion; it seems advisable to choose a' (say) 4 units larger than the value where U becomes negligible.
Forward recurrence with correctly rounded starting values $U(3,5) = 5.2847\,10^{-6}$, $U(4,5) = 9.172\,10^{-7}$, gives oscillating (absolute) errors of relatively slowly decreasing amplitude, approximately 10^{-11}, that gradually drowns the exponentially decreasing true solution; the estimate of $U(a,5)$ itself became negative for $a = 10$, and then the results oscillated with approximate amplitude 10^{-11}, while the correct results decrease from the order of 10^{-11} to 10^{-15} as $a = 10 : 15$. The details are left for Problem 25b.

It is conceivable that this procedure can be used for all x in some interval around 5, but we refrain from presenting the properties of the parabolic cylinder function needed for determining the interval.

If the problem is nonlinear, one can instead solve the original problem with two seeds, (say) y'_N, y''_N, and study how the results deviate. The seeds should be so close that a linearization like $f(y'_n) - f(y''_n) \approx r_n(y'_n - y''_n)$ is acceptable, but $y'_n - y''_n$ should be well above the rounding error level. A more recent and general treatment of these matters is found in [17, Chapter 6].

Review Questions

1. Give expressions for the shift operator E^k in terms of Δ, ∇, and hD, and expressions for the central difference operator δ^2 in terms of E and hD.
2. Derive the best upper bound for the error of $\Delta^n y_0$, if we only know that the absolute value of the error of y_i, $i = 0, \ldots, n$ does not exceed ϵ.
3. There is a theorem (and a corollary) about existence and uniqueness of approximation formulas of a certain type that are exact for polynomials of certain class. Formulate these results, and sketch the proofs.
4. What bound can be given for the k'th difference of a function in terms of a bound for the k'th derivative of the same function?

5. Formulate the basic theorem concerning the use of operator expansions for deriving approximation formulas for linear operators.

6. Formulate Peano's Remainder Theorem, and compute the Peano kernel for a given symmetric functional (with at most four subintervals).

7. Express polynomial interpolation formulas in terms of forward and backward difference operators.

8. Give Stirling's interpolation formula for quadratic interpolation with approximate bounds for truncation error and irregular error.

9. Derive central difference formulas for $f'(x_0)$ and $f''(x_0)$ that are exact for $f \in \mathcal{P}_4$. They should only use function values at x_j, $j = 0, \pm 1, \pm 2, \ldots$, as many as needed. Give asymptotic error estimates.

10. Derive the formula for the general solution of the difference equation $y_{n+k} + a_1 y_{n+k-1} + \ldots + a_k y_n = 0$, when the characteristic equation has simple roots only. What is the general solution, when the characteristic equation has multiple roots?

11. What is the general solution of the difference equation $\Delta^k y_n = an + b$?

12. Prove Lemma 3.3.14, and present the main features of its application to the parabolic cylinder function.

Problems and Computer Exercises

1. (a) Show that

$$(1+\Delta)(1-\nabla) = 1, \qquad \Delta - \nabla = \Delta\nabla = \delta^2 = E - 2 + E^{-1},$$
$$\delta^2 y_n = y_{n+1} - 2y_n + y_{n-1}.$$

(b) Let $\Delta^p y_n, \nabla^p y_m, \delta^p y_k$ all denote the same quantity. How are n, m, k connected? Along which lines in the difference scheme are the subscripts constant?

(c) Given the values of $y_n,\ \nabla y_n, \ldots,\ \nabla^k y_n$, for a particular value of n. Find a recurrence relation for computing $y_n,\ y_{n-1}, \ldots,\ y_{n-k}$, by simple additions only. On the way you obtain the full difference scheme of this sequence.

(d) *Repeated summation by parts.* Show that if $u_1 = u_N = v_1 = v_N = 0$, then

$$\sum_{n=1}^{N-1} u_n \Delta^2 v_{n-1} = -\sum_{n=1}^{N-1} \Delta u_n \Delta v_n = \sum_{n=1}^{N-1} v_n \Delta^2 u_{n-1}.$$

(e) Show that if $\Delta^k v_n \to 0$, as $n \to \infty$, then $\sum_{n=m}^{\infty} \Delta^k v_n = -\Delta^{k-1} v_m$.

(f) Show that $(\mu\delta^3 + 2\mu\delta) f_0 = f_2 - f_{-2}$

(g) Prove, e.g., by means of summation by parts, that $\sum_{n=0}^{\infty} u_n z^n,\ |z| = 1,\ z \neq 1$, is convergent if $u_n \to 0$ monotonically. Formulate similar results for real cosine and sine series.

2. (a) Prove, e.g., by induction, the following two formulas:

$$\Delta_x^j \binom{x}{k} = \binom{x}{k-j}, \quad j \le k,$$

where Δ_x means differencing with respect to x, with $h = 1$.

$$\Delta^j x^{-1} = \frac{(-h)^j j!}{x(x+h)\cdots(x+jh)}.$$

Find the analogous expression for $\nabla^j x^{-1}$.

(b) What formulas with derivatives instead of differences are these formulas analogous to?

(c) Show the following formulas, if $x,\ a$ are integers:

$$\sum_{n=a}^{x-1} \binom{n}{k-1} = \binom{x}{k} - \binom{a}{k},$$

$$\sum_{n=x}^{\infty} \frac{1}{n(n+1)\cdots(n+j)} = \frac{1}{j} \cdot \frac{1}{x(x+1)\cdots(x+j-1)}.$$

Modify these results for non-integer x; $x - a$ is still an integer.

(d) Suppose that $b \neq 0, -1, -2, \ldots$, and set

$$c_0(a,b) = 1, \quad c_n(a,b) = \frac{a(a+1)\ldots(a+n-1)}{b(b+1)\ldots(b+n-1)}, \quad n = 1, 2, 3, \ldots$$

Show, e.g., by induction that $(-\Delta)^k c_n(a,b) = c_k(b-a,b)c_n(a,b+k)$, hence $(-\Delta)^n c_0(a,b) = c_n(b-a,b)$.

(e) Compute for $a = e$, $b = \pi$ (say), $c_n(a,b)$, $n = 1:100$. How do you avoid overflow? Compute $\Delta^n c_0(a,b)$, both numerically by the difference scheme, and according to the formula in (d). Compare the results and formulate your experiences. Do the same with $a = e$, $b = \pi^2$.

Do the same with $\Delta^j x^{-1}$ for various values of $x,\ j$ and h.

3. Set

$$\mathrm{Y}_{ord} = (y_{n-k}, y_{n-k+1}, \ldots, y_{n-1}, y_n),$$
$$\mathrm{Y}_{dif} = (\nabla^k y_n,\ \nabla^{k-1} y_n, \ldots, \nabla y_n,\ y_n).$$

Note that the results of this problem also hold if the y_j are column vectors.

(a) Find a matrix P, such that $\mathrm{Y}_{dif} = \mathrm{Y}_{ord} P$. Show that

$$\mathrm{Y}_{ord} = \mathrm{Y}_{dif} P \quad \text{hence} \quad P^{-1} = P.$$

How do you generate this matrix by means of a simple recurrence relation?

Hint: P is related to the Pascal matrix, but do not forget the minus signs in

this triangular matrix. Compare Problem 3 of Sec. 1.3.

(b) Suppose that $\sum_{j=0}^{k} \alpha_j E^{-j}$ and $\sum_{j=0}^{k} a_j \nabla^j$ represent the same operator. Set $\alpha = (\alpha_k, \alpha_{k-1}, \ldots, \alpha_0)^T$, and $a = (a_k, a_{k-1}, \ldots, a_0)^T$, i.e. $\mathrm{Y}_{ord} \cdot \alpha \equiv \mathrm{Y}_{dif} \cdot a$. Show that $Pa = \alpha$, $P\alpha = a$.

(c) The matrix P depends on the integer k. Is it true that the matrix which is obtained for a certain k is a submatrix of the matrix you obtain for a larger value of k?

(d) Compare this method of performing the mapping $\mathrm{Y}_{ord} \mapsto \mathrm{Y}_{dif}$ with the ordinary construction of a difference scheme. Consider the number of arithmetic operations, the kind of arithmetic operations, rounding errors, convenience of programming in a language with matrix operations as primary operations etc. Compare in the same way this method of performing the inverse mapping with the algorithm in Problem 1c.

4. (a) Set $f(x) = \tan x$. Compute by the use of the table of $\tan x$ (in Example 3.3.2), and the interpolation and differentiation formulas given in the above examples (almost) as accurately as possible

$$f'(1.35),\ f(1.322),\ f'(1.325),\ f''(1.32).$$

Estimate the influence of rounding errors of the function values and estimate the truncation errors.

(b) Write a program for computing a difference scheme. Use it for computing the difference scheme for more accurate values of $\tan x$, $x = 1.30 : 0.01 : 1.35$, and calculate improved values of the functionals in (a). Compare the error estimates with the true errors.

(c) Verify the assumptions of Theorem 3.3.7 for one of the three interpolation formulas in Sec. 3.3.4.

(d) It is rather easy to find the values at $\theta = 0$ of the first two derivatives of Stirling's interpolation formula. You find thus explicit expressions for the coefficients in the formulas for $f'(x_0)$ and $f''(x_0)$ in (3.3.49) and (3.3.45), respectively. Check numerically a few coefficients in these equations, and explain why they are reciprocals of integers. Also note that each coefficient in (3.3.49) has a simple relation to the corresponding coefficient in (3.3.45).

5. (a) Study Bickley's table (Table 3.2.1), and derive some of the formulas, in particular the expressions for δ and μ in terms of hD, and vice versa.

(b) Show that $h^{-k}\delta^k - D^k$ has an expansion into *even* powers of h, when k is even. Find an analogous result for $h^{-k}\mu\delta^k - D^k$ when k is odd.

6. (a) Compute

$$f'(10)/12,\quad f^{(3)}(10)/720,\quad f^5(10)/30240,$$

by means of (3.3.26), given values of $f(x)$ for integer values of x. (This is asked for, e.g., in applications of Euler–Maclaurin's formula, Sec. 3.4.4.) Do this for $f(x) = x^{-3/2}$. Compare with the correct derivatives. Then do the same also for $f(x) = (x^3 + 1)^{-1/2}$.

(b) Study the backwards differentiation formula, see Example 3.3.6, on a com-

puter. Compute $f'(1)$ for $f(x) = 1/x$, for $h = 0.02$ and $h = 0.03$, and compare with the exact result. Make a semi-logarithmic plot of the total error after n terms, $n = 1 : 29$. Study also the sign of the error. For each case, try to find out whether the achievable accuracy is set by the rounding errors or by the semi-convergence of the series.

Hint: A formula mentioned in Problem 2(a) can be helpful. Also note that this problem is both similar and very different from the function $\tan(x)$ that was studied in Example 3.3.6.

(c) Set $x_i = x_0 + ih$, $t = (x - x_2)/h$. Show that

$$y(x) = y_2 + t\Delta y_2 + \frac{t(t-1)}{2}\Delta^2 y_2 + \frac{t(t-1)(t-2)}{6}\Delta^3 y_1$$

equals the interpolation polynomial in $\mathcal{P}_4$ determined by the values (x_i, y_i), $i = 1 : 4$. (Note that $\Delta^3 y_1$ is used instead of $\Delta^3 y_2$ which is located outside the scheme. Is this OK?)

7. (a) Show the validity of the algorithm in (3.3.42).

(b) A well known formula reads

$$P(D)(e^{\alpha t}u(t)) = e^{\alpha t}P(D+\alpha)u(t),$$

where P is an arbitrary polynomial. Prove this, as well as the following analogous formulas:

$$P(E)(a^n u_n) = a^n P(aE)u_n,$$
$$P(\Delta/h)\big((1+\alpha h)^n u_n\big) = (1+\alpha h)^n P((1+\alpha h)\Delta/h + \alpha)u_n.$$

Can you find a more beautiful or more practical variant?

8. Find the Peano kernel $K(u)$ for the functional $\Delta^2 f(x_0)$. Compute $\int_{\mathbf{R}} K(u)\,du$ both by direct integration of $K(u)$, and by computing $\Delta^2 f(x_0)$ for a suitably chosen function f.

9. Set $y_j = y(t_j)$, $y'_j = y'(t_j)$. The following relations are of great interest in the numerical integration of the differential equations $y' = f(y)$:

(a) The **implicit Adams formula**:

$$y_{n+1} - y_n = h\left(a_0 y'_{n+1} + a_1 \nabla y'_{n+1} + a_2 \nabla^2 y'_{n+1} + \cdots\right).$$

Show that $\nabla = -\ln(1-\nabla)\sum a_i \nabla^i$, and find a recurrence relation for the coefficients. The coefficients a_i, $i = 0 : 6$, read as follows. Check a few of them.

$$a_i = 1, \quad -\frac{1}{2}, \quad -\frac{1}{12}, \quad -\frac{1}{24}, \quad -\frac{19}{720}, \quad -\frac{3}{160}, \quad -\frac{863}{60480}.$$

Alternatively, derive the coefficients by means of the matrix representation, of a truncated power series.

(b) The **explicit Adams formula**:

$$y_{n+1} - y_n = h\left(b_0 y'_n + b_1 \nabla y'_n + b_2 \nabla^2 y'_n + \cdots\right).$$

Show that $\sum b_i \nabla^i E^{-1} = \sum a_i \nabla^i$, and show that

$$b_n - b_{n-1} = a_n, \quad (n \geq 1).$$

The coefficients $b_i,\ i = 0:6$, read as follows. Check a few of them.

$$b_i = 1,\ \frac{1}{2},\ \frac{5}{12},\ \frac{3}{8},\ \frac{251}{720},\ \frac{95}{288},\ \frac{19087}{60480}.$$

(c) Apply the the second order explicit Adams formula, i.e.

$$y_{n+1} - y_n = h(y'_n + \tfrac{1}{2}\nabla y'_n),$$

to the differential equation $y' = -y^2$, with initial condition $y(0) = 1$ and step size $h = 0.1$. Two initial values are needed for the recurrence; $y_0 = y(0) = 1$, of course, and we choose[48] $y_1 = 0.9090$. Then compute $y'_0 = -y_0^2$, $y'_1 = -y_1^2$. Then the explicit Adams formula yields y_2, and so on. Compute a few steps, and compare with the exact solution.[49]

10. Let $y_j = y_0 + jh$. Find the asymptotic behavior as $h \to 0$ of

$$(5(y_1 - y_0) + (y_2 - y_1))/(2h) - y'_0 - 2y'_1.$$

Comment: This is of interest in the analysis of cubic spline interpolation in Sec. 4.6.4.

11. It sometimes happens that the values of some function $f(x)$ can be computed by some very time-consuming algorithm only, and that one therefore computes it much sparser than is needed for the application of the results. It was common in the pre-computer age to compute sparse tables that needed interpolation by polynomials of a high degree; then one needed a simple procedure for **subtabulation**, i.e. to obtain a denser table for some section of the table. Today a similar situation may occur in connection with the graphical output of the results of (say) a numerical solution of a differential equation.
Define the operators ∇ and ∇_k by the equations

$$\nabla f(x) = f(x) - f(x-h), \quad \nabla_k f(x) = f(x) - f(x-kh),\ (k < 1),$$

and set

$$\nabla_k^r = \sum_{s=r}^{\infty} c_{rs}(k)\nabla^s.$$

(a) In order to compute the coefficients c_{rs}, $r \leq s \leq m$, you are advised to use a subroutine for finding the coefficients in the product of two polynomials, truncate the result, and apply the subroutine $m - 1$ times.

[48] There are several ways of obtaining $y_1 \approx y(h)$, e.g., by one step of Runge's 2nd order method, see Sec. 1.4.3, or by a series expansion, like in Example 3.1.2.

[49] For an *implicit* Adams formula it is necessary, in this example, to solve a quadratic equation in each step.

(b) Given

f_n	∇f_n	$\nabla^2 f_n$	$\nabla^3 f_n$	$\nabla^4 f_n$
1	0.181269	0.032858	0.005956	0.001080

Compute for $k = \frac{1}{2}$, $f_n = f(x_n)$, $\nabla_k^j f_n$ for $j = 1:4$. Compute $f(x_n - h)$ and $f(x_n - 2h)$, by means of both $\{\nabla^j f_n\}$ and $\{\nabla_k^j f_n\}$ and compare the results. How big difference of the results did you expect, and how big difference do you obtain?

12. Solve the following difference equations. A solution in complex form should be transformed to real form. As a check, compute (say) y_2 both by recurrence and by your closed form expression.

(a) $y_{n+2} - 2y_{n+1} - 3y_n = 0$, $y_0 = 0$, $y_1 = 1$;

(b) $y_{n+2} - 4y_{n+1} + 5y_n = 0$, $y_0 = 0$, $y_1 = 2$;

(c) There exist problems with two-point boundary conditions for difference equations, as for differential equations. $y_{n+2} - 2y_{n+1} - 3y_n = 0$, $y_0 = 0$, $y_{10} = 1$;

(d) $y_{n+2} + 2y_{n+1} + y_n = 0$, $y_0 = 1$, $y_1 = 0$;

(e) $y_{n+1} - y_n = 2^n$, $y_0 = 0$;

(f) $y_{n+2} - 2y_{n+1} - 3y_n = 1 + \cos\frac{\pi n}{3}$, $y_0 = y_1 = 0$;

Hint: The right hand side is $\Re(1 + a^n)$, where $a = e^{\pi i/3}$.

(g) $y_{n+1} - y_n = n$, $y_0 = 0$;

(h) $y_{n+1} - 2y_n = n2^n$, $y_0 = 0$;

13. (a) Prove Lemma 3.3.10.

(b) Consider the difference equation $y_{n+2} - 5y_{n+1} + 6y_n = 2n + 3(-1)^n$. Determine a particular solution of the form $y_n = an + b + c(-1)^n$.

(c) Solve also the difference equation $y_{n+2} - 6y_{n+1} + 5y_n = 2n + 3(-1)^n$. Why and how must you change the form of the particular solution?

14. (a) Show that the difference equation $\sum_{i=0}^{k} b_i \Delta^i y_n = 0$ has the characteristic equation: $\sum_{i=0}^{k} b_i (u-1)^i = 0$.

(b) Solve the difference equation $\Delta^2 y_n - 3\Delta y_n + 2y_n = 0$, with initial condition $\Delta y_0 = 1$.

(c) Find the characteristic equation for the equation $\sum_{i=0}^{k} b_i \nabla^i y_n = 0$?

15. The influence of wrong boundary slopes for cubic spline interpolation (with equidistant data)—see Sec. 4.6—is governed by the difference equation

$$e_{n+1} + 4e_n + e_{n-1} = 0, \quad 0 < n < m,$$

e_0, e_m given. Show that $e_n \approx u^n e_0 + u^{m-n} e_m$, $u = \sqrt{3} - 2 \approx -0.27$. More precisely

$$\left| e_n - (u^n e_0 + u^{m-n} e_m) \right| \leq \frac{2|u^{3m/2}| \max(|e_0|, |e_m|)}{1 - |u|^m}.$$

Generalize the simpler of these results to other difference and differential equations.

16. The Fibonacci sequence is defined by the recurrence relation

$$y_n = y_{n-1} + y_{n-2}, \quad y_0 = 0, \quad y_1 = 1.$$

(a) Calculate $\lim_{n\to\infty} y_{n+1}/y_n$.

(b) The error of the secant method (see Sec. 6.2.2) satisfies approximately the difference equation $\epsilon_n = C\epsilon_{n-1}\epsilon_{n-2}$. Solve this difference equation. Determine p, such that $\epsilon_{n+1}/\epsilon_n^p$ tends to a finite nonzero limit as $n \to \infty$. Calculate this limit.

17. For several algorithms using the "divide and conquer strategy", such as the Fast Fourier Transform and some sorting methods, one can find that the work $W(n)$ for the application of them to data of size n satisfies a recurrence relation of the form:

$$W(n) = 2W(n/2) + kn,$$

where k is a constant. Find $W(n)$.

18. When the recursion

$$x_{n+2} = (32x_{n+1} - 20x_n)/3, \quad x_0 = 3,\ x_1 = 2,$$

was solved numerically in low precision (23 bits mantissa), one obtained for x_i, $i = 2:12$ the (rounded) values

$$1.33,\ 0.89,\ 0.59,\ 0.40,\ 0.26,\ 0.18,\ 0.11,\ 0.03,\ -0.46,\ -5.05,\ -50.80.$$

Explain the difference from the exact values $x_n = 3(2/3)^n$.

19. (a) k, N are given integers $0 \le k <\le N$. A "discrete Green's function" $G_{n,k},\ 0 \le n \le N$ for the central difference operator $-\Delta\nabla$ together with the boundary conditions given below, is defined as the solution $u_n = G_{n,k}$ of the difference equation with boundary conditions, a

$$-\Delta\nabla u_n = \delta_{n,k}, \quad u_0 = u_N = 0;$$

($\delta_{n,k}$ is Kronecker's delta). Derive a fairly simple expression for $G_{n,k}$.

(b) Find (by computer) the inverse of the tridiagonal matrix

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix}.$$

What is the relation between Problems (a) and (b)? Find a formula for the elements of A^{-1}. Express the solution of the inhomogeneous difference equation $-\Delta\nabla u_n = b_n$, $u_0 = u_N = 0$, both in terms of the Green function $G_{n,k}$

and in terms of A^{-1} (for general N).

(c) Try to find an analogous formula[50] for the solution of an inhomogeneous boundary value problem for the *differential* equation $-u'' = f(x)$, $u(0) = u(1) = 0$.

20. (a) Demonstrate the formula

$$\sum_0^\infty \frac{(-x)^n c_n}{n!} = e^{-x} \sum_0^\infty \frac{x^n (-\Delta)^n c_0}{n!}. \tag{3.3.66}$$

Hint: Use the relation $e^{-xE} = e^{-x(1+\Delta)} = e^{-x} e^{-x\Delta}$.

(b) For completely monotonic sequences $\{c_n\}$ and $\{(-\Delta)^n c_0\}$ are typically positive and decreasing sequences. For such sequences, the left hand side becomes extremely ill-conditioned for large x, (say) $x = 100$, while the graph of the terms on the right hand side (if exactly computed) are bell-shaped, almost like the normal probability density with mean x and standard deviation $\sqrt{x}$. We have called such a sum a *bell sum*. Such positive sums can be computed with little effort and no trouble with rounding errors, *if their coefficients are accurate*.

Compute the left hand side of (3.3.66), for $c_n = 1/(n+1)$, $x = 10 : 10 : 100$, and compute the right hand side, both with numerically computed differences and with exact differences; the latter are found in Problem 2a. (In this particular case you can also find the exact sum.)

Suppose that the higher differences $\{(-\Delta)^n c_0\}$ have been computed recursively from rounded values of c_n. Explain why one may fear that the right hand side of (3.3.66) does not provide much better results than the left hand side.

(c) Use (3.3.66) to derive the second expansion for $\operatorname{erf}(x)$ in Problem 10 of Sec. 3.1 from the first expansion.

Hint: Use one of the results of Problem 2 a.

(d) If $c_n = c_n(a, b)$ is defined as in Problem 2d, then the left hand side becomes the Maclaurin expansion of the Kummer function $M(a, b, -x)$; see Abramowitz and Stegun [1, Ch. 13]; Show that

$$M(a, b, -x) = e^{-x} M(b - a, b, x)$$

by means of the results of Problems 23a and 2d.

21. (a) The difference equation $y_n + 5y_{n-1} = n^{-1}$ was discussed in Sec. 1.3.3. It can also be written thus: $(6 + \Delta)y_{n-1} = n^{-1}$. The expansion of $(6 + \Delta)^{-1} n^{-1}$ into powers of $\Delta/6$ provides a particular solution of the difference equation. Compute this numerically for a few values of n. Try to prove the convergence, with or without the expression in Problem 2b. Is this the same as the particular solution $I_n = \int_0^1 x^n (x + 5)^{-1} dx$ that was studied in Chapter1?

[50] In a *differential* equation, analogous to Problem 21(a), the Kronecker delta is to be replaced by the Dirac delta function. Also note that the inverse of the differential operator here can be described as an integral operator with the Green's function as the "kernel".

Hint: What happens as $n \to \infty$? Can more than one solution of this difference equation be bounded as $n \to \infty$?

(b) Make a similar study to the difference equation related to the integral in Problem 9 of Sec.1.3. Why does the argument suggested by the hint of (a) not work in this case? Try another proof.

22. (a) Prove Lemma 3.3.14. How is the conclusion to be changed, if we do not suppose that $\gamma < \alpha$, though the coefficients are still positive? Show that a backward recurrence is still to be recommended.

(b) Work out on a computer the numerical details of Example 3.3.15, and compare with Abramowitz and Stegun, Example 19.28.1. (Some deviations are to be expected, since Miller used other rounding rules.) Try to detect the oscillating component by computing the difference scheme of the the computed $U(a, 5)$, and estimate roughly the error of the computed values.

23. (a) For which constant real a does the difference equation $y_{n+1} - 2ay_n + y_{n-1} = 0$ satisfy the root condition?

For which values of the real constant a does there exist a solution, such that $\lim_{n\to\infty} y_n = 0$? For these values of a, how do you construct a solution $y_n = y_n^*$ by a recurrence and normalization, so that this condition as well as the condition $y_0^* + 2\sum_{m=1}^{\infty} y_{2m}^* = 1$ are satisfied. Is y_n^* unique? Give also an explicit expression for y_n^*.

For the other real values of a, show that y_n^* does not exist, but that for any given y_0, y_1 a solution can be accurately constructed by forward recurrence. Give an explicit expression for this solution in terms of Chebyshev polynomials (of the first and the second kind). Is it true that backward recurrence is also stable, though more complicated than forward recurrence?

(b) The Bessel function $J_k(z)$ satisfies the difference equation,

$$J_{k+1}(z) - (2k/z)J_k(z) + J_{k-1}(z) = 0, \quad k = 1,\ 2,\ 3, \ldots,$$

and the identities,

$$J_0(z) + 2J_2(z) + 2J_4(z) + 2J_6(z) + \ldots = 1;$$

$$J_0(z) - 2J_2(z) + 2J_4(z) - 2J_6(z) + \ldots = \cos z;$$

see Abramowitz and Stegun [1], 9.1.27, 9.1.46 and 9.1.47.

Show how one of the identities can be used for normalizing the trial sequence obtained by a backwards recurrence. Under what condition does Problem 26(a) give the hint to use the backwards recurrence for this difference equation?

Study the section on Bessel functions of integer order in Numerical Recipes. Apply this technique for $z = 10,\ 1,\ 0.1$ (say). The asymptotic formula (see [1, **9.3.1**])

$$J_k(z) \sim \frac{1}{\sqrt{2\pi k}}\Big(\frac{ez}{2k}\Big)^k, \quad k \gg 1,\ z \text{ fixed.}$$

may be useful for your decision where to start the backward recurrence. Use at least two starting points, and subtract the results (after normalization).

Comment: The above difference equation for $J_k(z)$ is also satisfied by a function denoted $Y_k(z)$,

$$Y_k(z) \sim \frac{-2}{\sqrt{2\pi k}}\Big(\frac{ez}{2k}\Big)^{-k}, \quad (k \gg 1).$$

How do these two solutions disturb each other, when forward or backward recurrence is used?

(c) *A counterexample* to the technique with frozen coefficients. Consider the difference equation $y_{n+1} - (-1)^n y_n + y_{n-1} = 0$. The technique with frozen coefficients leads to the consideration of the difference equations

$$z_{n+1} - 2az_n + z_{n-1} = 0, \quad a \in [-0.5, 0.5];$$

all of them have only bounded solutions. Find by numerical experiment that, nevertheless, there seems to exist unbounded solutions y_n of the first difference equation.

Comment: A theoretical proof of this is found by noting that the mapping $(y_{2n},\ y_{2n+1}) \mapsto (y_{2n+2},\ y_{2n+3})$ is represented by a matrix that is independent of n and has an eigenvalue that is less than -1.

24. Let $\{b_n\}_{-\infty}^{\infty}$ be a given sequence, and consider the difference equation,

$$y_{n-1} + 4y_n + y_{n+1} = b_n,$$

which can also be written in the form $(6 + \delta^2)y_n = b_n$.

(a) Show that the difference equation has at most one solution that is bounded for $-\infty < n < +\infty$. Find a particular solution in the form of an expansion into powers of the operator $\delta^2/6$. (This is hopefully bounded.)

(b) Apply it numerically to the sequence $b_n = (1+n^2h^2)^{-1}$, for a few values of the step size h, e.g., $h = 0.1, 0.2, 0.5, 1$. Study for $n = 0$ the rate of decrease (?) of the terms in the expansion. Terminate when you estimate that the error is (say) 10^{-6}. Check how well the difference equation is satisfied by the result.

(c) Study theoretically bounds for the terms when $b_n = \exp(i\omega hn)$ $\omega \in \mathbf{R}$. Does the expansion converge? Compare your conclusions with numerical experiments. Extend to the case when $b_n = B(nh)$, where $B(t)$ can be represented by an absolutely convergent Fourier integral, $B(t) = \int_{-\infty}^{\infty} e^{i\omega t}\beta(\omega)d\omega$. Note that $B(t) = (1+t^2)^{-1}$ if $\beta(\omega) = \frac{1}{2}e^{-|\omega|}$. Compare the theoretical results with the experimental results in (b).

(d) Put $Q = \delta^2/6$. Show that $\tilde{y}_n \equiv (1 - Q + Q^2 + \ldots \pm Q^{k-1})b_n/6$ satisfies the difference equation $(1 + Q)(\tilde{y}_n - y_n) = Q^k b_n/6$.

Comment: This procedure is worthwhile if the sequence b_n is so smooth that (say) 2 or 3 terms give satisfactory accuracy.

3.4 Acceleration of Convergence

3.4.1 Introduction

If a sequence $\{s_n\}_0^\infty$ converges slowly towards a limit s, but has a sort of regular behavior when n is large, it can under certain conditions be transformed into another infinite sequence $\{s'_n\}$, that converges much faster to the same limit. Here s'_n usually depends on the first n elements of the original sequence only. This is called **convergence acceleration**. Such a *sequence* transformation may be iterated, to yield a sequence of infinite sequences, $\{s''_n\}$, $\{s'''_n\}$ etc., hopefully with improved convergence towards the same limit s. For an *infinite series* convergence acceleration means the convergence acceleration of its sequence of partial sums. Some algorithms are most easily discussed in terms of sequences, others in terms of series.

Several transformations, linear as well as nonlinear, have been suggested and are successful, under various conditions. Some of them, like Aitken, repeated averages, and Euler's transformation, are most successful on *oscillating sequences* (alternating series or series in a complex variable). Others, like variants of Aitken acceleration, Euler–Maclaurin and Richardson, work primarily on *monotonic sequences* (series with positive terms). Some techniques for convergence acceleration transform a power series into a sequence of rational functions, e.g., continued fractions, Padé approximation, and the ϵ-algorithm

Convergence acceleration cannot be applied to "arbitrary sequences"; some sort of conditions are necessary that restrict the variation of the future elements of the sequence, i.e. the elements which are not computed numerically. In this section, these conditions are of a rather general type, in terms of *monotonicity, analyticity or asymptotic behavior* of simple and usual types. Nevertheless some of these techniques may even sometimes be successfully applied to *semi-convergent sequences*. Several of them can also use a limited number of coefficients of a power series for the computation of values of an *analytic continuation* of a function, outside the circle of convergence of the series that defined it.

There are also methods (due to Lindelöf, Plana and others) that transform an infinite series to an integral in the complex plane. They can, with appropriate numerical procedures for computing the integral, compete with the methods mentioned for the purposes mentioned, but they have the additional property to be applicable to some *ill-conditioned series*.

In addition to the "general purpose" techniques to be discussed in this chapter, there are other techniques of convergence acceleration based on the use of more specific knowledge about a problem. For example, Poisson summation formula

$$\sum_{n=-\infty}^{\infty} f(n) = \sum_{j=-\infty}^{\infty} \hat{f}(j), \quad \hat{f}(\omega) = \int_{-\infty}^{\infty} f(\omega) e^{-2\pi i \omega x}\, dx; \tag{3.4.1}$$

($\hat{f}$ is the Fourier transform of f). This can be amazingly successful to a certain class of series $\sum a(n)$, namely if $a(x)$ has a rapidly decreasing Fourier Transform. The Poisson formula is also an invaluable tool for the design and analysis of numerical methods for several problems; see Theorem 3.4.3.

Irregular errors are very disturbing when these techniques are used. They sometimes set the limit for the reachable accuracy. For the sake of simplicity we therefore use IEEE double precision, in most examples.

3.4.2 Comparison Series and Aitken Acceleration

Suppose that the terms in the series $\sum_{j=1}^{\infty} a_j$ behave, for large j, like the terms of a series $\sum_{j=1}^{\infty} b_j$, i.e. $\lim_{j\to\infty} a_j/b_j = 1$. Then if the sum $s = \sum_{j=1}^{\infty} b_j$ is known one can write

$$\sum_{j=1}^{\infty} a_j = s + \sum_{j=1}^{\infty}(a_j - b_j),$$

where the series on the right hand side converges more quickly than the given series. We call this making use of a simple **comparison problem**. The same idea is used in many other contexts—for example, in the computation of integrals where the integrand has a singularity. Usual comparison series are

$$\sum_{j=1}^{\infty} n^{-2} = \pi^2/6, \qquad \sum_{j=1}^{\infty} n^{-4} = \pi^4/90, \quad etc.$$

A general expression for $\sum_{j=1}^{\infty} n^{-2r}$, is given by (3.4.26). No simple closed form is known for $\sum_{j=1}^{\infty} n^{-3}$.

Example 3.4.1.

The term $a_j = (j^4+1)^{-1/2}$ behaves, for large j, like $b_j = j^{-2}$, whose sum is $\pi^2/6$. Thus

$$\sum_{j=1}^{\infty} a_j = \pi^2/6 + \sum_{j=1}^{\infty}\big((j^4+1)^{-1/2} - j^{-2}\big) = 1.64493 - 0.30119 = 1.3437.$$

Five terms on the right hand side are sufficient for four-place accuracy in the final result. Using the series on the left hand side, one would not get four-place accuracy until after 20,000 terms.

This technique is unusually successful in this example. The reader is advised to find out that and why it is less successful for $a_j = (j^4 + j^3 + 1)^{-1/2}$.

An important comparison sequence is a geometric sequence

$$y_n = a + bk^n,$$

for which

$$\nabla s_n = y_n - y_{n-1} = bk^{n-1}(k-1).$$

It this is fitted to the three most recently computed terms of a given sequence, $y_n = s_n$ for (say) $n = j, j-1, j-2$, then $\nabla y_j = \nabla s_j$, $\nabla y_{j-1} = \nabla s_{j-1}$, and

$$k = \nabla s_j / \nabla s_{j-1}.$$

Hence

$$bk^j = \frac{\nabla s_j}{1 - 1/k} = \frac{\nabla s_j}{1 - \nabla s_{j-1}/\nabla s_j} = \frac{(\nabla s_j)^2}{\nabla^2 s_j}.$$

This yields a comparison sequence for each j. Suppose that $|k| < 1$. Then the comparison sequence has the $s'_j = \lim_{n\to\infty} y_n = a = y_j - bk^j$, i.e.

$$s \approx s'_j = s_j - \frac{(\nabla s_j)^2}{\nabla^2 s_j}. \tag{3.4.2}$$

This is called **Aitken acceleration**[51] and is the most popular *nonlinear* acceleration methods.

If $\{s_n\}$ is exactly a geometric sequence, i.e. if $s_n - a = k(s_{n-1} - a) \ \forall \ n$, then $s'_j = s \ \forall j$. Otherwise it can be shown (Henrici [24, 1964]) that under the assumptions

$$\lim_{j\to\infty} s_j = s, \quad \text{and} \quad \lim \frac{s_{j+1} - s_j}{s_j - s_{j-1}} = k^*, \quad |k^*| < 1, \tag{3.4.3}$$

the sequence $\{s'_j\}$ converges faster than does the sequence $\{s_j\}$. The above assumptions can often be verified for sequences arising from iterative processes and for many other applications.

If you want the sum of slowly convergent *series*, it may seem strange to compute the sequence of partial sums, and then compute the first and second differences of rounded values of this sequence in order to apply Aitken acceleration. The *a-version* of Aitken acceleration works on the terms a_j of an infinite series instead of on its partial sums s_j.

Clearly we have $a_j = \nabla s_j$, $j = 1 : N$. The a-version of Aitken acceleration thus reads $s'_j = s_j - a_j^2/\nabla a_j$, $j = 1 : N$. We want to determine a'_j so that

$$\sum_{k=1}^{j} a'_k = s'_j, \quad j = 1 : N.$$

Then

$$a'_1 = 0, \quad a'_j = a_j - \nabla(a_j^2/\nabla a_j), \quad j = 2 : N,$$

and $s'_N = s_N - a_N^2/\nabla a_N$ (show this). We may expect that this a-version of Aitken acceleration handles rounding errors better.

The condition $|k^*| < 1$ is a *sufficient* condition only. In practice, Aitken acceleration seems *most efficient if* $k^* = -1$. Indeed, it often converges even if $k^* < -1$; see Problem 7. It is *much less successful if* $k^* \approx 1$, e.g., for slowly convergent series with positive terms.

The Aitken acceleration process can often be *iterated*, to yield sequences, $\{s''_n\}_0^\infty$, $\{s'''_n\}_0^\infty$, etc., defined by the formulas

$$s''_j = s'_j - \frac{(\nabla s'_j)^2}{\nabla^2 s'_j}, \qquad s'''_j = s''_j - \frac{(\nabla s''_j)^2}{\nabla^2 s''_j} \ldots \tag{3.4.4}$$

[51] Alexander Craig Aitken (1895–1967), Scotch mathematician born in New Zealand.

Example 3.4.2.

By (3.1.10), it follows for $x = 1$ that

$$1 - 1/3 + 1/5 - 1/7 + 1/9 - \ldots = \arctan 1 = \pi/4 \approx 0.7853981634.$$

This series converges very slowly. Even after 500 terms there still occur changes in the third decimal. Consider the partial sums $s_j = \sum_{n_0}^{j} (-1)^j (2n+1)^{-1}$, with $n_0 = 5$, and compute the **iterated Aitken** sequences as indicated above.

The (sufficient) theoretical condition mentioned above is not satisfied, since $\nabla s_n / \nabla s_{n-1} \to -1$ as $n \to \infty$. Nevertheless, we shall see that the Aitken acceleration works well, and that the iterated accelerations converge rapidly. One gains two digits for every pair of terms, in spite of the slow convergence of the original series. The results in the table below were obtained using IEEE double precision. The errors of s'_j, s''_j, ... are denoted e'_j, e''_j, ...

j	s_j	e_j	e'_j	e''_j	e'''_j
5	0.744012	−4.1387e−2			
6	0.820935	3.5536e−2			
7	0.754268	−3.1130e−2	−1.7783e−4		
8	0.813092	2.7693e−2	1.1979e−4		
9	0.760460	−2.4938e−2	−8.4457e−5	−1.3332e−6	
10	0.808079	2.2681e−2	6.1741e−5	7.5041e−7	
11	0.764601	−2.0797e−2	−4.6484e−5	−4.4772e−7	−1.0289e−8

Example 3.4.3.

Set $a_n = e^{-\sqrt{n+1}}$, $n \geq 0$. As before, we denote by s_n the partial sums of $\sum a_n$, $s = \lim s_n = 1.67040681796634$, and use the same notations as above. Note that

$$\nabla s_n / \nabla s_{n-1} = a_n / a_{n-1} \approx 1 - \tfrac{1}{2} n^{-1/2}, \quad (n \gg 1],$$

so this series is slowly convergent. Computations with plain and iterated Aitken in IEEE double precision gave the results below:

j	e_{2j}	$e_{2j}^{(j)}$
0	−1.304	−1.304
1	−0.882	−4.10e−1
2	−0.640	−1.08e−1
3	−0.483	−3.32e−2
2	−0.374	−4.41e−3
5	−0.295	−7.97e−4
6	−0.237	−1.29e−4
7	−0.192	−1.06e−5
8	−0.158	−1.13e−5

The sequence $\{e_{2j}^{(j)}\}$ is monotonic until $j = 8$. After this $|e_{2j}^{(j)}|$ is mildly fluctuating around 10^{-5} (at least until $j = 24$), and the differences $\nabla s_{2j}^{(j)} = \nabla e_{2j}^{(j)}$ are sometimes several powers of 10 smaller than the actual errors and are misleading as error estimates. The rounding errors have taken over, and it is almost no use to compute more terms.

It is possible to use more terms for obtaining higher accuracy by applying iterated Aitken acceleration to a **thinned sequence** e.g., s_4, s_8, $s_{12}, \ldots$, Problem 4. Note the thinning is performed on a *sequence* that converges to the limit to be computed, e.g., the partial sums of a series. Only in so-called *bell sums* (see Problem 30) we shall do *a completely different kind of thinning*, namely a thinning of the *terms* of a series.

The convergence ratios of the thinned sequence are much smaller; for the series of the previous example they become approximately

$$\left(1 - \tfrac{1}{2}n^{-1/2}\right)^4 \approx 1 - 2n^{-1/2}, \quad n \gg 1.$$

The most important point is, though, that the rounding errors become more slowly amplified, so that terms far beyond the eighth number of the un-thinned sequence can be used in the acceleration, resulting in a much improved final accuracy.

How to realize the thinning depends on the sequence; a different thinning will be used in the next example.

Example 3.4.4.

We shall compute,

$$s = \sum_{n=1}^{\infty} n^{-3/2} = 2.612375348685488.$$

If all partial sums are used in Aitken acceleration, it turns out that the error $|e_{2j}^{(j)}|$ is decreasing until $j = 5$, when it is 0.07, and it remains on approximately this level for a long time.

j	0	1	2	3	4	5
E_{2j+1}	−1.61	−0.94	−4.92e−1	−2.49e−1	−1.25e−1	−6.25e−2
$E_{2j+1}^{(j)}$	−1.61	−1.85	−5.06e−2	−2.37e−4	−2.25e−7	2.25e−10

A much better result is obtained by means of thinning, but since the convergence is much slower here than in the previous case, we shall try "geometric" thinning rather than the "arithmetic" thinning used above, i.e. we now set $S_m = s_{2^m}$. Then

$$\nabla S_m = \sum_{1+2^{m-1}}^{2^m} a_n, \quad S_j = S_0 + \sum_{m=1}^{j} \nabla S_m, \quad E_j = S_j - s.$$

(If maximal accuracy is wanted, it may be advisable to use the "divide and conquer technique" for computing these sums; see Problem 2.3.5, but it has not been

used here.) By the approximation of the sums by integrals one can show that $\nabla S_m/\nabla S_{m-1} \approx 2^{-1/2}$, $m \gg 1$. The table above shows the errors of the first thinned sequence and the results after iterated Aitken acceleration. The last result has used 1024 terms of the original series, but since

$$s_n - s = -\sum_{j=n}^{\infty} j^{-3/2} \approx -\int_n^{\infty} t^{-3/2}\,dt = -\frac{2}{3}n^{-1/2}, \tag{3.4.5}$$

10^{20} terms would have been needed for obtaining this accuracy without convergence acceleration.

For sequences such that

$$s_n - s = c_0 n^{-p} + c_1 n^{-p-1} + O(n^{-p-2}), \quad p > 0,$$

where s, c_0, c_1 are unknown, the following variant of Aitken acceleration, (Bjørstad et al. [3]) is more successful:

$$s'_n = s_n - \frac{p+1}{p}\frac{\Delta s_n \nabla s_n}{\Delta s_n - \nabla s_n}. \tag{3.4.6}$$

It turns out that s'_n is two powers of n more accurate than s_n,

$$s'_n - s = O(n^{-p-2}),$$

see Problem 12. More generally, suppose that there exists a longer (unknown) asymptotic expansion of the form

$$s_n = s + n^{-p}(c_0 + c_1 n^{-1} + c_2 n^{-2} + \ldots), \quad n \to \infty. \tag{3.4.7}$$

This is a rather common case. Then we can extend this to an to an *iterative variant*, where p is to be increased by 2 in each iteration; $i = 0, 1, 2, \ldots$ is a superscript, i.e.

$$s_n^{i+1} = s_n^i - \frac{p+2i+1}{p+2i}\frac{\Delta s_n^i \nabla s_n^i}{\Delta s_n^i - \nabla s_n^i}. \tag{3.4.8}$$

If p is also unknown, it can be estimated by means of the equation,

$$\frac{1}{p+1} = -\Delta\frac{\Delta s_n}{\Delta s_n - \nabla s_n} + O(n^{-2}). \tag{3.4.9}$$

Example 3.4.5.

We consider the same series as in the previous example, i.e. $s = \sum n^{-3/2}$. We use (3.4.8) without thinning. Here $p = -1/2$, see Problem 13. As usual, the errors are denoted $e_j = s_j - s$, $e_{2j}^i = s_{2j}^i - s$. In the right column of the table below, we show the errors from a computation with 12 terms of the original series,

From this point the errors were around 10^{-10} or a little below. The rounding errors have taken over, and the differences are, as in Example 3.3.4, misleading

j	e_{2j}	e_{2j}^j
0	−1.612	−1.612
1	−1.066	−8.217e−3
2	−0.852	−4.617e−5
3	−0.730	+2.528e−7
4	−0.649	−1.122e−9
5	−0.590	−6.34−12
6	−0.544	−1.322e−9

for error estimation. If needed, higher accuracy can be obtained by "arithmetic thinning" with more terms.

In this computation only 12 terms were used. In the previous example a less accurate result was obtained by means of 1024 terms of the same series, but we must appreciate that the technique of Example 3.3.5 did not require the existence of an asymptotic expansion for s_n and may therefore have a wider range of application.

There are not yet so many theoretical results that give justice to the practically observed efficiency of iterated Aitken accelerations for oscillating sequences. One reason for this can be that the transformation (3.4.2), which the algorithms are based on, is *nonlinear*). For methods of convergence acceleration that are based on *linear* transformations, theoretical estimates of convergence rates and errors are closer to the practical performance of the methods.

In a generalization of Aitken acceleration one considers a transformation that is exact for sequences satisfying

$$a_0(s_n - a) + \cdots + a_k(s_{n-k} - a) = 0, \quad \forall \ \ n. \tag{3.4.10}$$

Shanks considered the sequence transformation

$$e_k(s_n) = \frac{\begin{vmatrix} s_n & s_{n+1} & \cdots & s_{n+k} \\ s_{n+1} & s_{n+2} & \cdots & s_{n+k+1} \\ \vdots & \vdots & \cdots & \vdots \\ s_{n+k} & s_{n+k+1} & \cdots & s_{n+2k} \end{vmatrix}}{\begin{vmatrix} \Delta^2 s_n & \cdots & \Delta^2 s_{n+k-1} \\ \vdots & \cdots & \vdots \\ \Delta^2 s_{n+k-1} & \cdots & \Delta^2 s_{n+2k-2} \end{vmatrix}}, \quad k = 1, 2, 3, \ldots \tag{3.4.11}$$

For $k = 1$ Shanks' transformation reduces to Aitken's Δ^2 process. It can be proved that $e_k(s_n) = a$ if and only if s_n satisfies (3.4.10). The determinants in the definition of $e_k(s_n)$ have a very special structure and are called **Hankel determinants**[52]. Such determinants satisfy a recurrence relationship, which can be used for implementing the transformation. An elegant recursive procedure to compute $e_k(s_n)$ directly, the **epsilon algorithm**, will be discussed further in Sec. sec3.5.3 in connection with continued fraction and Padé approximants.

[52]Named after the German mathematician Hermann Hankel (1839–1873).

3.4.3 Euler's Transformation

In 1755 Euler gave the first version of what is now called **Euler's transformation**. Let

$$S = \sum_{j=0}^{\infty} (-1)^j u_j,$$

be an alternating series ($u_j \geq 0$). Then Euler showed that

$$S = \sum_{k=0}^{\infty} \frac{1}{2^k} \Delta^k u_k, \tag{3.4.12}$$

Often it is better to apply Euler's transformation to the tail of a series.

We shall now apply another method of acceleration based on **repeated averaging** of the partial sums. Consider again the same series as in Example 3.4.2, i.e..

$$\sum_{j=0}^{\infty} (-1)^j (2j+1)^{-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4}. \tag{3.4.13}$$

Let S_N be the sum of the first N terms. The columns to the right of the S_N-column in the scheme given in Table 3.4.1 are formed by building averages.

Each number in a column is the mean of the two numbers which stand to the left and upper left of the number itself. In other words, each number is the mean of its "west" and "northwest" neighbor. The row index of M tells how many terms are used from the original series, while the column index -1 equals the number of repeated averagings. Only the digits which are different from those in the previous column are written out.

Table 3.4.1. *Summation by repeated averaging.*

N	S_N	M_2	M_3	M_4	M_5	M_6	M_7
6	0.744 012						
7	0.820 935	782 474					
8	0.754 268	787 602	5038				
9	0.813 092	783 680	5641	340			
10	0.760 460	786 776	5228	434	387		
11	0.808 079	784 270	5523	376	405	396	
12	0.764 601	786 340	5305	414	395	400	398

Notice that the values in each column oscillate. In general, for an alternating series, it follows from the next theorem together with (3.2.4) that *if the absolute value of the* jth *term, considered as a function of* j*, has a* kth *derivative which approaches zero monotonically for* $j > N_0$*, then every other value in column* M_{k+1} *is larger than the sum, and every other is smaller.* The above premise is satisfied here, since if $f(j) = (2j+1)^{-1}$ then $f^{(k)}(j) = c_k(2j+1)^{-1-k}$, which approaches zero monotonically.

If round-off is ignored, it follows from column M_6 that $0.785396 \le \pi/4 \le 0.785400$. To take account of round-off error, we set $\pi/4 = 0.785398 \pm 3 \cdot 10^{-6}$. The actual error is only $1.6\,10^{-7}$. In Example3.4.2 iterated Aitken accelerations gave about one decimal digit more with the same data.

It is evident how the above method can be applied to any *alternating series*. The diagonal elements are equivalent to the results from using Euler's transformation.

Euler's transformation and the averaging method, can be generalized for the convergence acceleration of a general complex power series

$$S(z) = \sum_{j=1}^{\infty} u_j z^{j-1}. \tag{3.4.14}$$

The alternating series obtained for $z = -1$. Other applications include *Fourier series*. They can be brought to this form, with $z = e^{i\phi}$, $-\pi \le \phi \le \pi$; see Problem 14 and Example 3.4.7. The irregular errors of the coefficients play a big role if $|\phi| \ll \pi$, and it is important to reduce their effects by means of a variant of the thinning technique, described (for Aitken acceleration) in the previous section. Another interesting application is the *analytic continuation* of the power series outside its circle of convergence; see Example 3.4.8.

Theorem 3.4.1.

The tail of the power series in (3.4.14) *can formally be transformed into the expansion, ($z \neq 1$).*

$$S(z) - \sum_{j=1}^{n} u_j z^{j-1} = \sum_{j=n+1}^{\infty} u_j z^{j-1} = \frac{z^n}{1-z} \sum_{s=0}^{\infty} P^s u_{n+1}, \quad P = \frac{z}{1-z}\Delta. \tag{3.4.15}$$

Set $N = n + k - 1$, and set

$$M_{n,1} = \sum_{j=1}^{n} u_j z^{j-1}; \quad M_{N,k} = M_{n,1} + \frac{z^n}{1-z} \sum_{s=0}^{k-2} P^s u_{n+1}; \quad n = N - k + 1. \tag{3.4.16}$$

These quantities can be computed by the following recurrence formula that yields several estimates based on N terms from the original series.[53] *This is called the* **generalized Euler transformation**.

$$M_{N,k} = \frac{M_{N,k-1} - zM_{N-1,k-1}}{1-z}, \quad k = 2 : N. \tag{3.4.17}$$

For $z = -1$, this is the repeated average algorithm described above, and $P = -\frac{1}{2}\Delta$.

Assume that $|z| \le 1$, that $\sum u_j z^{j-1}$ converges, and that $\Delta^s u_N \to 0$, $s = 0 : k$ as $N \to \infty$. Then $M_{N,k} \to S(z)$, as $N \to \infty$. If, moreover, $\Delta^{k-1} u_j$ has a constant sign for $j \ge N - k + 2$, then the following strict error bounds are obtained:

$$|M_{N,k} - S(z)| \le |z(M_{N,k} - M_{N-1,k-1})| = |M_{N,k} - M_{N,k-1}|, \quad (k \ge 2). \tag{3.4.18}$$

[53] See Algorithm 3.3.1 for an adaptive choice of a kind of optimal output.

Proof. We first note that, as $N \to \infty$, $P^s u_N \to 0$, $s = 0 : k$, and hence, by (3.4.16), $\lim M_{N,k} = \lim M_{N,0} = S(z)$.

Euler's transformation can be formally derived by operators as follows:

$$S(z) - M_{n,1} = z^n \sum_{i=0}^{\infty} (zE)^i u_{n+1} = \frac{z^n}{1 - zE} u_{n+1}$$
$$= \frac{z^n}{1 - z - z\Delta} u_{n+1} = \frac{z^n}{1 - z} \sum_{s=0}^{\infty} P^s u_{n+1}.$$

In order to derive (3.4.17), note that this relation can equivalently be written thus,

$$M_{N,k} - M_{N,k-1} = z(M_{N,k} - M_{N-1,k-1}), \tag{3.4.19}$$

$$M_{N,k-1} - M_{N-1,k-1} = (1 - z)(M_{N,k} - M_{N-1,k-1}). \tag{3.4.20}$$

Remembering that $n = N - k + 1$, we obtain, by (3.4.16),

$$M_{N,k} - M_{N-1,k-1} = \frac{z^{N-k+1}}{1 - z} P^{k-2} u_{N-k+2}, \tag{3.4.21}$$

and it can be shown (Problem 17) that

$$M_{N,k-1} - M_{N-1,k-1} = z^n P^{k-2} u_{n+1} = z^{N-k+1} P^{k-2} u_{N-k+2}. \tag{3.4.22}$$

By (3.4.21) and (3.4.22), we now obtain (3.4.20) and hence also the equivalent equations (3.4.19) and (3.4.17).

Now substitute j for N into (3.4.22), and add the p equations obtained for $j = N + 1, \ldots, N + p$. We obtain:

$$M_{N+p,k-1} - M_{N,k-1} = \sum_{j=N+1}^{N+p} z^{j-k+1} P^{k-2} u_{j-k+2}.$$

Then substitute $k + 1$ for k, and $N + 1 + i$ for j. Let $p \to \infty$, while k is fixed. It follows that

$$S(z) - M_{N,k} = \sum_{j=N+1}^{\infty} z^{j-k} P^{k-1} u_{j-k+1} = \frac{z^{N-k+1} \cdot z^{k-1}}{(1 - z)^{k-1}} \sum_{i=0}^{\infty} z^i \Delta^{k-1} u_{N-k+2+i}, \tag{3.4.23}$$

hence

$$|S(z) - M_{N,k}| \le \left|(z/(1 - z))^{k-1} z^{N-k+1}\right| \sum_{i=0}^{\infty} \left|\Delta^{k-1} u_{N-k+2+i}\right|.$$

We now use the assumption that $\Delta^{k-1} u_j$ has constant sign for $j \ge N - k + 2$. Since $\sum_{i=0}^{\infty} \Delta^{k-1} u_{N-k+2+i} = -\Delta^{k-2} u_{N-k+2}$, it follows that

$$|S(z) - M_{N,k}| \le \left| z^{N-k+1} \frac{z^{k-1} \Delta^{k-2} u_{N-k+2}}{(1 - z)^{k-1}} \right| = \left| \frac{z \cdot z^{N-k+1}}{1 - z} P^{k-2} u_{N-k+2} \right|.$$

Now, by (3.4.21), $|S(z) - M_{N,k}| \leq |z| \cdot |M_{N,k} - M_{N-1,k-1}|$. This is the first part of (3.4.18). The second part then follows from (3.4.19). □

Comments: Note that the elements $M_{N,k}$ become rational functions of z for fixed N, k. If the term u_n, as a function of n, belongs to $\mathcal{P}_k$, then the classical Euler transformation (for $n = 0$) yields the exact value of $S(z)$ after k terms, if $|z| < 1$. This follows from (3.4.15), because $\sum u_j z^j$ is convergent, and $P^s u_{n+1} = 0$ for $s \geq k$. In this particular case, $S(z) = Q(z)(1-z)^{-k}$, where Q is a polynomial; in fact the Euler transformation gives $S(z)$ correctly for all $z \neq 1$.

The advantage of the recurrence formula (3.4.17), instead of a more direct use of (3.4.15), is that it provides a whole lower triangular matrix of estimates, so that one can, by means of a simple test, decide when to stop. This yields a result with strict error bound, if $\Delta^{k-1} u_j$ has a constant sign (for all j with a given k), and if the effect of rounding errors is evidently smaller than TOL. If these conditions are not satisfied, there is a small risk that the algorithm may terminate if the error estimate is incidentally small, e.g., near a sign change of $\Delta^{k-1} u_j$.

The irregular errors of the initial data are propagated to the results. In the long run, they are multiplied by approximately $|z/(1-z)|$ from a column to the next—this is less than one if $\Re z < 1/2$—but in the beginning this growth factor can be as large as $(1+|z|)/|1-z|$. It plays no role for alternating series; its importance when $|1-z|$ is smaller will be commented in Example 3.4.7.

The following algorithm is mainly based on the above theorem, but the possibility for the irregular errors to become dominant has been taken into account (somewhat) in the third alternative of the termination criterion.

Algorithm 3.4.1 The Generalized Euler Transformation

This algorithm is based on Theorem 3.4.1, with a tolerance named TOL, and a termination criterion based on (3.4.18), by the computation and inspection of the elements of M in a certain order, until it finds a pair of neighboring elements that satisfies the criterion.
The classical Euler transformation would only consider the diagonal elements M_{NN}, $N = 1, 2, ...$ and the termination would have been based on $|M_{NN} - M_{N-1,N-1}|$. The strategy used in this algorithm is superior for an important class of series.

```
function [sum,errest,n,kk] = euler(z,u,Tol)
%
% EULER applies the generalized Euler transform to a power series
% with coeffcients given by u at z. The elements of M are
% inspected in a certain order, until a pair of neighboring
% elements are found that satisfies a termination criterion.
Nmax = length(u);
errest = Inf;  olderrest = errest;
N = 1;  kk = 1; M(1,1) = u(1);
while (errest > Tol) & (N < Nmax) & (errest < olderrest)
    N = N+1;  olderrest = errest;
```

```
        M(N,1) = M(N-1,1)+ u(N)*z^(N-1); % New partial sum
        for k = 2:N,
           M(N,k) = (M(N,k-1) - z*M(N-1,k-1))/(1-z);
           temp = abs(M(N,k) - M(N,k-1))/2;
           if temp < errest,
              kk = k; errest = temp;
           end
        end
     end
     sum = (M(N,kk) + M(N,kk-1))/2;
```

An oscillatory behavior of the values $|M_{N,k} - M_{N,k-1}$ in the same row, indicates that the irregular errors have become dominant. The smallest error estimates may then become unreliable.

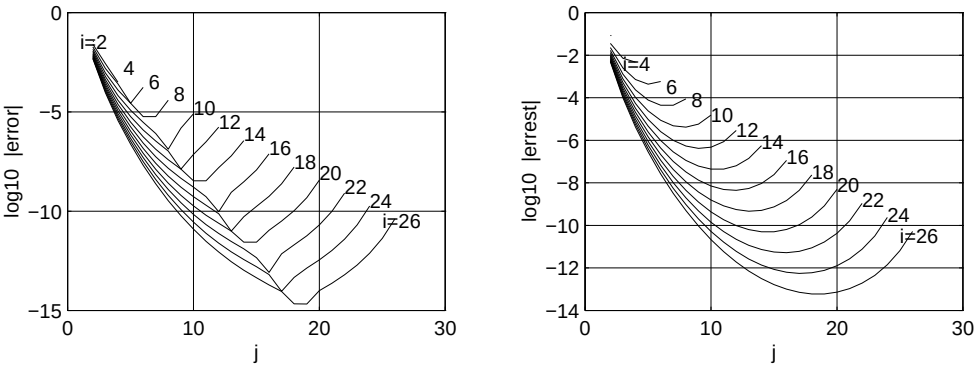


Figure 3.4.1. *Logarithms of the actual errors and the error estimates for $M_{N,k}$ in a more extensive computation for the alternating series in (3.4.13) with completely monotonic terms. The tolerance is here set above the level, where the irregular errors become important; for a smaller tolerance parts of the lowest curves may become less smooth in some parts.*

The above algorithm gives a strict error bound if, in the notation used in the theorem, $\Delta^{k-1}u_i$ has a constant sign for $i \geq N - k + 2$ (in addition to the other conditions of the theorem). We recall that a sequence, for which this condition is satisfied *for every* k, is called completely monotonic; see Definition 3.2.6.

It may seem difficult to check if this condition is satisfied. It turns out that many sequences that can be formed from sequences like $\{n^{-\alpha}\}$, $\{e^{-\alpha n}\}$ by simple operations and combinations, belong to this class. The generalized Euler transformation yields a sequence that converges at least as fast as a geometric series. The convergence ratio depends on z; it is less than one in absolute value for any complex z, except for $z > 1$ on the real axis. *So, the generalized Euler transformation often provides an analytic continuation of a power series outside its circle of convergence.*

For *alternating series*, with completely monotonic terms, i.e. for $z = -1$, the convergence ratio typically becomes $\frac{1}{3}$. This is in good agreement with Figure 3.4.1.

Note that the minimum points for the errors lie almost on a straight line in Figure 3.5.1, and that the optimal value of k/N is approximately $\frac{2}{3}$, if $N \gg 1$, and if there are no irregular errors.

Example 3.4.6.

A program, essentially the same as Algorithm 3.4.3, is applied to the series

$$\sum_{j=1}^{\infty}(-1)^j j^{-1} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \ldots = \ln 2 = 0.69314\,71805\,599453.$$

with TOL$= 10^{-6}$, It stops when $N = 12$, $kk = 9$. The errors $e_k = M_{N,k} - \ln 2$ and the differences $\frac{1}{2}\nabla_k M_{N,k}$ along the last row of M read:

k	1	2	3	...	9	10	11	12
e_k	−3.99e-2	1.73e-3	−1.64e-4	...	−4.51e-7	5.35e-7	−9.44e-7	2.75e-6
$\nabla/2$		2.03e-2	−9.47e-4	...	−4.93e-7	4.93e-7	−7.40e-7	1.85e-6

Note that $|errest| = 4.93\,10^{-7}$ and $sum - \ln 2 = \frac{1}{2}(e_9 + e_8) = 4.2\,10^{-8}$. Almost full accuracy is obtained for TOL $= 10^{-16}$, $maxN = 40$. The results are $N = 32$, $kk = 22$, $errest = 10^{-16}$, $|error| = 2\,10^{-16}$. Note that $errest < |error|$; this can happen when we ask for such a high accuracy that the rounding errors are not negligible.

Example 3.4.7. *Application to Fourier series.*

Consider a complex power series

$$S(z) = \sum_{n=1}^{\infty} u_n z^{n-1}, \quad z = e^{i\phi}.$$

A Fourier series that is originally of the form $\sum_{-\infty}^{\infty}$ or in trigonometric form, can easily be brought to this form; see Problem 14. As we shall see, the results can often be improved considerably by the application of thinning. Let THIN be a positive integer. The thinned form of $S(z)$ reads

$$S(z) = \sum_{p=1}^{\infty} u_p^* z^{\mathrm{THIN}\cdot(p-1)}, \quad u_p^* = \sum_{j=1}^{thin} u_{j+thin\cdot(p-1)}\, z^{j-1}.$$

For example, if $z = e^{i\pi/3}$ and THIN $=3$, the series becomes an alternating series, perhaps with complex coefficients. It does not matter in the numerical work that u_p^* depends on z.

We consider the case $S(z) = -\ln(1-z)/z = \sum z^{n-1}/n$, which is typical for a power series with completely monotonic terms. (The rates of convergence are the same for almost all series of this class.) Numerical computation, essentially by the above algorithm, gave the following results. The coefficients u_j are computed in IEEE double precision. We make the rounding errors during the computations less

important by subtracting the first row of partial sums by its last element; it is, of course, added again to the final result.[54] The first table shows, for various ϕ, the most accurate result that can be obtained without thinning. These limits are due to the rounding errors; we can make the pure truncation error arbitrarily small by choosing N large enough.

ϕ	π	$2\pi/3$	$\pi/2$	$\pi/3$	$\pi/4$	$\pi/6$	$\pi/8$	$\pi/12$	$\pi/180$
|error|	2e-16	8e-16	1e-14	6e-12	1e-9	7e-8	5e-7	3e-5	2e-1
N	30	33	36	36	36	36	40	40	100
kk	21	22	20	21	20	14	13	10	(3)

Note that a rather good accuracy is obtained also for $\phi = \pi/8$ and $\phi = \pi/12$, where the algorithm is "unstable", since $|\frac{z}{1-z}| > 1$. In this kind of computations "instability" does not mean that the algorithm is hopeless, but it shows the importance of a good termination criterion. The question is to navigate safely between Scylla and Charybdis. For a small value like $\phi = \pi/180$, the sum is approximately $4.1 + 1.5i$. The smallest error with 100 terms (or less) is 0.02; it is obtained for $k = 3$. Also note that kk/N increases with ϕ.

By *thinning*, much better results are obtained for $\phi \ll \pi$, in particular for $\phi = \pi/180$. This series that has "essentially positive" terms originally can become "essentially alternating" by thinning. We present the errors obtained for four values of the parameter THIN, with different amount of work. Compare $|error|$, kk, etc. with appropriate values in the table above. We see that, by thinning, it is possible to calculate the Fourier series very accurately also for small values of ϕ.

THIN	80	120	90	15
$thin \cdot \phi$	π	$2\pi/3$	$\pi/2$	$\pi/12$
|error|	2e-14	1e-14	3e-13	3e-5
N	28	31	33	41
kk	20	22	18	10
total no. terms	5040	3720	2970	615

Roughly speaking, the optimal convergence rate of the Euler Transformation depends on z in the same way for all power series with completely monotonic coefficients; independently of the rate of convergence of the original series. The above tables from a particular example can therefore—with some safety margin—be used as a guide for the application of the Euler transformation with thinning to any series of this class.

Say that you want the sum of a series $\sum u_n z^n$ for $z = e^{i\phi}$, $\phi = \pi/12$, with relative $|error| < 10^{-10}$. You see in the first table that $|error| = 6\,10^{-12}$ for $\phi = \pi/3 = 4\pi/12$ without thinning. The safety margin is hopefully large enough. Therefore, try $Thin = 4$. We make two tests with completely monotonic terms: $u_n = n^{-1}$ and $u_n = \exp(-\sqrt{n})$. $Tol = 10^{-10}$ is hopefully large enough to make the irregular errors relatively negligible. In both tests the actual $|error|$ turns out

[54] Tricks like this can often be applied in linear computations with a slowly varying sequence of numbers. See, e.g., the discussion of rounding errors in Richardson extrapolation in Sec. 3.3.5.

to be $4\,10^{-11}$, and the total number of terms is $4 \cdot 32 = 128$. The values of *errest* are $6\,10^{-11}$ and $7\,10^{-11}$; both slightly overestimate the actual errors and are still smaller than TOL.

Example 3.4.8. *Application to a divergent power series, (analytic continuation).*
Consider a complex power series

$$S(z) = \sum_{n=1}^{\infty} u_n z^{n-1}, \quad |z| > 1.$$

As in the previous example we study in detail the case of $u_n = 1/n$. It was mentioned above that the generalized Euler transformation theoretically converges in the z-plane, cut along the interval $[1, \infty]$. The limit is $-z^{-1}\ln(1-z)$, a single-valued function in this region. For various z outside the unit circle, we shall see that rounding causes bigger problems here than for Fourier series. The error estimate of Algorithm 3.3.1, usually underestimated the error, sometimes by a factor of ten. The table reports some results from experiments without thinning.

z	-2	-4	-10	-100	-1000	$2i$	$8i$	$1+i$	$2+i$
|error|	2e-12	2e-8	4e-5	3e-3	5e-2	8e-11	1e-3	1e-7	2e-2
N	38	41	43	50	51	40	39	38	39
kk	32	34	39	50	51	28	34	22	24

Thinning can be applied also in this application, but here not only the argument ϕ is increased (this is good), but also $|z|$ (this is bad). Nevertheless, for $z = 1+i$, the error becomes 10^{-7}, $3\,10^{-9}$, 10^{-9}, $4\,10^{-8}$, for $thin = 1,\ 2,\ 3,\ 4$, respectively. For $z = 2+i$, however, thinning improved the error only from 0.02 to 0.01. All this is for IEEE double precision.

We shall encounter other methods for alternating series and complex power series, which are even more efficient than the generalized Euler transformation; see the epsilon algorithm in Sec. 3.5.3.

3.4.4 Euler–Maclaurin's Formula

In the summation of series with essentially positive terms the tail of the sum can be approximated by an integral by means of the trapezoidal rule.

As an example, consider the sum $S = \sum_{j=1}^{\infty} j^{-2}$. The sum of the first nine terms is, to four decimal places, 1.5398. It immediately occurs to one to compare the tail of the series with the integral of x^{-2} from 10 to ∞. We approximate the integral according to the trapezoidal rule; see Sec. 1.2

$$\int_{10}^{\infty} x^{-2}\,dx \approx T_1 + T_2 + T_3 + \ldots = \frac{1}{2}(10^{-2} + 11^{-2}) + \frac{1}{2}(11^{-2} + 12^{-2}) + \ldots$$
$$= \sum_{j=10}^{\infty} j^{-2} - \frac{1}{2}10^{-2}.$$

Hence it follows that

$$\sum_{j=1}^{\infty} j^{-2} \approx 1.5398 + [-x^{-1}]_{10}^{\infty} + 0.0050 = 1.5398 + 0.1050 = 1.6448.$$

The correct answer is $\pi^2/6 = 1.64493406684823$. We would have needed about 10,000 terms to get the same accuracy by direct addition of the terms!

The above procedure is not a coincidental trick, but a very useful method. A further systematic development of the idea leads to the important Euler–Maclaurin summation formula. We first derive this heuristically by operator techniques and exemplify its use, including a somewhat paradoxical example that shows that a strict treatment with the consideration of the remainder term is necessary for very practical reasons. Since this formula has several other applications, e.g., in numerical integration, we formulate it more generally than needed for the summation of infinite series.

Consider to begin with a rectangle sum on the finite interval $[a, b]$, with n steps of equal length h, $a + nh = b$; with the operator notation introduced in Sec. 3.2.2.

$$h\sum_{i=0}^{n-1} f(a+ih) = h\sum_{i=0}^{n-1} E^i f(a) = h\frac{E^n-1}{E-1}f(a) = \frac{(E^n-1)}{D}\frac{hD}{e^{hD}-1}f(a).$$

We apply, to the second factor, the expansion derived in Example 3.1.5, with the Bernoulli numbers B_ν. (Recall that $a + nh = b$, $E^n f(a) = f(b)$, etc.)

$$h\sum_{i=0}^{n-1} f(a+ih) = \frac{(E^n-1)}{D}\left(1 + \sum_{\nu=1}^{\infty}\frac{B_\nu (hD)^\nu}{\nu!}\right) f(a) \tag{3.4.24}$$

$$= \int_a^b f(x)\,dx + \sum_{\nu=1}^{k}\frac{h^\nu B_\nu}{\nu!}\big(f^{(\nu-1)}(b) - f^{(\nu-1)}(a)\big) + R_{k+1}.$$

Here R_{k+1} is a remainder term that will be discussed thoroughly in Theorem 3.4.3. Set $h = 1$, and assume that $f(b)$, $f'(b)$, ... tend to zero as $b \to \infty$. Recall that $B_1 = -\frac{1}{2}$, $B_{2j+1} = 0$ for $j > 0$, and set $k = 2r+1$. This yields **Euler–Maclaurin's summation formula**[55]

$$\sum_{i=0}^{\infty} f(a+i) = \int_a^{\infty} f(x)\,dx + \frac{f(a)}{2} - \sum_{j=1}^{r}\frac{B_{2j} f^{(2j-1)}(a)}{(2j)!} + R_{2r+2} \tag{3.4.25}$$

$$= \int_a^{\infty} f(x)\,dx + \frac{f(a)}{2} - \frac{f'(a)}{12} + \frac{f^{(3)}(a)}{720} - \dots$$

in a form suitable for the convergence acceleration of series of essentially positive terms. We tabulate a few coefficients related to the Bernoulli and the Euler numbers.

[55] Leonhard Euler (1707–1783), incredibly prolific Swiss mathematician. He gave fundamental contributions to many branches of mathematics and to the mechanics of rigid and deformable bodies as well as to fluid mechanics. Colin Maclaurin (1698–1764), British mathematician. They apparently discovered the summation formula independently; see Goldstine [21, p. 84]. Euler's publication came 1738.

Table 3.4.2. *Bernoulli and Euler numbers;* $B_1 = -1/2, E_1 = 1$.

$2j$	0	2	4	6	8	10	12
B_{2j}	1	$\frac{1}{6}$	$-\frac{1}{30}$	$\frac{1}{42}$	$-\frac{1}{30}$	$\frac{5}{66}$	$-\frac{691}{2730}$
$\frac{B_{2j}}{(2j)!}$	1	$\frac{1}{12}$	$-\frac{1}{720}$	$\frac{1}{30240}$	$-\frac{1}{1209600}$	$\frac{1}{47900160}$	
$\frac{B_{2j}}{2j(2j-1)}$	1	$\frac{1}{12}$	$-\frac{1}{360}$	$\frac{1}{1260}$	$-\frac{1}{1680}$	$\frac{1}{1188}$	$-\frac{691}{360360}$
E_{2j}	1	-1	5	-61	1385	-50521	2702765

There are some obscure points in this operator derivation, but we shall consider it as a heuristic calculation only and shall not try to legitimate the various steps of it. With an appropriate interpretation, a more general version of this formula will be proved by other means in Theorem 3.4.3. A general remainder term is obtained there, if you let $b \to \infty$ in (3.4.31). You do not need it often, because the following much simpler error bound is usually applicable—but there are exceptions.

The Euler–Maclaurin expansion (on the right hand side) is typically semi-convergent only. Nevertheless a few terms of the expansion often gives startlingly high accuracy with simple calculations. For example, if $f(x)$ is completely monotonic, i.e. if

$$(-1)^j f^{(j)}(x) \geq 0, \quad x \geq a, \quad j \geq 0,$$

then the partial sums oscillate strictly around the true result; the first neglected term is then a strict error bound. (This statement also follows from the theorem below.)

Before we prove the theorem we shall exemplify how the summation formula is used in practice.

Example 3.4.9.

We return to the case of computing $S = \sum_{j=1}^{\infty} j^{-2}$. and treat it with more precision and accuracy. With $f(x) = x^{-2}$, $a = 10$, we find $\int_a^{\infty} f(x)dx = a^{-1}$, $f'(a) = -2a^{-3}$, $f'''(a) = -24a^{-5}, \ldots$. By (3.4.25), $(r = 2)$,

$$\begin{aligned}\sum_{x=1}^{\infty} x^{-2} &= \sum_{x=1}^{9} x^{-2} + \sum_{i=0}^{\infty} (10+i)^{-2} \\ &= 1.53976\,7731 + 0.1 + 0.005 + 0.00016\,6667 - 0.00000\,0333 + R_6 \\ &= 1.64493\,4065 + R_6.\end{aligned}$$

Since $f(x) = x^{-2}$ is completely monotonic, the first neglected term is a strict error bound; it is less than $720\,10^{-7}/30240 < 3 \cdot 10^{-9}$. (The actual error is approximately $2 \cdot 10^{-9}$.)

Although the Euler–Maclaurin expansion, in this example; seems to converge rapidly, it is in fact, only semi-convergent for any $a > 0$, and this is rather typical.

We have namely $f^{(2r-1)}(a) = -(2r)!a^{-2r-1}$, and, by Example 3.1.5, $B_{2r}/(2r)! \approx (-1)^{r+1}2(2\pi)^{-2r}$. The ratio of two successive terms is thus $-(2r+2)(2r+1)/(2\pi a)^2$, hence the modulus of terms increase when $2r+1 > 2\pi a$.

The "rule" that one should terminate a semi-convergent expansion at the term of smallest magnitude, is in general no good for Euler–Maclaurin applications, since the high order derivatives (on the right hand side) are typically much more difficult to obtain than a few more terms in the expansion on the left hand side. Typically, you first choose r, $r \leq 3$, depending on how tedious the differentiations are, and then you choose a in order to meet the accuracy requirements.

In this example we were lucky to have access to simple closed expressions for the derivatives and the integral of f. In other cases, one may use the possibilities for the numerical integration on an infinite interval mentioned in Chapter 5. In Problem 20 (a) you find two formulas that result from the substitution of the formulas (3.3.50) that express higher derivatives in terms of central differences into the Euler–Maclaurin expansion.

An expansion of $f(x)$ into negative powers of x is often useful both for the integral and for the derivatives.

Example 3.4.10.

We consider $f(x) = (x^3+1)^{-1/2}$, for which the expansion

$$f(x) = x^{-3/2}(1+x^{-3})^{-1/2} = x^{-1.5} - \frac{1}{2}x^{-4.5} + \frac{3}{8}x^{-7.5} - \dots$$

was derived and applied in Example 3.1.6. It was found that

$$\int_{10}^{\infty} f(x)dx = 0.632410375,$$

correctly rounded, and that $f'''(10) = -4.13 \cdot 10^{-4}$ with less than 1% error. The $f'''(10)$ term in the Euler–Maclaurin expansion is thus $-5.73\,10^{-7}$, with absolute error less than $6 \cdot 10^{-9}$. Inserting this into Euler–Maclaurin's summation formula, together with the numerical values of $\sum_{n=0}^{9} f(n)$ and $\frac{1}{2}f(10) - \frac{1}{12}f'(10)$, we obtain $\sum_{n=0}^{\infty} f(n) = 3.7941\,1570 \pm 10^{-8}$. The reader is advised to work out the details as an exercise.

Example 3.4.11.

Let $f(x) = e^{-x^2}$, $a = 0$. Since all derivatives of odd order vanish at $a = 0$, then the expansion (3.4.25) may give the impression that $\sum_{j=0}^{\infty} e^{-j^2} = \int_0^{\infty} e^{-x^2}\,dx + 0.5 = 1.386\,2269$, but the sum (that is easily computed without any convergence acceleration) is actually 1.386 3186, hence the remainder R_{2r+2} cannot tend to zero as $r \to \infty$. The infinite Euler–Maclaurin expansion, where all terms but two are zero, *is convergent but is not valid*. Recall the distinction between the convergence and the validity of an infinite expansion, made in Sec. 3.1.2.

In this case $f(x)$ is not completely monotonic; for example, $f''(x)$ changes sign at $x = 1$. With appropriate choice of r, the general error bound (3.4.31) will tell that the error is very small, but it cannot be used for proving that it is zero—because this is not true.

The mysteries of these examples have hopefully raised the appetite for a more substantial theory, including an error bound for the Euler–Maclaurin formula. We first need some tools that are interesting in their own right.

The **Bernoulli polynomial** $B_n(t)$ is an $n'th$ degree polynomial defined by the **symbolic** relation $B_n(t) = (B + t)^n$, where the exponents of B become subscripts after the expansion according to the binomial theorem. The Bernoulli numbers B_j were defined in Example 3.1.5. Their recurrence relation (3.1.16) can be written in the form

$$\sum_{j=0}^{n-1} \binom{n}{j} B_j = 0, \quad n \geq 2,$$

or "symbolically" $(B + 1)^n = B^n = B_n$, (for the computation of B_{n-1}), $n \neq 1$, hence $B_0(t) = 1$, $B_1(t) = t + B_1 = t - 1/2$ and

$$B_n(1) = B_n(0) = B_n, \quad n \geq 2,$$

The **Bernoulli function** $\hat{B}_n(t)$ is a *piecewise polynomial* defined for $t \in \mathbf{R}$ by the equation $\hat{B}_n(t) = B_n(t - [t])$. (Note that $\hat{B}_n(t) = B_n(t)$ if $0 \leq t < 1$.)

Lemma 3.4.2.

(a) $\hat{B}'_{n+1}(t)/(n+1)! = \hat{B}_n(t)/n!$, $(n > 0)$,
$\hat{B}_n(0) = B_n$. *(For $n = 1$ this is the limit from the right.)*

$$\int_0^1 \frac{B_n(t)}{n!}\, dt = \begin{cases} 1, & \text{if } n = 0; \\ 0, & \text{otherwise.} \end{cases}$$

(b) *The piecewise polynomials $\hat{B}_p(t)$ are periodic; $\hat{B}_p(t + 1) = \hat{B}_p(t)$. $\hat{B}_1(t)$ is continuous, except when t is an integer. For $n \geq 2$, $\hat{B}_n \in C^{n-2}(-\infty, \infty)$.*

(c) *The Bernoulli functions have the following (modified) Fourier expansions,* $(r \geq 1)$,

$$\frac{\hat{B}_{2r-1}(t)}{(2r-1)!} = (-1)^r 2 \sum_{n=1}^{\infty} \frac{\sin 2n\pi t}{(2n\pi)^{2r-1}}, \quad \frac{\hat{B}_{2r}(t)}{(2r)!} = (-1)^{r-1} 2 \sum_{n=1}^{\infty} \frac{\cos 2n\pi t}{(2n\pi)^{2r}}.$$

Note that $\hat{B}_n(t)$ is an even (odd) function, when n is (even odd).

(d) $|\hat{B}_{2r}(t)| \leq |B_{2r}|$.

Proof. Statement (a) follows directly from the symbolic binomial expansion of the Bernoulli polynomials.

The demonstration of statement (b) is left for a problem. The reader is advised to draw the graphs of a few low order Bernoulli functions.

The Fourier expansion for $\hat{B}_1(t)$ follows from the Fourier coefficient formulas (3.2.7), (modified for the period 1 instead of 2π). The expansions for $\hat{B}_p(t)$, are then

obtained by repeated integrations, term by term, with the use of (a). Statement (d) then follows from the Fourier expansion, because $\hat{B}_{2r}(0) = B_{2r}$. □

Comments: For $t = 0$ we obtain an interesting classical formula, together with a useful asymptotic approximation that was obtained in a different way in Sec. 3.1.

$$\sum_{n=1}^{\infty} n^{-2r} = \frac{|B_{2r}|(2\pi)^{2r}}{2(2r)!}; \qquad \frac{|B_{2r}|}{(2r)!} \sim \frac{2}{(2\pi)^{2r}}. \tag{3.4.26}$$

Also note, how the rate of decrease of the Fourier coefficients is related to the type of singularity of the Bernoulli function at the integer points. (It does not help that the functions are smooth in the interval $[0, 1]$.)

The Bernoulli polynomials have a generating function that is elegantly obtained by means of the following "symbolic" calculation.

$$\sum_{0}^{\infty} \frac{B_n(y)x^n}{n!} = \sum_{0}^{\infty} \frac{(B+y)^n x^n}{n!} = e^{(B+y)x} = e^{Bx}e^{yx} = \frac{xe^{yx}}{e^x - 1}. \tag{3.4.27}$$

If the series is interpreted as a power series in the complex variable x, the convergence radius is 2π.

Theorem 3.4.3. The Euler–Maclaurin Formula.

Set $x_i = a + ih$, $x_n = b$, suppose that $f \in C^{2r+2}(a, b)$, and let $\hat{T}(a : h : b)f$ be the trapezoidal sum

$$\hat{T}(a : h : b)f = \sum_{i=1}^{n} \frac{h}{2}\big(f(x_{i-1}) + f(x_i)\big) = h\bigg(\sum_{i=0}^{n-1} f(x_i) + \tfrac{1}{2}(f(b) - f(a))\bigg). \tag{3.4.28}$$

Then

$$\hat{T}(a : h : b)f - \int_a^b f(x)\,dx = \frac{h^2}{12}\big(f'(b) - f'(a)\big) - \frac{h^4}{720}\big(f'''(b) - f'''(a)\big) \tag{3.4.29}$$
$$+ \ldots + \frac{B_{2r}h^{2r}}{(2r)!}\big(f^{(2r-1)}(b) - f^{(2r-1)}(a)\big) + R_{2r+2}(a, h, b)f.$$

The remainder $R_{2r+2}(a, h, b)f$ is $O(h^{2r+2})$. It is represented by an integral with a kernel of constant sign in (3.4.30)*. An upper bound for the remainder is given in* (3.4.31)*. The estimation of the remainder is very simple in certain important particular cases:*

- *If $f^{(2r+2)}(x)$ does not change sign in the interval $[a, b]$ then $R_{2r+2}(a, h, b)f$ has the same sign as the first neglected*[56] *term.*
- *If $f^{(2r+2)}(x)$ and $f^{(2r)}(x)$ have the same constant sign in $[a, b]$, then the value of the left hand side of* (3.4.29) *lies between the values of the partial sum of the expansion displayed in* (3.4.29) *and the partial sum with one term less.*[57]*.*

[56] If $r = 0$ all terms of the expansion are "neglected".

[57] Formally this makes sense for $r \geq 2$ only, but if we interpret $f^{(-1)}$ as "the empty symbol", it makes sense also for $r = 1$. If f is completely monotonic *the statement holds for every* $r \geq 1$. This is easy to apply, because simple criteria for complete monotonicity etc. are given in Sec. 3.3.6

In the limit, as $b \to \infty$, these statements still hold—also for the summation formula (3.4.25)*—provided that the left hand side of* (3.4.29) *and the derivatives $f^{(\nu)}(b)$ ($\nu = 1 : 2r+1$) tend to zero, if it is also assumed that*

$$\int_a^\infty |f^{(2r+2)}(x)|\,dx < \infty.$$

Proof. To begin with we consider a single term of the trapezoidal sum, and set $x = x_{i-1} + ht$, $t \in [0,1]$, $f(x) = F(t)$. Suppose that $F \in C^p[0,1]$, where p is an *even* number.

We shall apply *repeated integration by parts*, Lemma 3.2.7, to the integral $\int_0^1 F(t)\,dt = \int_0^1 F(t)B_0(t)\,dt$. Use statement (a) of Lemma 3.4.2 in the equivalent form, $\int B_j(t)/j!\,dt = (B_{j+1}(t)/(j+1)!$

Consider the first line of the expansion in the next equation. Recall that $B_\nu = 0$ if ν is odd and $\nu > 1$. Since $B_{j+1}(1) = B_{j+1}(0) = B_{j+1}$, j will thus be odd in all non-zero terms, except for $j = 0$. Then, with no loss of generality, we assume that p is even.

$$\begin{aligned}
\int_0^1 F(t)\,dt &= \sum_{j=0}^{p-1}(-1)^j F^{(j)}(t)\frac{B_{j+1}(t)}{(j+1)!}\Big|_{t=0}^1 + (-1)^p\int_0^1 F^{(p)}(t)\frac{B_p(t)}{p!}\,dt\\
&= \frac{F(1)+F(0)}{2} + \sum_{j=1}^{p-1}\frac{-B_{j+1}}{(j+1)!}(F^{(j)}(1) - F^{(j)}(0)) + \int_0^1 F^{(p)}(t)\frac{B_p(t)}{p!}\,dt\\
&= \frac{F(1)+F(0)}{2} - \sum_{j=1}^{p-3}\frac{B_{j+1}}{(j+1)!}(F^{(j)}(1) - F^{(j)}(0)) - \int_0^1 F^{(p)}(t)\frac{B_p - B_p(t)}{p!}\,dt.
\end{aligned}$$

The upper limit of the sum is reduced to $p-3$, since the last term (with $j = p-1$) has been moved under the integral sign, and all values of j are odd. Set $j+1 = 2k$ and $p = 2r+2$. Then k is an integer that runs from 1 to r. Hence

$$\sum_{j=1}^{p-3}\frac{B_{j+1}}{(j+1)!}(F^{(j)}(1) - F^{(j)}(0)) = \sum_{k=1}^{r}\frac{B_{2k}}{(2k)!}(F^{(2k-1)}(1) - F^{(2k-1)}(0)).$$

Now set $F(t) = f(x_{i-1}+ht)$, $t \in [0,1]$. Then $F^{(2k-1)}(t) = h^{2k-1}f^{(2k-1)}(x_{i-1}+ht)$, and make abbreviations like $f_i = f(x_i)$, $f_i^{(j)} = f^{(j)}(x_i)$ etc..

$$\int_{x_{i-1}}^{x_i} f(x)\,dx = h\int_0^1 F(t)\,dt = \frac{h(f_{i-1}+f_i)}{2} - \sum_{k=1}^{r}\frac{B_{2k}h^{2k}}{(2k)!}(f_i^{(2k-1)} - f_{i-1}^{(2k-1)}) - R,$$

where R is the local remainder that is now an integral over $[x_{i-1}, x_i]$. Adding these equations, for $i = 1 : n$, yields a result equivalent to (3.4.29), namely

$$\int_a^b f(x)\,dx = \hat{T}(a:h:b)f - \sum_{k=1}^{r}\frac{B_{2k}h^{2k}}{(2k)!}f^{(2k-1)}(x)\Big|_{x=a}^b - R_{2r+2}(a,h,b)f,$$

$$R_{2r+2}(a,h,b)f = h^{2r+2}\int_a^b \Big(B_{2r+2} - \hat{B}_{2r+2}((x-a)/h)\Big)\frac{f^{(2r+2)}(x)}{(2r+2)!}\,dx. \quad (3.4.30)$$

By Lemma 3.4.2, $|\hat{B}_{2r+2}(t)| \le |B_{2r+2}|$, hence the kernel $B_{2r+2} - \hat{B}_{2r+2}((x-a)/h)$ has the same sign as B_{2r+2}. Suppose that $f^{(2r+2)}(x)$ does not change sign on (a,b). Then

$$sign\, f^{(2r+2)}(x) = sign\,\big(f^{(2r+1)}(b) - f^{(2r+1)}(a)\big),$$

hence $R_{2r+2}(a,h,b)f$ has the same sign as the first neglected term.
The second statement about "simple estimation of the remainder" then follows from Theorem 3.1.3, since the Bernoulli numbers (with even subscripts) have alternating signs.

If $sign\, f^{(2r+2)}(x)$ is not constant, then we note instead that

$$|B_{2r+2} - \hat{B}_{2r+2}((x-a)/h)| \le |2B_{2r+2}|,$$

and hence

$$\begin{aligned}|R_{2r+2}(a,h,b)f| &\le h^{2r+2}\frac{|2B_{2r+2}|}{(2r+2)!}\int_a^b |f^{(2r+2)}(x)|dx \\ &\approx 2\Big(\frac{h}{2\pi}\Big)^{2r+2}\int_a^b |f^{(2r+2)}(x)|dx. \end{aligned} \quad (3.4.31)$$

If $\int_a^\infty |f^{(2r+2)}(x)|dx < \infty$ this holds also in the limit as $b \to \infty$.
□

Note that there are (at least) three parameters here that can be involved in *different* natural limit processes: For example, one of the parameters can tend to its limit, while the two others are kept fixed. The remainder formula (3.4.31) contains all you need for settling various questions about convergence.

- $b \to \infty$; natural when Euler–Maclaurin's formula is used as a summation formula, or for deriving an approximation formula valid when b is large.
- $h \to 0$; natural when Euler–Maclaurin's formula is used in connection with numerical integration. You see how the values of derivatives of f at the endpoints a, b can highly improve the estimate of the integral of f, obtained by the trapezoidal rule with constant step size. Euler–Maclaurin's formula is also useful for the design and analysis of other methods for numerical integration; see Romberg's method in the next section.
- $r \to \infty$; $\lim_{r\to\infty} R_{2r+2}(a,h,b)f = 0$ can be satisfied only if $f(z)$ is an entire function, such that $|f^{n)}(a)| = o((2\pi/h)^n)$ as $n \to \infty$. Fortunately, this type of convergence is rarely needed in practice. With appropriate choice of b and h, the expansion is typically rapidly semi-convergent. Since the derivatives of are typically more expensive to compute than the values of f, one frequently reduces h (in integration) or increases b (in summation or integration over an infinite interval), and truncates the expansion several terms before one has reached the smallest term that is otherwise the standard procedure with alternating semi-convergent expansion.

Variations of the Euler–Maclaurin summation formula, with *finite differences instead of derivatives* in the expansion, are given in Problem 20, where you also find *a more general form of the formula*, and two more variations of it.

Euler–Maclaurin's formula can also be used for finding an algebraic expression for a finite sum; see Problem 32 or, as in the following example, for finding an expansion that determines the asymptotic behavior of a sequence or a function.

Example 3.4.12. *An expansion that generalizes Stirling's formula.*

We shall use Euler–Maclaurin formula for $f(x) = \ln x$, $a = m > 0$, $h = 1$, $b = n \geq m$. We obtain

$$\hat{T}(m:1:n)f = \sum_{i=m+1}^{n} \ln i - \tfrac{1}{2}\ln n + \tfrac{1}{2}\ln m = \ln(n!) - \tfrac{1}{2}\ln n - \ln(m!) + \tfrac{1}{2}\ln m,$$

$$f^{(2k-1)}(x) = (2k-2)!x^{1-2k}, \qquad \int_m^n f(x)\,dx = n\ln n - n - m\ln m + m.$$

Note that $\hat{T}(m:1:n)f$ and $\int_m^n f(x)\,dx$ are unbounded as $n \to \infty$, but their difference is bounded. Putting these expressions into (3.4.29), and separating the terms containing n from the terms containing m gives

$$\ln(n!) - (n+\tfrac{1}{2})\ln n + n - \sum_{k=1}^{r} \frac{B_{2k}}{2k(2k-1)n^{2k-1}} \tag{3.4.32}$$

$$= \ln(m!) - (m+\tfrac{1}{2})\ln m + m - \sum_{k=1}^{r} \frac{B_{2k}}{2k(2k-1)m^{2k-1}} - R_{2r+2}(m:1:n).$$

By (3.4.31), after a translation of the variable of integration,

$$|R_{2r+2}(m:1:n)| \leq \int_m^n \frac{|2B_{2r+2}|}{(2r+2)x^{2r+2}}\,dx$$

$$\leq \frac{|2B_{2r+2}|}{(2r+2)(2r+1)|m^{2r+1}|} \approx \frac{(2r)!}{\pi|2\pi m|^{2r+1}}. \tag{3.4.33}$$

Now let $n \to \infty$ with fixed r, m. First, note that the integral in the error bound converges. Next, in most texts of calculus Stirling's formula is derived in the following form:

$$n! \sim \sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n} \quad (n \to \infty). \tag{3.4.34}$$

If you take the natural logarithm of this, it follows that the left hand side of (3.4.32)

tends to $\frac{1}{2}\ln(2\pi)$ [58] , and hence

$$\ln(m!) = (m+\tfrac{1}{2})\ln m - m + \tfrac{1}{2}\ln(2\pi) + \sum_{k=1}^{r} \frac{B_{2k}}{2k(2k-1)m^{2k-1}} + R, \qquad (3.4.35)$$

where a bound for R is given by (3.4.33). The numerical values of the coefficients are found in Table 3.4.4.

Almost the same derivation works also for $f(x) = \ln(x+z)$, $m = 0$, where z is a complex number, not on the negative real axis. A few basic facts about the Gamma function are needed; see details in Henrici [26, Sec. 11.11, Example 3].

The result is that *you just replace the integer m by the complex number z in the expansion* (3.4.35). According to [1, Sec. 6.1.42] R is to be multiplied by $K(z) = \text{upper bound}_{u\geq 0}|z^2/(u^2+z^2)|$. For z real and positive, $K(z) = 1$, and since $f'(x) = (z+x)^{-1}$ is completely monotonic, it follows from Theorem 3.4.3 that, *in this case, R is less in absolute value than the first term neglected and has the same sign.*

It is customary to *write* $\ln\Gamma(z+1)$ *instead of* $\ln(z!)$. The gamma function is one of the most important transcendental functions; see, e.g., Handbook [1, Sec. 6.5] and Lebedev[30].

This formula (with $m = z$) is useful for the practical computation of $\ln\Gamma(z+1)$. Its semi-convergence is best if $\Re z$ is large and positive. If this condition is not satisfied, the situation can easily be improved by means of logarithmic forms of the

- *reflection formula*: $\Gamma(z)\Gamma(1-z) = \pi/\sin\pi z$,
- *recurrence formula*: $\Gamma(z+1) = z\Gamma(z)$.

By simple applications of these formulas the computation of $\ln\Gamma(z+1)$ for an arbitrary $z \in \mathbf{C}$ is reduced to the computation of the function for a number z', such that $|z'| \geq 17$, $\Re z' > \frac{1}{2}$, for which the total error, if $r = 5$, becomes typically less than 10^{-14}. See Problem 24.

Although, in this section, the main emphasis is on the application of the Euler–Maclaurin formula to the computation of sums and limits, we shall comment a little on its possibilities for other applications [59].

[58] You may ask why we refer to (3.4.34). Why not? Well, it is not necessary, because it is easy to prove that the left hand side of (3.4.32) increases with n and is bounded; it thus tends to some limit C (say). The proof that $C = \ln\sqrt{2\pi}$ *exactly* is harder, without the Wallis product idea (from 1655) that is probably used in your calculus text, or something equally ingenious or exotic. However, if you compute the right hand side of (3.4.32) for $m = 17$, $r = 5$ (say), and estimate the remainder, you will obtain C to a fabulous guaranteed accuracy, in negligible computer time after a rather short programming time. And you may then replace $\frac{1}{2}\ln 2\pi$ by your own C in (3.4.35), if you like.

[59] As you may have noted, we write "the Euler–Maclaurin formula" mainly for (3.4.29) that is used in general theoretical discussions, or if other applications than the summation of an infinite series are the primary issue. The term "the Euler–Maclaurin summation formula" is mainly used in connection with (3.4.25), i.e. when the summation of an infinite series is the issue. "The Euler–Maclaurin expansion" denotes both the right hand side of (3.4.29), except for the remainder, and for the corresponding terms of (3.4.25). These distinctions are convenient for us, but they are neither important nor in general use.

- It shows that the *global truncation error of the trapezoidal rule* for $\int_a^b f(x)\,dx$ with step size h, *has an expansion into powers of* h^2. Note that although the expansion contains derivatives at the boundary points only, the remainder requires that $|f^{(2r+2)}|$ is integrable in the interval $[a, b]$. The Euler–Maclaurin formula is thus the theoretical basis for the application of *repeated Richardson extrapolation* to the results of the trapezoidal rule, known as *Romberg's method*; see Sec5.4.2. Note that *the validity depends on the differentiability properties of* f.

- The Euler–Maclaurin formula can be used for highly accurate numerical integration when the values of some derivatives of f are known at $x = a$ and $x = b$. More about this in Chapter 5.

- Theorem 3.3.3 shows that the trapezoidal rule is second order accurate, unless $f'(a) = f'(b)$, but there exist *interesting exceptions*. Suppose that the function f is infinitely differentiable for $x \in \mathbf{R}$, and that f *has* $[a, b]$ *as an interval of periodicity*, i.e. $f(x + b - a) = f(x), \forall x \in \mathbf{R}$. Then $f^{(k)}(b) = f^{(k)}(a)$, for $k = 0, 1, 2, \ldots$, hence *every term in the Euler–Maclaurin expansion is zero* for the integral over *the whole period* $[a, b]$. One could be led to believe that the trapezoidal rule gives the exact value of the integral, but this is usually not the case; for most periodic functions f, $\lim_{r\to\infty} R_{2r+2} f \neq 0$; *the expansion converges, of course, though not necessarily to the correct result.*

We shall illuminate these amazing properties of the trapezoidal rule from different points of view in several places in this book, e.g. in Sec. 5.3. See also applications to the so-called bell sums in Problem 30.

3.4.5 Repeated Richardson Extrapolation

Let $F(h)$ denote the value of a certain quantity obtained with step length h. In many calculations one wants to know the limiting value of $F(h)$ as the step length approaches zero. However, the work to compute $F(h)$ often increases sharply as $h \to 0$. In addition, the effects of round-off errors often set a practical bound for how small h can be chosen.

Often, one has some knowledge of how the truncation error $F(h) - F(0)$ behaves when $h \to 0$. If

$$F(h) = a_0 + a_1 h^p + O(h^r), \qquad h \to 0, r > p,$$

where $a_0 = F(0)$ is the quantity we are trying to compute and a_1 is unknown, then a_0 and a_1 can be estimated if we compute F for two step lengths, h and qh, $q > 1$:

$$\begin{aligned} F(h) &= a_0 + a_1 h^p + O(h^r), \\ F(qh) &= a_0 + a_1 (qh)^p + O(h^r), \end{aligned}$$

from which eliminating a_1 we get

$$F(0) = a_0 = F(h) + \frac{F(h) - F(qh)}{q^p - 1} + O(h^r). \tag{3.4.36}$$

This formula is called **Richardson extrapolation**, or the *deferred approach to the limit.*[60] Examples of this were mentioned in Chapter 1—the application of the above process to the trapezoidal rule for numerical integration (where $p = 2$, $q = 2$), and for differential equations—$p = 1$, $q = 2$ for Euler's method, $p = 2$, $q = 2$ for Runge's 2nd order method.

The term $(F(h) - F(qh))/(q^p - 1)$ is called the *Richardson correction.* It is used in in (3.4.36) for improving the result. Sometimes, however, it is used only for estimating the error. This can make sense, e.g., if the values of F are afflicted by other errors, usually irregular, suspected to be comparable in size to the correction. If the irregular errors are negligible, this error estimate is asymptotically correct. More often, the Richardson correction is used as error estimate for the improved (or extrapolated) value $F(h) + (F(h) - F(qh))/(q^p - 1)$, but this is typically a strong overestimate; the error estimate is $O(h^p)$, while the error is $O(h^r)$, $(r > p)$.

Suppose that a more complete expansion of $F(h)$ in powers of h, is known to exist,

$$F(h) = a_0 + a_1 h^{p_1} + a_2 h^{p_2} + a_3 h^{p_3} + \ldots \qquad 0 < p_1 < p_2 < p_3 < \ldots, \tag{3.4.37}$$

where the exponents are typically known, while the coefficients are unknown. Then one can *repeat the use of Richardson extrapolation* in a way described below. This process is, in many numerical problems—especially in the numerical treatment of integral and differential equations—one of the simplest ways to get results which have tolerable truncation errors. The application of this process becomes especially simple when the step lengths form a geometric series $H, H/q, H/q^2, \ldots$, where $q > 1$ and H is the **basic step length**.

Theorem 3.4.4. *Suppose that an expansion of the form of (3.4.37), where* $0 < p_1 < p_2 < p_3 < \ldots$, *holds for* $F(h)$, *and set* $F_1(h) = F(h)$,

$$F_{k+1}(h) = \frac{q^{p_k} F_k(h) - F_k(qh)}{q^{p_k} - 1} = F_k(h) + \frac{F_k(h) - F_k(qh)}{q^{p_k} - 1}, \tag{3.4.38}$$

for $k = 1 : (n-1)$, *where* $q > 1$. *Then* $F_n(h)$ *has an expansion of the form*

$$F_n(h) = a_0 + a_n^{(n)} h^{p_n} + a_{n+1}^{(n)} h^{p_{n+1}} + \ldots; \quad a_\nu^{(n)} = \prod_{k=1}^{n-1} \frac{q^{p_k} - q^{p_\nu}}{q^{p_k} - 1} a_\nu. \tag{3.4.39}$$

Note that $a_\nu^{(n)} = 0$ *for* $\nu < n$.

Proof. Set temporarily $F_k(h) = a_0 + a_1^{(k)} h^{p_1} + a_2^{(k)} h^{p_2} + \ldots + a_\nu^{(k)} h^{p_\nu} + \ldots$. Put this expansion into the first expression on the right hand side of (3.4.38), and,

[60] The idea of a deferred approach to the limit is sometimes used also in the experimental sciences—for example, when some quantity is to be measured in complete vacuum (difficult or expensive to produce). It can then be more practical to measure the quantity for several different values of the pressure. Expansions analogous to equation (3.4.37) can sometimes be motivated by the kinetic theory of gases.

substituting $k+1$ for k, put it into the left hand side. By matching the coefficients for h^{p_ν} we obtain

$$a_\nu^{(k+1)} = a_\nu^{(k)}(q^{p_k} - q^{p_\nu})/(q^{(p_k)} - 1).$$

By (3.4.37), the expansion holds for $k = 1$, with $a_\nu^{(1)} = a_\nu$. The recursion formula then yields the product formula for $a_\nu^{(n)}$. Note that $a_\nu^{(\nu+1)} = 0$, hence $a_\nu^{(n)} = 0, \forall \nu < n$. □

The product formula is for theoretical purpose. The recurrence formula is for practical use. If an expansion of the form of (3.4.37) is known to exist, the above theorem gives a way to compute increasingly better estimates of a_0. The leading term of $F_n(h) - a_0$ is $a_n^{(n)} h^{p_n}$, the exponent of h increases with n. A moment's reflection on equation (3.4.38) will convince the reader that (using the notation of the theorem) $F_{k+1}(h)$ is determined by the $k+1$ values

$$F_1(H), F_1(H/q), \ldots, F_1(H/q^k).$$

With some changes in notation we obtain the following algorithm.

Algorithm 3.4.2 Repeated Richardson extrapolation

For $m = 1 : N$, set $T_{m,1} = F(H/q^{m-1})$, and compute, for $m = 2 : N,\ k = 1 : m-1$,

$$T_{m,k+1} = T_{m,k} + \frac{T_{m,k} - T_{m-1,k}}{q^{p_k} - 1}. \tag{3.4.40}$$

It is sometimes advantageous to write

$$T_{m,k+1} = \frac{q^{p_k} T_{m,k} - T_{m-1,k}}{q^{p_k} - 1}.$$

The computations can be set up in a scheme, where an extrapolated value in the scheme is obtained by using the quantity to its left and the correction diagonally above. (In a computer the results are simply stored in a lower triangular matrix.)

Suppose that TOL is the permissible error. Then, according to the argument above, one continues the process, until two values *in the same row* agree to the desired accuracy, i.e. $|T_{m,k} - T_{m,k-1}| < Tol - CU$, where CU is an upper bound of the irregular error, (see below). (TOL should, of course, be larger than CU.) If no other error estimate is available, $\min_k |T_{m,k} - T_{m,k-1}| + CU$ is usually chosen as error estimate, even though it is typically a strong overestimate.

Typically $k = m$, and T_{mm} is accepted as the numerical result, but this is not always the case. For instance, if H has been chosen so large that the use of the basic asymptotic expansion is doubtful, then the uppermost diagonal of the extrapolation scheme contains nonsense and should be ignored, except for its element in the first column. Such a case is detected by inspection of the difference quotients in a column. If for some k, where $T_{k+2,k}$ has been computed and the modulus of the relative irregular error of $T_{k+2,k} - T_{k+1,k}$ is less than (say) 20%, *and*, most important,

Table 3.4.3. *Scheme for repeated Richardson extrapolation*

	$\frac{\Delta}{q^{p_1}-1}$	$\frac{\Delta}{q^{p_2}-1}$	$\frac{\Delta}{q^{p_3}-1}$	
T_{11}				
T_{21}	T_{22}			
T_{31}	T_{32}	T_{33}		
T_{41}	T_{42}	T_{43}	T_{44}	

the difference quotient $(T_{k+1,k} - T_{k,k})/(T_{k+2,k} - T_{k+1,k})$ is is very different from its theoretical value q^{p_k}, then the uppermost diagonal is to be ignored (except for its first element). In such a case, one says that *H is outside the asymptotic regime*.

In this discussion a bound for the inherited irregular error is needed. We shall now derive such a bound. Fortunately, it turns out that the numerical stability of the Richardson scheme is typically very satisfactory, (although the total error bound for T_{mk} will never be smaller than the largest irregular error in the first column).

Denote by ϵ_1 the the column vector with the irregular errors of the initial data. We neglect the rounding errors committed during the computations. [61] Then the inherited errors satisfy the same linear recursion formula as the $T_{m,k}$, i.e.

$$\epsilon_{m,k+1} = \frac{q^{p_k}\epsilon_{m,k} - \epsilon_{m-1,k}}{q^{p_k} - 1}.$$

Denote the k'th column of errors by ϵ_k, and set $\|\epsilon_k\| = \max_m |\epsilon_{m,k}|$, $\|\epsilon_1\| = U$. Then $\|\epsilon_{k+1}\| \leq \frac{q^{p_k}+1}{q^{p_k}-1}\|\epsilon_k\|$. Hence, for every k, $\|\epsilon_{k+1}\| \leq C\|\epsilon_1\| = CU$, where C is the infinite product

$$C = \prod_{k=1}^{\infty} \frac{q^{p_k}+1}{q^{p_k}-1} = \prod_{k=1}^{\infty} \frac{1+q^{-p_k}}{1-q^{-p_k}}$$

that converges as fast as $\sum q^{-p_k}$; the multiplication of ten factors are thus more than enough for obtaining a sufficiently accurate value of C.

Example 3.4.13.

The most common special case is an expansion where $p_k = 2k$,

$$F(h) = a_0 + a_1h^2 + a_2h^4 + a_3h^6 + \ldots \tag{3.4.41}$$

The headings of the columns of Table 3.4.5 then become $\Delta/3, \Delta/15, \Delta/63, \ldots$. In this case *we find that* $C = \frac{5}{3} \cdot \frac{7}{15} \cdots < 2$ (after less than 10 factors).

[61] They are usually for various reasons of less importance. One can also *make* them smaller, in floating-point computation, by subtracting a suitable constant from all initial data. This is applicable to all linear methods of convergence acceleration.

For (systems of) ordinary differential equations there exist some general theorems, according to which the form of the asymptotic expansion (3.4.37) of the global error can be found.

- For Numerov's method for ordinary differential equations, discussed in Example 3.3.14 and Problem 28, one can show that we have the same exponents in the expansion for the global error, but $a_1 = 0$. (and the first heading disappears). We thus have the same product as above, except that the first factor disappears, *hence* $C < 2 \cdot \frac{3}{5} = 1.2$.
- For *Euler's method* for ordinary differential equations, presented in Sec. 1.4.2, $p_k = k$; the headings are $\Delta/1, \Delta/3, \Delta/7, \Delta/15, \ldots$. *Hence* $C = 3 \cdot \frac{5}{3} \cdot \frac{9}{7} \cdots = 8.25$.
- For *Runge's 2nd order method*, presented in Sec. 1.4.3, the exponents are the same, but $a_1 = 0$ (and the first heading disappears). We thus have the same product as for Euler's method, except that the first factor disappears, *hence* $C = 8.25/3 = 2.75$.

In the special case that $p_j = j \cdot p$, $j = 1, 2, 3, \ldots$ in (3.4.37), i.e. for expansions of the form

$$F(h) = a_0 + a_1 h^p + a_2 h^{2p} + a_3 h^{3p} + \ldots, \tag{3.4.42}$$

it is not necessary that the step sizes form a geometric progression. We can choose any increasing sequence of integers $q_1 = 1, q_2, \ldots, q_k$, set $h_i = H/q_i$, and use an algorithm that looks very similar to repeated Richardson extrapolation. *In cases where both are applicable, i.e. if* $p_k = p \cdot k$, $q_i = q^i$, *they are identical, otherwise they have different areas of application.*

Note that the expansion (3.4.42) is a usual power series in the variable $x = h^p$, which can be approximated by a polynomial in x. Suppose that $k + 1$ values $F(H), F(H/q_2), \ldots, F(H/q_k)$ are known. Then by the corollary to Theorem 3.2.1, a polynomial $Q \in \mathcal{P}_k$ is uniquely determined by the interpolation conditions

$$Q(x_i) = F(H/q_i), \quad x_i = (H/q_i)^p, \quad i = 1 : k.$$

Our problem is to find $Q(0)$. Many interpolation formulas can be used for this extrapolation. *Neville's algorithm*, which is derived in Sec. 4.2, is particularly convenient in this situation. (4.2.27) yields, after a change of notation, the following recursion.

Algorithm 3.4.3 Neville's algorithm
For $m = 1 : N$, set $T_{m,1} = F(H/q_m)$, where $1 = q_1 < q_2 < q_3 \ldots$, is any increasing sequence of integers, and compute, for $m = 2 : N$, $k = 1 : m - 1$,

$$T_{m,k+1} = T_{m,k} + \frac{T_{m,k} - T_{m-1,k}}{(q_m/q_{m-k})^p - 1} = \frac{(q_m/q_{m-k})^p T_{m,k} - T_{m-1,k}}{(q_m/q_{m-k})^p - 1}. \tag{3.4.43}$$

The computations can be set up in a triangle matrix as for repeated Richardson extrapolations, without headings.

Example 3.4.14. *Computation of π by means of regular polygons.*

The ancient Greeks computed approximate values of the circumference of the unit circle, 2π, by inscribing a regular polygon and computing its perimeter. Archimedes considered the inscribed 96-sided regular polygon, whose perimeter is $6.28206 = 2 \cdot 3.14103$. In general, a regular n-sided polygon inscribed (circumscribed) in a circle with radius 1 has circumference $2a$ ($2b$), where

$$a_n = n\sin(\pi/n), \qquad b_n = n\tan(\pi/n).$$

Clearly $a_n < \pi < b_n$, and gives lower and upper bounds for π.

Setting $h = 1/n$, we have

$$a_n = a(h) = \frac{1}{h}\sin \pi h = \pi - \frac{\pi^3}{3!}h^2 + \frac{\pi^5}{5!}h^4 - \frac{\pi^7}{7!}h^6 + \dots,$$
$$b_n = b(h) = \frac{1}{h}\tan \pi h = \pi + \frac{\pi^3}{3}h^2 + \frac{2\pi^5}{15}h^4 + \frac{17\pi^7}{315}h^6 - \dots,$$

so $a(h)$ and $b(h)$ satisfy the assumptions for repeated Richardson extrapolation with $p_k = 2k$.

We now try to find a recursion formula that leads from a_n and b_n to a_{2n} and b_{2n}. Setting $n_m = n_1 \cdot 2^{m-1}$, and

$$s_m = 1/\sin(\pi/n_m), \qquad t_m = 1/\tan(\pi/n_m),$$

we have $a_{n_m} = n_m/s_m$, and $b_{n_m} = n_m/t_m$. From the trigonometric formula $\tan(z/2) = \sin z/(1+\cos z)$, we obtain the recursion

$$t_m = s_{m-1} + t_{m-1}, \qquad s_m = \sqrt{t_m^2 + 1}, \quad m = 1, 2, \dots. \tag{3.4.44}$$

Note that no trigonometric functions are used, only the square root, and this could be computed by Newton's method. Further, there is no cancellation with consequential round-off errors in these formulas.

Taking $n_1 = 6$, gives $a_6 = 6/2 = 3$, and $b_6 = 6/\sqrt{3} = 3.4641\dots$. The following table gives a_{n_m} and b_{n_m} for $n_1 = 6$, $m = 1:5$, computed in IEEE double precision using the recursion (3.4.44).

m	n_m	a_{n_m}	b_{n_m}
1	6	**3**.00000000000000	**3**.46410161513775
2	12	**3.1**0582854123025	**3.2**1539030917347
3	24	**3.1**3262861328124	**3.1**5965994209750
4	48	**3.13**935020304687	**3.14**608621513143
5	96	**3.141**03195089051	**3.14**271459964537

From this we can deduce that $3.1410 < \pi < 3.1427$, or the famous, slightly weaker, rational lower and upper bounds of Archimedes $3\frac{10}{71} < \pi < 3\frac{1}{7}$. A correctly rounded value of π to twenty digits reads 3.14159 26535 89793 23846 and correct digits in the table are shown in boldface.

The following table gives the *Richardson scheme* applied to a_{n_m}, $m = 1 : 5$,

3.14110472164033			
3.14156197063157	**3.14159**245389765		
3.14159073296874	**3.14159265**045789	**3.14159265357**789	
3.14159253350506	**3.14159265354**081	**3.141592653589**75	**3.14159265358979**

The errors in successive columns decay as $4^{-2k}, 4^{-3k}, 4^{-4k}$, and the final number is correct to all 14 decimals shown. Hence the accuracy used in computing values in the previous table, which could be thought excessive, has been put to good use![62]

Repeated Richardson extrapolation applied to the series b_{n_m} gives the following table.

3.13248654051871			
3.14108315307218	**3.1416**5626057574		
3.14156163947608	**3.14159**353856967	**3.141592**54298228	
3.14159072781668	**3.141592**66703939	**3.1415926532**0557	**3.1415926536**3782

The improvement is seen to be not as rapid in this case. Only ten correct digits are obtained. The explanation for this is that the coefficients in the Taylor expansion of $\tan x$ decay at a lower rate than for $\sin x$.

Example 3.4.15. *Application to numerical differentiation.*

By Bickley's table for difference operators, i.e. Table 3.3.1 in Sec. 3.3.2, we know that

$$\frac{\delta}{h} = \frac{2\sinh(hD/2)}{h} = D + a_2h^2D^3 + a_4h^4D^5 + \dots,$$
$$\mu = \cosh(hD/2) = 1 + b_2h^2D^2 + b_4h^4D^4 + \dots,$$

where the values of the coefficients are now unimportant to us. Hence

$$f'(x) - \frac{f(x+h) - f(x-h)}{2h} = Df(x) - \frac{\mu\delta f(x)}{h} \quad \text{and} \quad f''(x) - \frac{\delta^2 f(x)}{h^2}$$

have expansions into *even* powers of h. Repeated Richardson extrapolation can thus be used with step sizes H, $H/2$, $H/4, \dots$ and headings $\Delta/3$, $\Delta/15$, $\Delta/63, \dots$. For numerical examples, see Problems of this section.

Richardson extrapolation can be applied in the same way to the computation of higher derivatives. Because of the division by h^k in the difference approximation of $f^{(k)}$, *irregular errors in the values of* $f(x)$ *are of much greater importance*

[62] An extension of this example was used as a test problem for a package for (in principle) arbitrarily high precision floating point arithmetic in Matlab for a PC. For instance, π was obtained to 203 decimal places with 22 polygons and 21 Richardson extrapolations in less than half a minute. The extrapolations took a small fraction of this time. Nevertheless they increased the number of correct decimals from approximately 15 to 203.

in numerical differentiation than in interpolation and integration. It is therefore important to use high order approximations in numerical differentiation, so that larger values of h can be used.

Suppose that the irregular errors of the values of f are bounded in magnitude by ERB, these errors are propagated to $\mu\delta f(x)$, $\delta^2 f(x)$,... with bounds equal to ERB/h, 4ERB/h^2, ... As mentioned earlier, the Richardson scheme (in the version used here) is no rascal; it multiplies the latter bounds by a factor less than 2.

For applications of repeated Richardson extrapolation to numerical integration; see Sec. 5.4.

Review Questions

1. Describe three procedures for improving the speed of convergence of certain series. Give examples of their use.

2. State the original version of Euler's transformation.

3. (a) What pieces of information appear in the Euler–Maclaurin formula? Give the generating function for the coefficients. What do you know about the remainder term?

(b) Give at least three important uses of the Euler–Maclaurin formula.

Problems and Computer Exercises

1. (a) Compute $\sum_{n=1}^{\infty} \frac{1}{(n+1)^3}$ to eight decimal places by using $\sum_{n=N}^{\infty} \frac{1}{n(n+1)(n+2)}$, for a suitable N, as a comparison series. Estimate roughly how many terms you would have to add without and with the comparison series.

Hint: You find the exact sum of this comparison series in Problem 3.3.2.

(b) Compute the sum also by Euler–Maclaurin's formula or one of its variants in Problem 20(a).

2. Study, or write yourself, programs for some of the following methods: [63]

- iterated Aitken acceleration
- modified iterated Aitken, according to (3.4.8) or an a-version.
- generalized Euler transformation
- one of the central difference variants of Euler–Maclaurin's formula, given in Problem 20(a)

The programs are needed in two slightly different versions.

Version i: For studies of the convergence rate, for a series (sequence) where

[63] We have Matlab in mind, or some other language with complex arithmetic and graphical output.

one knows a sufficiently accurate value *exa* of the sum (the limit). The risk of drowning in figures becomes smaller, if you make graphical output, e.g. like Figure 3.4.1.

Version ii: For a run controlled by a tolerance, like in Algorithm 3.3.1, appropriately modified for the various algorithms. Print also i and, if appropriate, jj. If *exa* is known, it should be subtracted from the result, because it is of interest to compare *errest* with the actual error.

Comment: If you do not know *exa*, find a sufficiently good *exa* by a couple of runs with very small tolerances, before you study the convergence rates (for larger tolerances).

3. The formula for Aitken acceleration is sometimes given in the forms

$$s_n - \frac{(\Delta s_n)^2}{\Delta^2 s_n} \quad \text{or} \quad s_n - \frac{\Delta s_n \nabla s_n}{\Delta s_n - \nabla s_n}.$$

Show that these are equivalent to s'_{n+2} or s'_{n+1}, respectively, in the notations of (3.4.2). Also note that the second formula is $\lim_{p\to\infty} s'_n$ (not s'_{n+1}) in the notation of (3.4.6).

4. (a) Try iterated Aitken with thinning for $\sum_1^\infty e^{-\sqrt{n}}$, according to the suggestions after Example 3.4.3.

(b) Study the effect of small random perturbations to the terms.

5. *Oscillatory series* of the form $\sum_{n=1}^\infty c_n z^n$. Suggested examples:

$$c_n = e^{-\sqrt{n}}, \quad 1/(1+n^2), \quad 1/n, \; 1/(2n-1),$$
$$n/(n^2+n+1) \quad , 1/\sqrt{n}, \quad 1/\ln(n+1),$$

where $z = -1$, -0.9, $e^{i3\pi/4}$, i, $e^{i\pi/4}$, $e^{i\pi/16}$, for the appropriate algorithms mentioned in Problem 2 above. Apply thinning. Try also classical Euler transformation on some of the cases.

Study how the convergence ratio depends on z, and compare with theoretical results. Compare the various methods with each others.

6. *Essentially positive series.* of the form $\sum_{n=1}^\infty c_n z^n$, where

$$c_n = e^{-\sqrt{n}}, \quad 1/(1+n^2), \quad 1/(5+2n+n^2)), \quad (n\cdot\ln(n+1))^{-2},$$
$$1/\sqrt{n^3+n}, n^{-4/3}, \quad 1/((n+1)(\ln(n+1))^2);$$

$z = 1$, 0.99, 0.9, 0.7, $e^{i\pi/16}$, $e^{i\pi/4}$, i. Use appropriate algorithms from Problem 2.

Try also Euler–Maclaurin's summation formula, or one of its variants, if you can handle the integral with good accuracy. Also try to find a good comparison series; it is not always possible.

Study the convergence rate. Try also *thinning* to the first two methods.

7. *Divergent series.* Apply, if possible, Aitken acceleration and the generalized Euler transformation to the following divergent series $\sum_1^\infty c_n z^n$. Compare the numerical results with the results obtained by analytic continuation, using the

analytic expression for the sum as a function of z.

(a) $c_n = 1$, $z = -1$; (b) $c_n = n$, $z = -1$;

(c) c_n is an arbitrary polynomial in n; (d) $c_n = 1$, $z = i$;

(e) $c_n = 1$, $z = 2$; (f) $c_n = 1$, $z = -2$.

8. Let y_n be the Fibonacci sequence defined, in Problem 3.3.16 by the recurrence relation,

$$y_n = y_{n-1} + y_{n-2}, \quad y_0 = 0, \quad y_1 = 1.$$

Show that the sequence $\{y_{n+1}/y_n\}_0^\infty$ satisfies the sufficient condition for Aitken acceleration, given in the text. Compute a few terms, compute the limit by Aitken acceleration(s), and compare with the exact result.

9. When the current through a galvanometer changes suddenly, its indicator begins to oscillate toward a new stationary value s. The relation between the successive turning points v_0, v_1, v_2, ...is $v_n - s \approx A \cdot (-k)^n$, $0 < k < 1$. Determine from the following series of measurements, Aitken extrapolated values v_2', v_3', v_4' which are all approximations to s:

$$v_0 = 659, \quad v_1 = 236, \quad v_2 = 463, \quad v_3 = 340, \quad v_4 = 406.$$

10. (a) Show that the a-version of Aitken acceleration can be *iterated*, for $i = 0 : N-2$,

$$a_{i+1}^{(i+1)} = 0, \quad a_j^{(i+1)} = a_j^{(i)} - \nabla\left((a_j^{(i)})^2/\nabla a_j^{(i)}\right), \quad j = i+2 : N,$$
$$s_N^{(i+1)} = s_N^{(i)} - (a_N^{(i)})^2/\nabla a_N^{(i)}.$$

(Note that $a_j^{(0)} = a_j$, $s_j^{(0)} = s_j$.) We thus obtain N estimates of the sum s. We cannot be sure that the last estimate $s_N^{(N-1)}$ is the best, due to irregular errors in the terms and during the computations. Accept instead, e.g., the average of a few estimates that are close to each other, or do you have a better suggestion? This also gives you a (not quite reliable) error estimate.

(b) Although we may expect that the a-version of Aitken acceleration handles rounding errors better than the s-version, the rounding errors may set a limit for the accuracy of the result. It is easy to combine *thinning* with this version. How?

(c) Study or write yourself a program for the a-version, and apply it on one or two problems, where you have used the s-version earlier. Also use thinning on a problem, where it is needed. We have here considered N as given. Can you suggest a better termination criterion, or a process for continuing the computation, if the accuracy obtained is disappointing?

11. A function $g(t)$ has the form

$$g(t) = c - kt + \sum_{n=1}^{\infty} a_n e^{-\lambda_n t},$$

where c, k, a_n and $0 < \lambda_1 < \lambda_2 < \ldots < \lambda_n$ are unknown constants and $g(t)$ is known numerically for $t_\nu = \nu h$, $\nu = 0, 1, 2, 3, 4$.

Find out how to eliminate c, in such a way that a sufficient condition for estimating kh by Aitken acceleration is satisfied. Apply this to the following data, where $h = 0.1$, $g_\nu = g(t_\nu)$.

$$g_0 = 2.14789, \quad g_1 = 1.82207, \quad g_2 = 1.59763, \quad g_3 = 1.40680, \quad g_4 = 1.22784.$$

Then, estimate also c.

12. Suppose that the sequence $\{s_n\}$ satisfies the condition $s_n - s = c_0 n^{-p} + c_1 n^{-p-1} + O(n^{-p-2})$, $p > 0$, $n \to \infty$, and set

$$s'_n = s_n - \frac{p+1}{p} \frac{\Delta s_n \nabla s_n}{\Delta s_n - \nabla s_n},$$

It was stated without proof in Sec. 3.3.2 that $s'_n - s = O(n^{-p-2})$.

(a) Design an a-version of this modified Aitken acceleration, or look up in [3].

(b) Since the difference expressions are symmetrical about n one can conjecture that this result would follow from a continuous analogue with derivatives instead of differences. It has been shown [3] that this conjecture is true, but we shall not prove that. Our (easier) problem is just the continuous analogue: suppose that a function $s(t)$ satisfies the condition $s(t) - s = c_0 t^{-p} + c_1 t^{-p-1} + O(t^{-p-2})$, $p > 0$, $t \to \infty$, and set

$$y(t) = s(t) - \frac{p+1}{p} \frac{s'(t)^2}{s''(t)}.$$

Show that $y(t) - s = O(t^{-p-2})$. Formulate and prove the continuous analogue to (3.4.9).

13. (a) Consider as in Example 3.4.5, the sum $\sum n^{-3/2}$. Show that the partial sum s_n has an asymptotic expansion of the form needed in that example, with $p = -1/2$.

Hint: Apply Euler–Maclaurin's formula (theoretically).

(b) Suppose that $\sum a_n$ is convergent, and that $a_n = a(n)$. $a(z)$ is analytic function at $z = \infty$ (for example a rational function), multiplied by some power of $z - c$. Show that such a function has an expansion like (3.4.7), and that the same holds for a product of such functions.

14. *Rewriting a Fourier series for convergence acceleration.*

Consider a *real* function with the Fourier expansion $F(\phi) = \sum_{n=-\infty}^{\infty} c_n e^{in\phi}$.

(a) Show that

$$F(\phi) = c_0 + 2\Re \sum_{n=1}^{\infty} c_n z^n, \quad z = e^{i\phi}.$$

Hint: Show that $c_{-n} = \bar{c}_n$.

(b) Set $c_n = a_n - ib_n$, where a_n, b_n are real. Show that

$$\sum_{n=0}^{\infty} (a_n \cos n\phi + b_n \sin n\phi) = \Re \sum_{n=0}^{\infty} c_n z^n.$$

(c) How would you rewrite the Chebyshev series $\sum_{n=0}^{\infty} T_n(x)/(1+n^2)$?
(d) Consider also how to handle a complex function $F(\phi)$.

15. Compute and plot

$$F(x) = \sum_{n=0}^{\infty} T_n(x)/(1+n^2), \quad x \in [-1,1].$$

Find out experimentally or theoretically how $F'(x)$ behaves near $x = 1$ and $x = -1$.

16. Compute to (say) 6 decimal places the double sum

$$S = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} \frac{(-1)^{m+n}}{(m^2+n^2)} = \sum_{n=1}^{\infty}(-1)^m f(m),$$

where

$$f(m) = \sum_{n=1}^{\infty}(-1)^n (m^2+n^2)^{-1}.$$

Compute, to begin with, $f(m)$ for $m = 1:10$, by the generalized Euler transformation. Do you need more values of $f(m)$?

Comment: There exists an explicit formula for $f(m)$ in this case, but you can solve this problem easily without using that.

17. We use the notation of Sec. 3.4.3 (the generalized Euler transformation). Assume that $N \geq k \geq 1$, and set $n = N - k + 1$. A sum is equal to zero, if the upper index is smaller than the lower index.

(a) Prove (3.4.22) that was given without proof in the text, i.e.

$$M_{N,k-1} - M_{N-1,k-1} = z^n P^{k-2} u_{n+1}, \quad (k \geq 2).$$

Hint: By subscript transformations in the definition of $M_{N,k}$, prove that

$$M_{N,k-1} - M_{N-1,k-1} = u_{n+1}z^n + \frac{z^n}{1-z}\sum_{s=0}^{k-3}(zE-1)P^s u_{n+1}.$$

Next, show that $zE-1 = (1-z)(P-1)$, and use this to simplify the expression.

(b) Derive the formulas

$$M_{k-1,k} = \frac{1}{1-z}\sum_{s=0}^{k-2} P^s u_1; \qquad M_{N,k} = M_{k-1,k} + \sum_{j=0}^{n-1} z^j P^{k-1} u_{j+1}.$$

Comment: The first formula gives the partial sums of the classical Euler transformation. The second formula relates the k'th column to the partial sums of the power series with the coefficients $P^{k-1}u_{j+1}$.

18. (a) If $u_j = a^j$, $z = e^{i\phi}$, $\phi \in [0, \pi]$, for which real values of $a \in [0, 1]$ does the series on the right of (3.4.15) converge faster than the series on the left?

(b) Find how the classical Euler transformation works if applied to the series

$$\sum z^n, \quad |z| = 1, \quad z \neq 1.$$

Compare how it works on $\sum u_n z^n$, for $u_n = a^n$, $z = z_1$, and for $u_n = 1$, $z = az_1$.

Consider similar questions for other convergence acceleration methods, that are primarily invented for oscillating sequences.

19. Compute $\sum_{k=1}^{\infty} k^{1/2}/(k^2+1)$ with an error of less than 10^{-6}. Sum the first ten terms directly. Then expand the summand in negative powers of k and use Euler–Maclaurin's summation formula. Or try a central difference variant of Euler–Maclaurin's summation formula given in the next problem; then you do not have to compute derivatives.

20. *Variations on the Euler–Maclaurin Theme*

Set $x_i = a + ih$, also for non-integer subscripts, and $x_n = b$.

Two variants with central differences instead of derivatives are interesting alternatives, if the derivatives needed in the Euler–Maclaurin Formula are hard to compute. Check a few of the coefficients on the right hand side of the formula

$$\sum_{j=1}^{\infty} \frac{B_{2j}(hD)^{2j-1}}{(2j)!} \approx \frac{\mu\delta}{12} - \frac{11\mu\delta^3}{720} + \frac{191\mu\delta^5}{60480} - \frac{2497\mu\delta^7}{3628800} + \dots. \tag{3.4.45}$$

Use the expansion for computing the sum given in the previous problem. This formula is given by Fröberg [19, p. 220], who attributes it to Gauss.

Compare the size of its coefficients with the corresponding coefficients of the Euler–Maclaurin Formula.

Suppose that $h = 1$, and that the terms of the given series can be evaluated also for non-integer arguments. Then another variant is to compute the central differences for (say) $h = 1/2$ in order to approximate *each* derivative needed more accurately by means of (3.3.50). This leads to the formula[64]

$$\sum_{j=1}^{\infty} \frac{B_{2j}D^{2j-1}}{(2j)!} \sim \frac{\mu\delta}{6} - \frac{7\mu\delta^3}{180} + \frac{71\mu\delta^5}{7560} - \frac{521\mu\delta^7}{226800} + \cdots. \tag{3.4.46}$$

($h = 1/2$ for the central differences; $h = 1$ in the series.) Convince yourself of the reliability of the formula, either by deriving it or by testing it for (say) $f(x) = e^{0.1\,x}$.

Show that the rounding errors of the function values cause almost no trouble in the numerical evaluation of these difference corrections.

[64] The formula is probably very old, but we have not found it in the literature.

21. (a) Derive formally in a similar way the following formula for an *alternating series*. Set x_i, $h = 1$, $b = \infty$, and assume that $\lim_{x\to\infty} f(x) = 0$.

$$\sum_{i=0}^{\infty}(-1)^i f(a+i) = \tfrac{1}{2}f(a) - \frac{1}{4}f'(a) + \frac{1}{48}f'''(a) - \cdots - \frac{(2^{2r}-1)B_{2r}}{(2j)!}f^{(2r-1)}(a) - \ldots. \tag{3.4.47}$$

Of course, the integral of f is not needed in this case[65] Compare it with some of the other methods for alternating series on an example of your own choice.

(b) Derive, e.g. by operators (without the remainder R), the following more general form of the Euler–Maclaurin Formula ([1, Formula 23.1.32]).

$$\sum_{k=0}^{m-1} hf(a+kh+\omega h) = \int_a^b f(t)dt + \sum_{j=1}^{p}\frac{h^j}{j!}B_j(\omega)(f^{(j-1)}(b) - f^{(j-1)}(a)) + R,$$

$$R = -\frac{h^p}{p!}\int_0^1 \hat{B}_p(\omega - t)\sum_{k=0}^{m-1} f^{(p)}(a+kh+th)\,dt.$$

If you use this formula for deriving the midpoint variant in (c), you will find a quite different expression for the coefficients; nevertheless it is the same formula. Tell how this is explained by Formula 23.1.10 in Handbook [1], i.e. by the "Multiplication Theorem"[66]

$$B_n(mx) = m^{n-1}\sum_{k=0}^{m-1} B_n(x + k/m), \quad n = 0, 1, 2, \ldots, \quad m = 1, 2, 3, \ldots$$

22. Prove statement (b) of the Lemma 3.4.2. (concerning the periodicity and the regularity of the Bernoulli functions).

23. Euler's constant is defined by $\gamma = \lim_{N\to\infty} F(N)$, where

$$F(N) = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N-1} + \frac{1}{2N} - \ln N.$$

(a) Use the Euler–Maclaurin formula with $f(x) = x^{-1}$, $h = 1$, to show that, for any integer N

$$\gamma = F(N) + \frac{1}{12}N^{-2} - \frac{6}{720}N^{-4} + \frac{120}{30240}N^{-6} - \cdots,$$

where every other partial sum is larger than γ, and every other is smaller.

(b) Compute γ to seven decimal places, using $N = 10$, $\sum_{n=1}^{10} n^{-1} = 2.92896825$, $\ln 10 = 2.30258509$.

[65] Note that the right hand side yields a finite value if f is a constant or, more generally, if f is a polynomial, although the series on the left hand side diverges. The same happens to other summation methods; see comments in the last example of Sec. 3.3.2.

[66] That formula and the remainder R are derived in Nörlund [33], p. 21 and p. 30, respectively.

(c) Show how repeated Richardson extrapolation can be used to compute γ from the following values:

N	1	2	4	8
$F(N)$	0.5	0.55685	0.57204	0.57592

(d) Extend (c) to a computation, where a larger number of values of $F(N)$ have been computed as accurately as possible, and so that the final accuracy of γ is limited by the effects of rounding errors. Check the result by looking up in an accurate table of mathematical constants, e.g., in [1].

(e) Set

$$S(r) = \sum_{m=1}^{r} \sum_{n=1}^{r} (m^2 + n^2)^{-1}.$$

By a continuous analog, with a double integral instead of a double sum, you may conjecture that $S(R) \sim a \ln R + b$ as $R \to \infty$. You may even suggest a value of the parameter a. Investigate the conjecture, by computing $S(R)$ for a suitable sequence of values of R. If you find support for it, try to estimate a and b.

24. *A digression about the Gamma function.*

(a) The Handbook [1, 6.1.40] gives an expansion for $\ln \Gamma(z)$ that agrees with formula (3.4.35) for $\ln z!$ (if we substitute z for m), except that the handbook writes $(z - \frac{1}{2}) \ln z$, where we have $(m + \frac{1}{2}) \ln m$. Explain concisely and completely that there is no contradiction here.

(b) An asymptotic expansion for computing $\ln \Gamma(z+1)$, $z \in \mathbf{C}$ is derived in Example 3.4.12. If r terms are used in the asymptotic expansion, the remainder reads:

$$K(z) \frac{(2r)!}{\pi |2\pi z|^{2r+1}} \quad K(z) = \sup_{u \geq 0} \frac{|z^2|}{|u^2 + z^2|}.$$

Set $z = x + iy$. Show the following more useful bound for $K(z)$, valid for $x > 0$,

$$K(z) \leq \begin{cases} 1, & \text{if } x \geq |y|; \\ \frac{1}{2}(x/|y| + |y|/x), & \text{otherwise.} \end{cases}$$

Find a uniform upper bound for the remainder if $r = 5,\ x \geq \frac{1}{2},\ |z| \geq 17$.

(c) Write a program, e.g., in Matlab, for the computation of $\ln \Gamma(z+1)$. Use the reflection and recurrence formulas to transform the input value z, to another $z = x + iy$ that satisfies $x \geq \frac{1}{2},\ |z| \geq 17$, for which this asymptotic expansion is to be used with $r = 5$.

Test the program, e.g., by computing the following quantities, and compare with their exact values, e.g.,

$$n!, \quad \Gamma(n + 1/2)/\sqrt{\pi}, \quad n = 0, 1, 2, 3, 10, 20.$$

$$\left|\Gamma(\tfrac{1}{2} + iy)\right|^2 = \frac{\pi}{\cosh(\pi y)}, \quad y = \pm 10, \pm 20.$$

If the original input value has a small modulus, there is some cancellation, when when the output from the asymptotic expansion is transformed to $\ln(1 + z_{input})$, resulting in a loss of (say) 1 or 2 decimal digits.

(d) It is often much better to work with $\ln\Gamma(z)$ than with $\Gamma(z)$. For example, one can avoid exponent overflow in the calculation of a binomial coefficient or a value of the beta function, $B(z, w) = \Gamma(z)\Gamma(w)/\Gamma(z + w)$, where (say) the denominator can become too big, even if the final result is of a normal order of magnitude.

Another context where the logarithms are much preferable is in connection with interpolation, numerical differentiation etc.; for $|z| \gg 1$ $\ln\Gamma(z)$ is locally approximated by a polynomial much better than $\Gamma(z)$. The following is an example (for a hand held calculator).

Given $10! = 3628800$; compute $\Gamma(x)$ for $x = 11 : 15$. Compute $\Gamma'(13)$ by using either repeated Richardson extrapolation or the central difference expansion, in two ways:

- Use the values of $\ln\Gamma(x)$, (and multiply the logarithmic derivative by $\Gamma(13)$).
- Use directly the values of $\Gamma(x)$.

The first alternative requires a few more operations. Were they worthwhile?

25. (a) Show that

$$\binom{2n}{n} \sim \frac{2^{2n}}{\sqrt{\pi n}}, \quad n \to \infty,$$

and give an asymptotic estimate of the relative error of this approximation. Check the approximation as well as the error estimate for $n = 5$ and $n = 10$.

(b) *Random errors in a difference scheme.* We know from Example 3.3.3 that if the items y_j of a difference scheme are afflicted with errors less than ϵ in absolute value, then the inherited error of $\Delta^n y_j$ is at most $2^n\epsilon$ in absolute value. If we consider the errors as independent random variables, uniformly distributed in the interval $[-\epsilon, \epsilon]$, show that the error of $\Delta^n y_j$ has the variance $\binom{2n}{n}\frac{1}{3}\epsilon^2$, hence the standard deviation is approximately $2^n\epsilon(9\pi n)^{-1/4}$, if $n \gg 1$. Check the result on a particular case by a Monte Carlo study.

Hint: It is known from Probability theory that the variance of $\sum_{j=0}^{n} a_j\epsilon_j$ is equal to $\sigma^2\sum_{j=0}^{n} a_j^2$, and that a random variable, uniformly distributed in the interval $[-\epsilon, \epsilon]$, has the variance $\sigma^2 = \epsilon^2/3$. Finally use (3.1.20) with $p = q = n$.

26. (a) The following table of values of a function $f(x)$ is given:

x	0.6	0.8	0.9	1.0	1.1	1.2	1.4
$f(x)$	1.820365	1.501258	1.327313	1.143957	0.951849	0.752084	0.335920

Compute using repeated Richardson extrapolation $f'(1.0)$ and $f''(1.0)$.

27. Compute an approximation to π using Richardson extrapolation with *Neville's algorithm*, based on three simple polygons, with $n = 2, 3$ and 6 sides, not in geometric progression. A 2-sided polygon can be interpreted as a diameter described up and down. Its "circumference" is thus equal to 4. Show that this gives even a little better than the result (3.14103) obtained for the 96-sided polygon without extrapolations.

28. *Numerov's method with Richardson extrapolations*[67]
(a) Show that the formula

$$h^{-2}(y_{n+1} - 2y_n + y_{n-1}) = y''_n + a(y''_{n+1} - 2y''_n + y''_{n-1})$$

is exact for polynomials of as high degree as possible, if $a = 1/12$. Show that the error has an expansion into *even* powers of h, and determine the first (typically non-vanishing) term of this expansion.

(b) This formula can be applied to the differential equation, $y'' = p(x)y$, with given initial values $y(0)$, $y'(0)$. Show that this yields the recurrence relation

$$y_{n+1} = \frac{(2 + \frac{10}{12}p_n h^2)y_n - (1 - \frac{1}{12}p_{n-1}h^2)y_{n-1}}{1 - \frac{1}{12}p_{n+1}h^2}.$$

Comment: If h is small, information about $p(t)$ is lost by outshifting in the factors $1 - \frac{1}{12}p_{n-1}h^2$ etc. (It is possible to rewrite the formulas in order to reduce the loss of information.) In the application below this causes no trouble with the step sizes suggested, in IEEE double precision. If you must use single precision, however, the outshifting may set a limit to the accuracy in the repeated Richardson extrapolation.

(c) Apply this method, together with two Richardson extrapolations in (d), to the problem of Example 3.1.1, i.e. $y'' = -xy$ with initial values $y(0) = 1$, $y'(0) = 0$, this time over the interval $0 \le x \le 4.8$. Denote the numerical solution by $y(x; h)$, i.e. $y_n = y(x_n; h)$.
Compute the seeds $y_1 = y(h, h)$ by the Taylor expansion in Example 3.1.1. The error of $y(0.2, 0, 2$ should be less than 10^{-10}, since we expect that the (global) errors after two Richardson extrapolations can be of that order of magnitude.
Compute $y(x; h)$, $x = 0 : h : 4.8$, for $h = 0.05$, $h = 0.1$, $h = 0.2$. Store these data in a 100×3 matrix (where you must put zeros into some places). Plot $y(x; 0.05)$ versus x for $x = 0 : 0.05 : 4.8$.

(d) You proved in (a) that the *local* error has an expansion containing *even powers of* h only. It can be shown that *the same is true for the global error* too. Assume (without proof) that

$$y(x, h) = y(x) + c_1(x)h^4 + c_2(x)h^6 + c_3(x)h^8 + O(h^{10}).$$

Perform the adequate repeated Richardson extrapolations to your stored results. Make semi-logarithmic plots of (the modulus of) the 4th order Richardson corrections for $x = 0 : 0.1 : 4.8$, obtained by means of $y(x; 0.05)$ and

[67] See also Example 3.3.14.

$y(x; 0.1)$. Plot in the same fashion the 6th order corrections for $x = 0 : 0.2 : 4.8$, obtained in the second Richardson extrapolation. The 6th order corrections are used as error estimates for the results from both these Richardson extrapolations.[68]

(e) Express, e.g., by the aid of Handbook [1, Sec. 10.4], the solution of this initial value problem in terms of Airy functions[69]

$$y(x) = \frac{\mathrm{Ai}(-x) + \mathrm{Bi}(-x)/\sqrt{3}}{2 \cdot 0.3550280539}.$$

Check a few of your results of the repeated Richardson extrapolation by means of Table 10.11 in the Handbook that, unfortunately, gives only 8 decimal places.

Comment: Your results should be more accurate than that. If they are not, the reason can be that the rounding errors have a large influence, but that is not the most probable reason in this case, if IEEE double precision is used. Experience shows that it is hard to avoid programming blunders in this problem. So do not consider the theory or the rounding errors as the primary suspects. Programming errors do not always yield results that are obviously crazy; sometimes the results look reasonable, although the accuracy is much lower than it should be.

29. (a) Determine the Bernoulli polynomials $B_2(x)$ and $B_3(x)$, and find the values and the derivatives at 0 and 1. Factorize the polynomial $B_3(x)$. Draw the graphs of a few periods of $\hat{B}_i(x)$, $i = 1, 2, 3..$

(b) In an old "Cours d'Analyse", we found a "symbolic" formula, essentially

$$h \sum_{j=0}^{n-1} g'(a + jh) = g(b + hB) - g(a + hB). \tag{3.4.48}$$

The expansion of the right hand side into powers of hB, has been followed by the replacement of the powers of B by Bernoulli numbers, the resulting expansion is not necessarily convergent, even if the first power series converges for any complex value of hB.

Show that the second expansion is equivalent to the Euler–Maclaurin formula, and that it is to be interpreted according to Theorem 3.4.3.

(c) If g is a polynomial, the expansion is finite. Show the following important formulas, and check them with known results for $k = 1 : 3$.

$$\sum_{j=0}^{n-1} j^{k-1} = \frac{(B + n)^k - B^k}{k} = \frac{B_k(n) - B_k}{k}. \tag{3.4.49}$$

[68] Although the 6th order correction yields an 8th order accurate result, it is hard to obtain an error estimate of that order without extra assumptions or extra computation.

[69] Airy functions are special functions (related to Bessel functions) with many applications to Mathematical Physics, e.g., the theory of diffraction of radio waves around the earth's surface.

Also find that (3.4.48) makes sense for $g(x) = e^{\alpha x}$, with the "symbolic" interpretation of the power series for e^{Bx}, if you accept the formula $e^{(B+\alpha)x} = e^{Bx}e^{\alpha x}$.

30. We have called $\sum a_n$ a *bell sum* if a_n as a function of n has a bell-shaped graph, and you must add many terms to get the desired accuracy. Under certain conditions you can get an accurate result by adding (say) every tenth term, and multiply this sum by 10, because both sums can be interpreted as trapezoidal approximations to the same integral, with different step size. Inspired by Euler–Maclaurin's formula, we may hope to be able to obtain high accuracy using an integer stepsize h that is (say) one quarter of the half-width of "the bell". In other words, we do not have to compute and add more than every hth term.

We shall study a class of series

$$S(t) = \sum_{n=0}^{\infty} c_n t^n/n!, \quad t \gg 1, \tag{3.4.50}$$

where $c_n > 0$, $\log c_n$ is rather slowly varying for n large; (say that) $\Delta^p \log c_n = O(n^{-p})$. Let $c(\cdot)$ be a smooth function such that $c(n) = c_n$. We consider $S(t)$ as an approximation to the integral

$$\int_0^{\infty} c(n)t^n/\Gamma(n+1)dn,$$

with a smooth and bell shaped integrand, almost like the normal frequency function, with standard deviation $\sigma \approx k\sqrt{t}$. .

(a) For $p = 1:5$, $t = 4^p$, plot $y = \sqrt{2\pi t}e^{-t}t^n/n!$ versus $x = n/t$, $0 \le x \le 3$; all 5 curves on the same picture.

(b) For $p = 1:5$, $t = 4^p$, plot $y = \ln(e^{-t}t^n/n!)$ versus $x = (n-t)/\sqrt{t}$, $\max(0, t - 8\sqrt{t}) \le n \le t + 8\sqrt{t}$; all 5 curves on the same picture. Give bounds for the error committed if you neglect the terms of the series $e^{-t}\sum_0^{\infty} t^n/n!$, which are cut out in your picture.

(c) With the same notation as in (b), use Stirling's asymptotic expansion to show theoretically that

$$\frac{e^{-t}t^n}{n!} = \frac{e^{-x^2/2}\big(1 + O(1/\sqrt{t})\big)}{\sqrt{2\pi t}}, \tag{3.4.51}$$

for $t \to \infty$, where the $O(1/\sqrt{t})$-term depends on x. Compare this with the plots.

Comment: If you are familiar with Probability, you recognize that this is related to the normal approximation to the Poisson distribution. It is well known that the mean is t, and the standard deviation is $\sqrt{t}$.

If you are familiar with Mathematical Physics, you see the resemblance to the *saddle point method*, if you interpret the sum of terms like the left hand side

(from $n = 0$ to ∞) as an approximation to an integral with stepsize $\Delta n = 1$, i.e.

$$e^{-t} \int_0^\infty t^n/\Gamma(n+1) dn \sim \int_{-\infty}^\infty \exp(-x^2/2)/\sqrt{2\pi} dx = 1, \quad (t \to \infty).$$

(Note that $dx = dn/\sqrt{t}$.) A crude approximation for (3.4.50) is $S(t) \approx c(t)e^t$. We aim, however, at higher accuracy than is common when these approximations are used in Probability and Mathematical Physics, We think, for example, of situations where the result is to be used in a calculation where cancellation causes many digits to be lost and a decent relative accuracy is needed in what will be left.

(d) Test these ideas by making numerical experiments with the series

$$e^{-t} \sum_{n \in \mathcal{N}} t^n/n!,$$

where $\mathcal{N} = \{\text{round}(t - 8\sqrt{t}) : h : \text{round}(t + 8\sqrt{t})\}$, for some integers h in the neighborhood of suitable fractions of $\sqrt{t}$, inspired by the outcome of the experiments. Do this for t =1000, 500, 200, 100, 50, 30. Compare with the exact result, and see how the trapezoidal error depends on h, and try to formulate an error estimate that can be reasonably reliable, in cases where the answer is not known. How large must t be, in order that it should be permissible to choose $h > 1$ if you want (say) 6 correct decimals?

(e) Compute, with an error estimate, $e^{-t} \sum_{n=1}^\infty t^n/(n \cdot n!)$, with 6 correct decimals for the values of t mentioned in (d). You can also check your result with tables and formulas in the Handbook [1, Ch. 5].

31. If you have a good program for generating primes, denote the nth prime by p_n, and try convergence acceleration to series like

$$\sum \frac{(-1)^n}{p_n}, \quad \sum \frac{1}{p_n^2},$$

or what have you? Due to the irregularity of the sequence of primes, you cannot expect the spectacular accuracy of the previous examples, but it can be fun to see how these methods work, e.g., in combination with some comparison series derived from asymptotic results about primes. The simplest one reads $p_n \sim n \ln n$, $(n \to \infty)$, which is equivalent to the classical prime number theorem.

32. *A summation formula based on the Euler numbers*

(a) The Euler numbers E_n were introduced by (3.1.19). The first values read $E_0 = 1$, $E_2 = -1$, $E_4 = 5$, $E_6 = -61$. They are all integers (Problem 3.1.7c). $E_n = 0$ for odd n, and the sign is alternating for even n. Their generating function reads

$$\frac{1}{\cosh z} = \sum_{j=0}^\infty \frac{E_j z^j}{j!}.$$

(a) Show, e.g., by means of operators the following expansion

$$\sum_{k=m}^{\infty}(-1)^{k-m}f(k) \approx \sum_{p=0}^{q}\frac{E_{2p}f^{(2p)}(m-\frac{1}{2})}{2^{2p+1}(2p)!} \tag{3.4.52}$$

Comment: No discussion of convergence etc. is needed; the expansion behaves much like the Euler–Maclaurin expansion, and so does the error estimation; see, e.g., [16].
The coefficient of $f^{(2p)}(m-\frac{1}{2})$ is approximately $2(-1)^p/\pi^{2p+1}$ when $p \gg 1$, e.g., for $p=3$ the approximation yields $-6.622\cdot 10^{-4}$, while the exact coefficient is $61/92160 \approx 6.619\cdot 10^{-4}$.
(b) Apply (3.4.52) for explaining the following curious observation, reported by Borwein et al. [6].

$$\sum_{k=1}^{50}\frac{4(-1)^k}{2k-1} = 3.12159465259\ldots$$
$$(\pi = 3.14159265359\ldots).$$

Note that only three digits disagree. There are several variations on this theme. Borwein et al. actually displayed the case with 40 decimal places based on 50,000 terms. Make "an educated guess" concerning how few digits disagreed.

3.5 Continued Fractions and Padé Approximants

3.5.1 Continued Fractions

Some functions cannot be well approximated by a power series, but can well be approximated by a quotient of power series. In order to study such approximations we first introduce algebraic **continued fractions**. Let r be a number and set

$$r = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3+}}}\ldots = b_0 + \frac{a_1}{b_1+}\,\frac{a_2}{b_2+}\,\frac{a_3}{b_3+}\ldots, \tag{3.5.1}$$

where the second expression is a convenient compact notation. If the number of terms is infinite, r is called an *infinite continued fraction*. The terminating fraction

$$r_n = \frac{p_n}{q_n} = b_0 + \frac{a_1}{b_1+}\,\frac{a_2}{b_2+}\cdots\frac{a_n}{b_n} \tag{3.5.2}$$

is called *the nth approximant* of the continued fraction. This can be evaluated *backwards* in n divisions using the recurrence: Set $r = y_0$, where

$$y_n = b_n, \qquad y_{i-1} = b_{i-1} + a_i/y_i, \quad i = n:-1:1, \tag{3.5.3}$$

It can happen that in an intermediate step the denominator y_i becomes zero and $y_{i-1} = \infty$. This does no harm if you proceed in the next step when you divide by y_{i-1} the result is set equal to 0. If it happens in the last step, the result is ∞.[70]

A drawback of evaluating an infinite continued fraction expansion by the backwards recursion (3.5.3) is that you have decide where to stop in advance. The following theorem shows how *forwards* (or top down) evaluation can be achieved.

Theorem 3.5.1.

Consider the continued fraction (3.5.1). *For* $n \geq 1$, $r_n = p_n/q_n$, *where* p_n, q_n *satisfies the* **recursion formula**

$$p_n = b_n p_{n-1} + a_n p_{n-2}, \quad p_{-1} = 1, \quad p_0 = b_0, \tag{3.5.4}$$

$$q_n = b_n q_{n-1} + a_n q_{n-2}, \quad q_{-1} = 0, \quad q_0 = 1. \tag{3.5.5}$$

Another useful formula reads

$$p_n q_{n-1} - p_{n-1} q_n = (-1)^{n-1} a_1 a_2 \cdots a_n. \tag{3.5.6}$$

Proof. We prove the recursion formulas by induction. First, for $n = 1$, we obtain

$$\frac{p_1}{q_1} = \frac{b_1 p_0 + a_1 p_{-1}}{b_1 q_0 + a_1 q_{-1}} = \frac{b_1 b_0 + a_1}{b_1 + 0} = b_0 + \frac{a_1}{b_1} = r_1.$$

Next, assume that the formulas are valid up to p_{n-1}, q_{n-1}, for *every* continued fraction. Note that p_n/q_n can be obtained from p_{n-1}/q_{n-1}, by the substitution of $b_{n-1} + a_n/b_n$ for b_{n-1}. Hence

$$\frac{p_n}{q_n} = \frac{(b_{n-1} + a_n/b_n)p_{n-2} + a_{n-1}p_{n-3}}{(b_{n-1} + a_n/b_n)q_{n-2} + a_{n-1}q_{n-3}} = \frac{b_n(b_{n-1}p_{n-2} + a_{n-1}p_{n-3}) + a_n p_{n-2}}{b_n(b_{n-1}q_{n-2} + a_{n-1}q_{n-3}) + a_n q_{n-2}}$$
$$= \frac{b_n p_{n-1} + a_n p_{n-2}}{b_n q_{n-1} + a_n q_{n-2}}.$$

This shows that the formulas are valid also for p_n, q_n. The proof of equation (3.5.6) is left for Problem 2. □

Note that since the denominators and numerators of the approximants satisfy a three term recurrence relation they can be evaluated by Clenshaw's algorithm. It is sometimes convenient to write the recursion formulas in matrix form; see Problem 2.

If we substitute $a_n x$ for a_n in (3.5.4)–(3.5.5) then $p_n(x)$ and $q_n(x)$ become polynomials in x of degree n and $n-1$, respectively.

Example 3.5.1.

Consider the following finite continued fraction

$$r(x) = 7 - \frac{3}{x-2-}\,\frac{1}{x-7+}\,\frac{10}{x-2-}\,\frac{2}{x-3}.$$

[70] Note that this works automatically in IEEE arithmetic, because of the rules of infinite arithmetic; see Sec. 2.2.3!

The algorithm in Theorem 3.5.1 can be used to convert this to rational function form

$$r(x) = \frac{(((7x - 101)x + 540)x - 1204)x + 958}{(((x - 14)x + 72)x - 151)x + 112}.$$

As indicated, the numerator and denominator can then be evaluated by Horner's rule. The backwards evaluation of the continued fraction form requires fewer operations than of the rational form. However, there at the four points $x = 1, 2, 3, 4$ a division by zero occurs even though $r(x)$ is well defined at these points. However, in IEEE arithmetic the continued fraction evaluates correctly at these points because of the rules of infinite arithmetic! Indeed the continued fraction form can be shown to have smaller errors for $x \in [0, 4]$ and to be immune to overflow; see Higham [27, § 27.1].

In practice the forward recursion for evaluating a continued fraction often generates very large or very small values for the numerators and denominators. There is a risk of *overflow or underflow* with these formulas. We are usually not interested in the p_n, q_n themselves, but in the ratios only. Then we can normalize p_n and q_n by multiplying them by the same factor after they have been computed. If we shall go on and compute p_{n+1}, q_{n+1}, however, *we have to multiply* p_{n-1}, q_{n-1} *by the same factor also*! One must also be careful about the numerical stability of these recurrence relations.

The formula

$$\frac{a_1}{b_1+}\,\frac{a_2}{b_2+}\,\frac{a_3}{b_3+}\cdots = \frac{k_1a_1}{k_1b_1+}\,\frac{k_1k_2a_2}{k_2b_2+}\,\frac{k_2k_3a_3}{k_3b_3+}\cdots, \tag{3.5.7}$$

where the k_i are any non-zero numbers, is known as an **equivalence transformation**. The proof of (3.5.7) is left for Problem 5. .

By the following division algorithm, a rational function can be expressed as a continued fraction that can be evaluated by relatively few arithmetic operations; see Cheney [12, *p.* 151]. Let R_0, R_1 be polynomials, and set $R = R_0/R_1$. The degree of a polynomial R_j is denoted by d_j. By successive divisions (of R_{j-1} by R_j) we obtain quotients Q_j and remainders R_{j+1} as follows. For $j = 1, 2, \ldots$, until $d_{j+1} = 0$,

$$R_{j-1} = R_jQ_j + R_{j+1}, \quad d_{j+1} < d_j, \tag{3.5.8}$$

hence

$$R = \frac{R_0}{R_1} = Q_1 + \frac{1}{R_1/R_2} = \ldots = Q_1 + \frac{1}{Q_2+}\,\frac{1}{Q_3+}\cdots\frac{1}{Q_k}. \tag{3.5.9}$$

By means of an equivalence transformation; see (3.5.7), this fraction can be transformed into a slightly more economic form, where the polynomials in the denominators have leading coefficient unity, while the numerators are in general different from 1.

Example 3.5.2. *Best Rational Approximations to a Real Number.*

Every positive number x can be expanded into a continued fraction with integer coefficients of the form,

$$x = b_0 + \frac{1}{b_1+}\,\frac{1}{b_2+}\,\frac{1}{b_3+}\cdots. \tag{3.5.10}$$

Set $x_0 = x$, $p_{-1} = 1$, $q_{-1} = 0$. For $n = 0, 1, 2, \ldots$ we construct a sequence of numbers,

$$x_n = b_n + \frac{1}{b_{n+1}+}\,\frac{1}{b_{n+2}+}\,\frac{1}{b_{n+3}+}\cdots.$$

Evidently $b_n = \lfloor x_n \rfloor$, the integer part of x_n, and $x_{n+1} = 1/(x_n - b_n)$. Compute p_n, q_n, according to the recursion formulas of Theorem 3.5.1, which can be written in vector form,

$$(p_n, q_n) = (p_{n-2}, q_{n-2}) + b_n(p_{n-1}, q_{n-1}),$$

(since $a_n = 1$). See Figure 4.3.1. Stop when $|x - p_n/q_n| < Tol$ or $n > nmax$. The details are left for Problem 1.

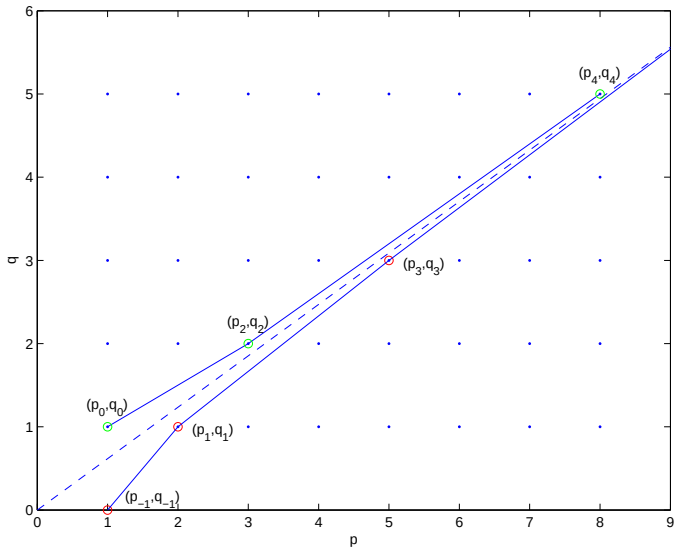


Figure 3.5.1. *Illustration to Example* 3.3.2. *The dashed line is* $\{(p, q) : xq = p\}$ *for* $x = \frac{1}{2}(\sqrt{5} + 1)$.

The above algorithm has been used several times in the previous sections, where some coefficients, known to be rational, has been computed in floating point. It is also useful for finding near commensurabilities between events with different periods;[71] see Problem 1c

[71] One of the convergents for $\log 2/\log 3$ reads 12/19. This is in a way basic for Western Music, where 13 quints make 7 octaves, i.e. $(3/2)^{12} \approx 2^7$.

The German mathematician Felix Klein [28][72] gave the following illuminating description of the sequence $\{(p_n, q_n)\}$ obtained by this algorithm (adapted to our notation):

> "Imagine pegs or needles affixed at all the integral points (p_n, q_n), and wrap a tightly drawn string about the sets of pegs to the right and to the left of the ray, $p = xq$. Then the vertices of the two convex string-polygons which bound our two point sets will be precisely the points $(p_n, q_n)\ldots$, the left polygon having the even convergents, the right one the odd."

Klein also points out that "such a ray makes a cut in the set of integral points" and thus makes Dedekind's definition of irrational numbers very concrete. This construction; see Figure 3.5.1, illustrates in a concrete way that the successive convergents are closer to x than any numbers with smaller denominators, and that the errors alternate in sign. We omit the details of the proof that this description is correct.

Note that, since $a_j = 1$, $\forall j$, equation (3.5.6) reads $p_n q_{n-1} - p_{n-1} q_n = (-1)^{n-1}$. This implies that the triangle with vertices at the points $(0, 0)$, (q_n, p_n), (q_{n-1}, p_{n-1}) has the smallest possible area, among triangles with integer coordinates, and hence there can be no integer points inside or on the sides of this triangle.

Theorem 3.5.2. (Seidel)[73]
Let all b_n be positive in the continued fraction

$$b_0 + \frac{1}{b_1+}\,\frac{1}{b_2+}\,\frac{1}{b_3+}\cdots.$$

Then this converges if and only if the series $\sum b_n$ diverges.

Proof. See Cheney [12, p. 184]. □

Figure 3.5.1 corresponds to the example, (see also Problem 3),

$$x = 1 + \frac{1}{1+}\,\frac{1}{1+}\,\frac{1}{1+}\cdots \tag{3.5.11}$$

From Theorem 3.5.2 it follows that this continued fraction is convergent. Then, note that $x = 1 + 1/x$, $x > 0$, hence $x = (\sqrt{5}+1)/2$. Note also that, by (3.5.6) with $a_j = 1$,

$$\left|x - \frac{p_n}{q_n}\right| \le \left|\frac{p_{n+1}}{q_{n+1}} - \frac{p_n}{q_n}\right| = \frac{|p_{n+1}q_n - p_n q_{n+1}|}{q_{n+1}q_n} = \frac{1}{q_{n+1}q_n} < \frac{1}{q_n^2}. \tag{3.5.12}$$

[72] Felix Christian Klein (1849–1925). He was born 25/4 1849 and delighted in pointing out that each of the day 5^2, month 2^2, and year 43^2 was the square of a prime.

[73] Philipp Ludwig von Seidel (1821–1896) German mathematician and astronomer. In 1846 he submitted his habilitation dissertation entitled "Untersuchungen über die Konvergenz and Divergenz der Kettenbrüche

Comment: If we know or guess that a result x of a computation is a rational number with a reasonably sized denominator, although it was practical to compute it in floating point arithmetic (afflicted by errors of various types), we have a good chance to reconstruct the exact result by applying the above algorithm as a post-processing.

If we just know that the exact x is rational, without any bounds for the number of digits in the denominator and numerator, we must be conservative in claiming that the last fraction that came out of the above algorithm is the exact value of x, even if $|x - p_n/q_n|$ is very small. In fact, the fraction may depend on TOL that is to be chosen with respect to the expected order of magnitude of the error of x. If TOL has been chosen smaller than the error of x, it may, e.g., happen that the last fraction obtained at the termination is wrong, while the correct fraction (with smaller numerator and denominator) may have appeared earlier in the sequence (or it may not be there at all).

So a certain judgment is needed at the application of this algorithm. The smaller the denominator and numerator are, the more likely it is that the fraction is correct. In a serious context, it is advisable to check the result(s) by using exact arithmetic. If x is the root of an equation (or a component of the solution of a system of equations), it is typically much easier to check afterwards that a suggested result is correct than to perform the whole solution process in exact arithmetic.

Continued fractions have also important applications in Analysis; some of the best algorithms for the numerical computation of important analytic functions are based on continued fractions. We shall not give complete proofs but refer to classical books of Perron [35], Wall [47] and Henrici [25, 26].

A continued fraction is said to be equivalent to a given series, iff the sequence of convergents is equal to the sequence of partial sums. There is typically an infinite number of such equivalent fractions. The construction of the continued fraction is particularly simple if we require that the denominators $q_n = 1$, $\forall n \geq 1$. For a power series we shall thus have

$$p_n = c_0 + c_1 x + c_2 x^2 + \ldots c_n x^n, \quad n \geq 1.$$

We must *assume that* $c_j \neq 0$ $\forall j \geq 1$.

We shall determine the the elements a_n, b_n by means of the recursion formulas of Theorem 3.5.1 (for $n \geq 2$) with initial conditions. We thus obtain the following equations,

$$\begin{aligned} p_n &= b_n p_{n-1} + a_n p_{n-2}; \quad p_0 = b_0, \quad p_1 = b_0 b_1 + a_1, \\ 1 &= b_n + a_n; \qquad b_1 = 1. \end{aligned}$$

The solution reads $b_0 = p_0 = c_0$, $b_1 = 1$, $a_1 = p_1 - p_0 = c_1 x$, and for $n \geq 2$,

$$\begin{aligned} a_n &= (p_n - p_{n-1})/(p_{n-2} - p_{n-1}) = -x c_n / c_{n-1}; \\ b_n &= 1 - a_n = 1 + x c_n / c_{n-1}; \end{aligned}$$

$$c_0 + c_1 x + \ldots + c_n x^n \ldots = c_0 + \frac{x c_1}{1-} \, \frac{x c_2/c_1}{1 + x c_2/c_1 -} \, \cdots \frac{x c_n/c_{n-1}}{1 + x c_n/c_{n-1} -} \, \cdots$$

Of course, an equivalent continued fraction gives by itself *no convergence acceleration, just because it is equivalent.* We shall therefore leave the subject of continued fractions equivalent to a series, after showing two instances of the numerous pretty formulas that can be obtained by this construction.

For

$$f(x) = e^x = 1 + x + x^2/2! + x^3/3! + \dots$$

and

$$f(x) = \frac{\arctan\sqrt{x}}{\sqrt{x}} = 1 - x/3 + x^2/5 - x^3/7 + \dots,$$

we obtain for $x = -1$ and $x = 1$, respectively, after simple equivalence transformations,

$$e^{-1} = 1 - \frac{1}{1+}\,\frac{1}{1+y} = \frac{1}{2+y} \;\Rightarrow\; e = 2 + y, \quad \text{where} \quad y = \frac{2}{2+}\,\frac{3}{3+}\,\frac{4}{4+}\,\frac{5}{5+}\dots;$$

$$\frac{\pi}{4} = \frac{1}{1+}\,\frac{1}{2+}\,\frac{9}{2+}\,\frac{25}{2+}\,\frac{49}{2+}\dots.$$

There exist, however, other methods to make a correspondence between a power series and a continued fraction. Some of them lead to a considerable convergence acceleration that often makes continued fractions very efficient for the *numerical computation of functions.* We shall return to such methods in Sec. 3.5.2.

Gauss developed a continued fraction for the ratio of two hypergeometric functions (see (3.1.13))

$$\frac{F(a,b+1,c+1;z)}{F(a,b,c;z)} = \frac{1}{1+}\,\frac{a_1 z}{1+}\,\frac{a_2 z}{1+}\,\frac{a_3 z}{1+}\dots, \tag{3.5.13}$$

$$a_{2n+1} = \frac{(a+n)(c-b+n)}{(c+2n)(c+2n+1)}, \qquad a_{2n} = \frac{(b+n)(c-a+n)}{(c+2n-1)(c+2n)}. \tag{3.5.14}$$

If in (3.5.13) we set $b = 0$, then $F(a,b,c;z) = 1$, and we obtain a continued fraction for $F(a,b+1,c+1;z)$. From this many continued fractions for elementary functions can be derived, such as

$$\ln(1+z) = \frac{z}{1+}\,\frac{z}{2+}\,\frac{z}{3+}\,\frac{2^2 z}{4+}\,\frac{2^2 z}{5+}\,\frac{3^2 z}{6+}\dots. \tag{3.5.15}$$

$$\frac{1}{2}\ln\left(\frac{1+z}{1-z}\right) = \frac{z}{1-}\,\frac{z^2}{3-}\,\frac{2^2 z^2}{5-}\,\frac{3^2 z^2}{7-}\,\frac{4^2 z^2}{9-}\dots \tag{3.5.16}$$

(3.5.17)

$$\arctan z = \frac{z}{1+}\,\frac{z^2}{3+}\,\frac{2^2 z^2}{5+}\,\frac{3^2 z^2}{7+}\,\frac{4^2 z^2}{9+}\dots \tag{3.5.18}$$

$$\tan z = \frac{z}{1-}\,\frac{z^2}{3-}\,\frac{z^2}{5-}\,\frac{z^2}{7-}\dots \tag{3.5.19}$$

$$\tanh z = \frac{e^{2z}-1}{e^{2z}+1} = \frac{z}{1+}\,\frac{z^2}{3+}\,\frac{z^2}{5+}\,\frac{z^2}{7+}\dots. \tag{3.5.20}$$

These expansions can be used also for complex values of z. In fact the fraction for the logarithm can be used in the whole complex plane except in the intervals $(-\infty, -1]$ and $[1, \infty)$. For $\arctan z$, there are similar branch cuts on the imaginary axis. The convergence is slow, when z is near a cut. For an elementary function like these, a program can use some properties of the functions for moving z to a domain, where the continued fraction converges rapidly.

The expansion for $\tan z$ is valid everywhere, except in the poles. In all these cases the region of convergence as well as the speed of convergence is considerably larger than for the power series expansions. For example, the 6'th convergent for $\tan \pi/4$ is almost correct to 11 decimal places.

Example 3.5.3.

Consider the continued fraction for $\ln(1+z)$ and set $z = 1$. The successive approximations to $\ln 2 = 0.69314\,71806$ are:

1/1	2/3	7/10	36/52	208/300	1572/2268	12876/18576
1.000000	0.66667	0.700000	0.692308	0.69333	0.693122	0.693152

Note that the fraction give alternatively upper and lower bounds for $\ln 2$. It can be shown that this is the case when the elements of the continued fraction are positive. To get the accuracy of the last approximation above would require as many as 50,000 terms of the series $\ln 2 = \ln(1+1) = 1 - 1/2 + 1/3 - 1/4 + \cdots$.

Example 3.5.4.

A collection of formulas concerning the important incomplete Gamma function is found in Abramowitz and Stegun [1, Sec. 6.5]. For the sake of simplicity we assume that $x > 0$, although the formulas can be used also in an appropriately cut complex plane. The parameter a may be complex in $\Gamma(a, x)$.[74]

$$
\begin{aligned}
\Gamma(a,x) &= \int_x^\infty t^{a-1}e^{-t}\,dt, \quad \Gamma(a,0) = \Gamma(a),\\
\gamma(a,x) &= \Gamma(a) - \Gamma(a,x) = \int_0^x t^{a-1}e^{-t}\,dt, \quad \Re a > 0,\\
\Gamma(a,x) &= e^{-x}x^a\Big(\frac{1}{x+}\,\frac{1-a}{1+}\,\frac{1}{x+}\,\frac{2-a}{1+}\,\frac{2}{x+}\cdots\Big),\\
\gamma(a,x) &= e^{-x}x^a\Gamma(a)\sum_{n=0}^{\infty}\frac{x^n}{\Gamma(a+1+n)}.
\end{aligned}
\tag{3.5.21}
$$

We mention these functions, because they have many applications. Several other important functions can, by simple transformations, be brought to particular cases of this function, e.g., the normal probability function, the chi-square probability function, the exponential integral, and the Poisson distribution.

[74] There are plenty of other notations for this function.

Continued fractions like these can often be derived by a theorem of Stieltjes[75], which relates continued fractions to orthogonal polynomials that satisfy a recurrence relation of the same type as the one given above. Another method of derivation is the Padé approximation, studied in the next section, that yields a rational function. Both techniques can be looked upon as a *convergence acceleration of an expansion into powers of* z *or* z^{-1}.

3.5.2 Padé Approximants.

The Padé[76] approximants are a particular type of rational approximations to a function $f(z)$ defined by a power series, The idea is to match the coefficients in the given series as far as possible with a rational approximation $P(x)/Q(x)$. Consider the example (Baker [2])

$$f(x) = \left(\frac{1+2x}{1+x}\right)^{1/2} = 1 + \frac{1}{2}x - \frac{5}{8}x^2 + \frac{13}{16}x^3 - \dots.$$

The first three coefficients are matched by the rational approximation

$$f_{11}(x) = \frac{1+7x/4}{1+5x/4} = 1 + \frac{1}{2}x - \frac{5}{8}x^2 + \frac{25}{32}x^3 - \dots.$$

Note that $f_{11}(x)$ has the value 1.4 for $x = \infty$, which agrees well with the limit $\sqrt{2}$ for $f(x)$. This is in contrast to the behavior of the Taylor series for $f(x)$, which does not converge for $x \geq 1/2$.

We now give a general definition of Padé approximants.

Definition 3.5.3.
The (m, n) *Padé approximant associated with*

$$f(z) = \sum_{i=0}^{\infty} c_i z^i. \tag{3.5.22}$$

is, if it exists*, defined to be a rational function*

$$f_{m,n}(z) = \frac{P_{m,n}(z)}{Q_{m,n}(z)} \equiv \frac{\sum_{j=0}^{m} p_j z^j}{\sum_{j=0}^{n} q_j z^j}, \qquad q_0 = 1, \tag{3.5.23}$$

that satisfies

$$r_{m,n}(z) = f(z) - f_{m,n}(z) = Rz^{m+n+1} + O(z^{m+n+2}), \quad z \to 0. \tag{3.5.24}$$

The Padé approximants to e^z are important because of their relation to methods for solving differential equations. Padé arranged the approximants $f_{m,n}(z)$,

[75] Thomas Jan Stieltjes (1856–1894), mathematician born in the Netherlands. On recommendations from his friend and colleague Charles Hermite in Paris he became docent 1886 and professor 1889 at the university in Toulouse, France.

[76] Henri Eugène Padé (1863–1953) a French mathematician, wrote his thesis under Charles Hermite's supervision.

$m, n = 0, 1, 2, \ldots$ in a semi-infinite table. The following is part of the **Padé table** for the exponential function $f(z) = e^z$.

$$\begin{array}{ccc} \dfrac{1}{1} & \dfrac{1+z}{1} & \dfrac{1+z+\frac{1}{2}z^2}{1} \\ \dfrac{1}{1-z} & \dfrac{1+\frac{1}{2}z}{1-\frac{1}{2}z} & \dfrac{1+\frac{2}{3}z+\frac{1}{6}z^2}{1-\frac{1}{3}z} \\ \dfrac{1}{1-z+\frac{1}{2}z^2} & \dfrac{1+\frac{1}{3}z}{1-\frac{2}{3}z+\frac{1}{6}z^2} & \dfrac{1+\frac{1}{2}z+\frac{1}{12}z^2}{1-\frac{1}{2}z+\frac{1}{12}z^2} \end{array}$$

The Padé approximants for e^z were given explicitly by Padé (1892) in his thesis. They are

$$P_{m,n}(z) = \sum_{j=0}^{m} \frac{(m+n-j)!\,m!}{(m+n)!\,(m-j)!}\,\frac{z^j}{j!}, \tag{3.5.25}$$

$$Q_{m,n}(z) = \sum_{j=0}^{n} \frac{(m+n-j)!\,n!}{(m+n)!\,(n-j)!}\,\frac{(-z)^j}{j!}, \tag{3.5.26}$$

with the error

$$r_{m,n}(z) = e^z - \frac{P_{m,n}(z)}{Q_{m,n}(z)} = (-1)^n \frac{m!n!}{(m+n)!(m+n+1)!} z^{m+n+1} + O(z^{m+n+2}). \tag{3.5.27}$$

Note that $P_{m,n}(z) = Q_{m,n}(-z)$, which reflects the property that $e^{-z} = 1/e^z$. Indeed, the nominator and denominator polynomials can be shown to approximate $e^{z/2}$ and $e^{-z/2}$, respectively.

There are several reasons for preferring the diagonal Padé approximants ($m = n$). For these

$$p_j = \frac{(2m-j)!\,m!}{(2m)!\,(m-j)!j!}, \qquad q_j = (-1)^j p_j, \quad j = 0 : m. \tag{3.5.28}$$

The coefficients satisfy the recursion

$$p_0 = 1, \quad p_{j+1} = \frac{(m-j)p_j}{(2m-j)(j+1)}, \quad j = 0 : m-1. \tag{3.5.29}$$

For the diagonal Padé approximants the error $R_{m,n}(z)$ satisfy $|R_{m,n}(z)| < 1$, for $\Re z < 0$. This is an important property in applications to solving differential equations.[77] To evaluate a diagonal Padé approximant of even degree we write

$$\begin{aligned} P_{2m,2m}(z) &= p_{2m}z^{2m} + \cdots + p_2 z^2 + p_0 \\ &\quad + z(p_{2m-1}z^{2m-2} + \cdots + p_3 z^2 + p_1) = u(z) + v(z). \end{aligned}$$

[77] Diagonal Padé approximants are used also for the evaluation of the matrix exponential e^A, $A \in \mathbf{R}^{n\times n}$; see Chapter 9.

and evaluate $u(z)$ and $v(z)$ separately. Then $Q_{2m}(z) = u(z) - v(z)$. A similar splitting can be used for an odd degree.

It was remarked in Sec. 2.2.4 that in order to compute the exponential function a range reduction should first be performed. If an integer k is determined such that

$$z^* = z - k\ln 2, \quad |z^*| \in [0, \ln 2] \tag{3.5.30}$$

then $\exp(z) = \exp(z^*) \cdot 2^k$. Hence only an approximation of $\exp(z)$ for $|z| \in [0, \ln 2]$ is needed; see Problem 5.

We now consider how to determine the Padé approximants in the general case.

Theorem 3.5.4.

Let $f(z)$ be a function defined by the power series (3.5.22)*. The coefficients q_j $j = 1 : n$, of the denominator of the Padé approximant $f_{m,n}(z)$ are determined by the linear system,*

$$\sum_{j=1}^{n} c_{i-j} q_j + c_i = 0, \quad i = m+1 : m+n, \tag{3.5.31}$$

where we set $c_i = 0$ for $i < 0$, provided that this linear system has a unique solution. Further, the coefficients of the numerator are

$$\sum_{j=0}^{k} c_{i-j} q_j = p_i, \quad i = 0 : m, \quad k = \min(i, n), \tag{3.5.32}$$

and the error constant R in (3.5.24) *reads*

$$R = \sum_{j=0}^{k} c_{i-j} q_j, \quad i = m+n+1.$$

Proof. Insert (3.5.22) and (3.5.24) into (3.5.23) and multiply both sides by the denominator:

$$\Big(\sum_{l=0}^{\infty} c_l z^l + R z^{m+n+1} + O(z^{m+n+2}) \Big) \sum_{j=0}^{n} q_j z^j = \sum_{i=0}^{m} p_i z^i.$$

Match the coefficients of z^i, $i = 0 : m+n+1$, and remember that $q_0 = 1$:

$$\sum_{j=0}^{n} c_{i-j} q_j = \begin{cases} p_i, & \text{if } 0 \le i \le m; \\ 0, & \text{if } m+1 \le i \le m+n; \\ R, & \text{if } i = m+n+1. \end{cases}$$

The statements follow from this. □

Note that $f_{m,n}$ uses c_l for $l = 0 : m+n$ only; R uses c_{m+n+1} also. So, if c_l is given for $l = 0 : r$ then $f_{m,n}$ is defined for $m + n \le r$, $m \ge 0$, $n \ge 0$.

There is an "if" in the theorem. There are in fact simple exceptional situations, where the linear system (3.5.31) is singular. The system can be written in more detail as

$$\begin{pmatrix} c_{m-n+1} & c_{m-n+2} & \cdots & c_m \\ c_{m-n+2} & c_{m-n+3} & \cdots & c_{m+1} \\ \vdots & \vdots & \cdots & \vdots \\ c_m & c_{m+1} & \cdots & c_{m+n-1} \end{pmatrix} \begin{pmatrix} q_n \\ q_{n-1} \\ \vdots \\ q_1 \end{pmatrix} = - \begin{pmatrix} c_{m+1} \\ c_{m+2} \\ \vdots \\ c_{m+n} \end{pmatrix}$$

where $c_i = 0$, $i < 0$. Note the system matrix has constant elements along the anti-diagonals. Such matrices are called **Hankel matrices**. It can be shown that singular cases occur in square blocks of the Padé table, where all the approximants are equal. This property, investigated by Padé, is known as the *block structure of the Padé table*. A Padé table where all the approximants are different is called **normal**. Otherwise it is called **non-normal**.

We shall indicate, how such singular situations can often be avoided by a more reasonable formulation of the request. These matters are discussed more thoroughly, e.g., in Cheney [12, Chap. 5].

Example 3.5.5.

Let for $f(z) = \cos z = 1 - \frac{1}{2}z^2$ and try to find

$$f_{1,1}(z) = (p_0 + p_1 z)/(q_0 + q_1 z), \quad q_0 = 1.$$

The coefficient matching according to the theorem, yields the equations,

$$p_0 = q_0 = 1, \qquad p_1 = q_1, \qquad -\frac{1}{2}q_0 = 0.$$

The last equation contradicts the condition that $q_0 = 1$. This single contradictory equation is in this case the "system" (3.5.31).

If this equation is ignored, we obtain $f_{1,1}(z) = (1 + q_1 z)/(1 + q_1 z) = 1$, with error $\approx \frac{1}{2}z^2$, in spite that we asked for an error that is $O(z^{m+n+1}) = O(z^3)$. If we instead allow that $q_0 = 0$, then $p_0 = 0$, and we obtain the same final result, since $f_{1,1}(z) = q_1 z/(q_1 z) = 1$.

In a sense, this singular case corresponds to a rather stupid request: we ask to approximate the even function $\cos z$ by a rational function where the numerator and the denominator end with odd powers of z. One should, of course, ask for the approximation by a rational function of z^2. What would you do, if $f(z)$ is an *odd* function?

Imagine a case where $f_{m-1,n-1}(z)$ happens to be a more accurate approximation to $f(z)$ than usual, say that $f_{m-1,n-1}(z) - f(z) = O(z^{m+n+1})$. (For instance, let $f(z)$ be the ratio of two polynomials of degree $m-1$ and $n-1$, respectively.) Let b be an arbitrary number, and choose

$$Q_{m,n}(z) = (z + b)Q_{m-1,n-1}(z),$$
$$P_{m,n}(z) = (z + b)P_{m-1,n-1}(z).$$

Table 3.5.1. *The Coefficients p_i and q_j of Padé approximations for e^z.*

$m\backslash i$	0	1	2	3	4
0	1	0	0	0	0
1	1	1/4	0	0	0
2	1	1/2	1/12	0	0
3	1	3/4	1/4	1/24	0
4	1	1	1/2	1/6	1/24

$m\backslash j$	0	1	2	3	4
0	1	−1	1/2	−1/6	1/24
1	1	−3/4	1/4	−1/24	0
2	1	−1/2	1/12	0	0
3	1	−1/4	0	0	0
4	1	0	0	0	0

Then

$$\begin{aligned} f_{m,n}(z) &= P_{m,n}(z)/Q_{m,n}(z) \\ &= P_{m-1,n-1}(z)/Q_{m-1,n-1}(z) = f_{m-1,n-1}(z), \end{aligned}$$

which is an $O(z^{m+n+1})$-accurate approximation to $f(z)$. Hence our request for this accuracy is satisfied by more than one pair of polynomials, $P_{m,n}(z)$, $Q_{m,n}(z)$, since b is arbitrary. This is impossible, unless the system (3.5.31) (that determines $Q_{m,n}$) is singular.

This illustrates another type of situations where the singular case occurs. Numerically, a similar situation occurs in a natural way, when one wants to approximate $f(z)$ by $f_{m,n}(z)$, although already $f_{m-1,n-1}(z)$ would represent $f(z)$ as well as possible with the limited precision of the computer. In this case we must expect the system (3.5.31) to be very close to a singular system. A reasonable procedure for handling this is to compute the Padé approximants for a sequence of increasing values of m, n, to estimate the condition numbers and to stop when it approaches the reciprocal of the machine unit. This illustrates a fact of some generality. *Unnecessary numerical trouble can be avoided by means of a well designed termination criterion.*

For $f(z) = -\ln(1-z)$, we have $c_l = 1/l$, $l > 0$. When $m = n$ the matrix of the system (3.5.31) turns out to be the notorious Hilbert matrix (with permuted columns), for which the condition number grows exponentially; see Example 2.4.7. (The elements of the usual Hilbert matrix are $a_{ij} = 1/(i+j-1)$.)

Example 3.5.6.

The Padé approximations $f_{m,n}(z)$ and the corresponding error terms were computed by a program using the formulas of Theorem 3.5.4 for $f(z) = e^z$, with $0 \le m \le 4$, $n = 4 - m$. In the Padé table these will be on the fourth diagonal, perpendicularly to the main diagonal. The input was the Malaurin coefficients of $f_{4,0}$. The results were first obtained in floating point arithmetic, but they were then converted into rational form by the algorithm described in Example 3.5.2. The coefficients p_i of the numerators $P_{m,4-m}$ and q_j of the denominators $Q_{m,4-m}$ are given in Table 3.5.1. For $m = n = 4$ the program found that the error term is $-4 \cdot 10^{-8} z^9$, while the error term of the Maclaurin expansion $f_{8,0}$ is $3 \cdot 10^{-6} z^9$.

When $m = n = 10$ the program gave warnings about divisions by zero, and

it estimated the condition number of the linear system (3.5.31) to be 10^{22}. The reciprocal of this number is a measure of how close the matrix of the system is to a singular matrix, (see Theorem 7.5.3). The computed coefficients of the Padé approximant had large errors. Nevertheless e was computed with full machine accuracy (for $z = 1$), and the error term was estimated to be less than $10^{-25}z^{21}$.

Example 3.5.7.

To evaluate $\ln(1 + x)$ one can use the relation

$$\ln(1+x) = \ln\left(\frac{1+z}{1-z}\right), \quad z = \frac{x/2}{1+x/2},$$

and use the continued fraction expansion given in (3.5.3). The convergents of this continued fraction are odd functions and Padé approximants. The first few are

$$s_{00} = 2z, \qquad s_{01} = \frac{3}{3-z^2}, \qquad s_{11} = 2z\frac{15+4z^2}{3(5-3z^2)},$$
$$s_{12} = \frac{105-55z^2}{105-90z^2+9z^4}, \qquad s_{22} = 2z\frac{945-735z^2+64z^4}{15(63-70z^2+15z^4)}.$$

Here the diagonal approximants s_{mm} are most interest. For example, the approximation s_{22} matches the Taylor series up to the term z^8 and the error is approximately equal to the term $z^{10}/11$. Note that the denominators are the Legendre polynomials in $1/z$,

3.5.3 The Epsilon Algorithm.

We shall here briefly introduce the important ϵ-algorithm and indicate the connections between Padé approximation, Aitken acceleration, linear difference equations and this algorithm.

If n is large, the heavy part of the computation of a Padé approximant

$$f_{m,n}(z) = P_{m,n}(z)/Q_{m,n}(z)$$

of $f(z)$ in (3.5.22) is the solution of the linear system (3.5.31). We see that if m or n is decreased by 1, most of the equations of the system will be the same. There are therefore relations between the polynomials $Q_{m,n}(z)$ for adjacent values of m, n, which have been subject to intensive research that has resulted in several interesting algorithms. See, e.g., the monographs of Brezinski [8, 9] and the literature cited there.

Here we are primarily interested in the use of Padé approximants as a convergence accelerator in the *numerical* computation of values of $f(z)$ for (say) $z = e^{i\phi}$, in particular for $z = \pm 1$. A natural question is whether it is possible to omit the calculation of the coefficients p_j, q_j, and find a recurrence relation that gives the function values directly. A very elegant solution to this problem, called the ϵ-algorithm, was found in 1956 by P. Wynn [48], after complicated calculations. We shall present the algorithm, but we refer to the original paper of Wynn for the proof.

A two-dimensional array of numbers $\epsilon_k^{(p)}$ is computed by the recurrence relation,

$$\epsilon_{k+1}^{(p)} = \epsilon_{k-1}^{(p+1)} + \frac{1}{\epsilon_k^{(p+1)} - \epsilon_k^{(p)}}, \tag{3.5.33}$$

which involves quantities in a rhombus

$$\begin{array}{ccc} & \epsilon_k^{(p)} & \\ \epsilon_{k-1}^{(p+1)} & & \epsilon_{k+1}^{(p)} \\ & \epsilon_k^{(p+1)} & \end{array}$$

If the following boundary conditions are used:

$$\begin{aligned} &\epsilon_{-1}^{(p)} = 0, \\ &\epsilon_0^{(p)} = f_{p,0}(z) = \sum_{j=0}^{p} c_j z^j, \\ &\epsilon_{2n}^{(-n)} = f_{0,n}(z) = \frac{1}{\sum_{j=0}^{n} d_j z^j}, \end{aligned} \tag{3.5.34}$$

this yields for *even* subscripts

$$\epsilon_{2n}^{(p)} = f_{p+n,n}(z), \tag{3.5.35}$$

The values of $\epsilon_{2n+1}^{(p)}$ with *odd* subscripts are auxiliary quantities only. The polynomials $f_{0,n}(z)$ are obtained from the Taylor expansion of $1/f(z)$. Several procedures for obtaining this were given in Sec. 3.1.

It seems easier to program the ϵ-algorithm it after a slight change of notation. We introduce an $r \times 2r$ matrix $A = [a_{ij}]$, $a_{ij} = \epsilon_k^{(p)}$, where $k = j - 2$, $p = i - j + 1$. Conversely, $i = k + p + 1, j = k + 2$. The ϵ-algorithm, together with the boundary conditions now takes the form:

for $i = 1 : r$
  $a_{i,1} = 0; \quad a_{i,2} = f_{i-1,0}(z); \quad a_{i,2i} = f_{0,i-1}(z);$
  for $j = 2 : 2 * i - 2$
    $a_{i,j+1} = a_{i-1,j-1} + 1/(a_{ij} - a_{i-1,j}).$
  end
end

Results:

$$f_{m,n}(z) = a_{m+n+1,2n+2}, \quad (m, n \geq 0, \quad m + n + 1 \leq r).$$

The above program sketch must be improved for practical use, e.g., something should be done about the risk for a division by zero.

An extension of the Aitken acceleration, due to Shanks [39] 1955, uses a comparison series with terms of the form

$$c_j = \sum_{\nu=1}^{p} \alpha'_\nu k_\nu^j, \quad j \geq 0, \quad k_\nu \neq 0. \tag{3.5.36}$$

Here α'_ν and k_ν are $2p$ parameters, to be determined, in principle, by means of c_j, $j = 0 : 2p-1$. The parameters may be complex. The power series becomes

$$S(z) = \sum_{j=0}^{\infty} c_j z^j = \sum_{\nu=1}^{p} \alpha'_\nu \sum_{j=0}^{\infty} k_\nu^j z^j = \sum_{\nu=1}^{p} \frac{\alpha'_\nu}{1 - k_\nu z}.$$

This is a rational function of z, and the "Ansatz" of Shanks is thus related to Padé approximation, but note that the poles at k_ν^{-1} should be simple and that $m < n$ for $S(z)$, because $S(z) \to 0$, as $z \to \infty$. Recall that the calculations for the Padé approximation determines the coefficients of $S(z)$ *without calculating the* $2n$ *parameters* α'_ν *and* k_ν. It can happen that m becomes larger than n, and if α'_ν and k_ν are afterwards determined, by the expansion of $S(z)$ into partial fractions, it can turn out that some of the k_ν are multiple poles.

This suggests a generalization of the Shanks approach but how? If we consider the coefficients q_j, $j = 1 : n$, occurring in (3.5.31) as known quantities then (3.5.31) can be interpreted as a *linear difference equation*[78] . The general solution of this is given by (3.5.36), if the zeros of the polynomial

$$Q(x) := 1 + \sum_{j=1}^{n} q_j x^j$$

are simple, but if multiple roots are allowed, the general solution reads,

$$c_l = \sum_{\nu} p_\nu(l) k_\nu^n,$$

where k_ν runs through the different zeros of $Q(x)$, and p_ν is an arbitrary polynomial, the degree of which equals the multiplicity -1 of the zero k_ν.

Essentially the same mathematical relations occur in several areas of numerical analysis, such as interpolation and approximation by a sum of exponentials, and in the design of quadrature rules with free nodes (see Sec. 5.2). For an application of the ϵ-algorithm to numerical quadrature, see Sec. 5.3.3.

3.5.4 The QD Algorithm.

Given the continued fraction

$$c(z) = \frac{a_1}{1+} \frac{a_2 z}{1+} \frac{a_3 z}{1+}, \tag{3.5.37}$$

[78] This can also be expressed in terms of the z-transform; see § 3.2.3.

we denote the nth approximant by

$$w_n(z) = P_n(z)/Q_n(z), \quad n = 1, 2, \ldots. \tag{3.5.38}$$

This corresponds to the finite continued fraction obtained by setting $a_{n+1} = 0$. The sequence of numerators $\{P_n\}$ and denominators $\{Q_n\}$ in (3.5.38) satisfy the recurrence relations:

$$\begin{aligned} P_0 = 0, \quad P_1 = 1, \qquad & P_n = z a_n P_{n-2} + P_{n-1}, \\ Q_0 = Q_1 = 1, \qquad & Q_n = z a_n Q_{n-2} + Q_{n-1}, \quad n \geq 2, \end{aligned}$$

Hence both P_n and Q_n are polynomials in z of degree $[(n-1)/2]$ and $[n/2]$, respectively. It can be shown that the polynomials P_n and Q_n have no common zero for $n = 1, 2, \ldots$, and for all z.

In the special case that all $a_i > 0$, the continued fraction

$$c(z) = \frac{a_1}{1+} \frac{a_2 z}{1+} \frac{a_3 z}{1+}, \tag{3.5.39}$$

is called a Stieltjes fraction.[79]

From the initial conditions and recurrence relations it follows that $Q_n(0) = 1$, $n = 0, 1, 2, \ldots$. Hence the rational function $w_n(z)$ is analytic at $z = 0$ and thus can be expanded in a Taylor series

$$\frac{P_n(z)}{Q_n(z)} = c_0^{(n)} + c_1^{(n)} z + c_2^{(n)} z^2 + \cdots \tag{3.5.40}$$

that converges for z sufficiently small. The coefficients $c_k^{(n)}$ in (3.5.40) can be shown to be independent of n for $k < n$. We denote by $c_k := c_k^{(n+1)}$ the ultimate value of $c_k^{(n)}$ for increasing values n and let

$$f(z) = c_0 + c_1 z + c_2 z^2 + \cdots, \tag{3.5.41}$$

be the formal power series formed with these coefficients. Then the power series $f(z)$ and the fraction $c(z)$ are said to **correspond** to each other. Note that the formal power series $f(z)$ corresponding to a given fraction $c(z)$ converges for any $z \neq 0$.

We now consider the converse problem: Given a (formal) power series $f(z)$, find a continued fraction $c(z)$ of the form (3.5.50) corresponding to it. Note that we do not require that the formal power series corresponding to the continued fraction converges, merely that the nth approximant w_n of the continued fraction satisfies

$$f(z) - w_n(z) = O(z^n).$$

[79] The theory of such fractions was first expounded by Stieltjes in a famous memoir, which appeared in 1894, the year of his death.

Example 3.5.8.
For $|z| < 1$,

$$\arctan z = z - \tfrac{1}{3}z^3 + \tfrac{1}{5}z^5 - \tfrac{1}{7}z^7 + \cdots.$$

The corresponding partial numerators and the corresponding continued fractions are

$$a_{2k} = \frac{(2k-1)^2}{(4k-3)(4k-1)}, \qquad a_{2k+1} = \frac{(2k)^2}{(4k-1)(4k+1)}.$$

The fraction converges for all z such that z^2 is not real and $z^2 \leq 1$. After an equivalence transformation we obtain

$$\arctan z = \frac{z}{1+}\,\frac{z^2}{3+}\,\frac{4z^2}{5+}\,\frac{9z^2}{7+}\,\frac{16z^2}{9+} \tag{3.5.42}$$

The convergents of the corresponding continued fractions are equal to Padé approximants.

The **qd algorithm**[80], can be used to compute such a continued fraction, if it exists.

For arbitrary integers n and $k \geq 0$, we define the Hankel matrices

$$H_k^{(n)} = \begin{pmatrix} c_n & c_{n+1} & \cdots & c_{n+k-1} \\ c_{n+1} & c_{n+2} & \cdots & c_{n+k} \\ \vdots & \cdots & \cdots & \vdots \\ c_{n+k-1} & c_{n+k-2} & \cdots & c_{n+2k-2} \end{pmatrix} \in \mathbf{R}^{k\times k}, \tag{3.5.43}$$

where we set $c_k = 0$ for $k < 0$. Further, we define the Hankel determinants

$$\mathbf{H}_k^{(n)} = \det\left(H_k^{(n)}\right), \quad k = 1, 2, \ldots. \tag{3.5.44}$$

associated with the formal power series (3.5.41).

Theorem 3.5.5. Henrici [26, Theorem 12.4c]
Given a formal power series (3.5.41)*, there exists at most one corresponding continued fraction. It exists precisely one such fraction if and and only if the Hankel determinants* (3.5.44) *satisfy* $H_k^{(n)} \neq 0$ *for* $n = 0, 1$ *and* $k = 1, 2, \ldots$.

The Hankel determinants satisfy the following important identity called **Jacobi's identity**:
For all integers n and $k \geq 1$

$$(H_k^{(n)})^2 - H_k^{(n-1)}H_k^{(n+1)} + H_{k+1}^{(n-1)}H_{k-1}^{(n+1)} = 0. \tag{3.5.45}$$

[80]The qd algorithm was originally given by the Swiss mathematician Heinz Rutishauser [38]

If the determinants $H_k^{(n)}$ are arranged in a triangular array

$$\begin{array}{llllll} 1 & & & & & \\ 1 & H_1^{(0)} = c_0 & & & & \\ 1 & H_1^{(1)} = c_1 & H_2^{(0)} & & & \\ 1 & H_1^{(2)} = c_2 & H_2^{(1)} & H_3^{(0)} & & \\ 1 & H_1^{(3)} = c_3 & H_2^{(2)} & H_3^{(1)} & H_4^{(0)} & \end{array}$$

then Jacobi's identity links together the entries in a star like configuration. Since the two first columns are trivial (3.5.45) may be used to calculate the Hankel determinants recursively from left to right.

The **quotient-difference scheme**, or qd scheme is a scheme

$$\begin{array}{lllllll} & q_1^{(0)} & & & & & \\ 0 & & e_1^{(0)} & & & & \\ & q_1^{(1)} & & q_2^{(0)} & & & \\ 0 & & e_1^{(1)} & & e_2^{(0)} & & \\ & q_1^{(2)} & & q_2^{(1)} & & q_3^{(0)} & \\ 0 & & e_1^{(2)} & & e_2^{(1)} & & \\ & q_1^{(3)} & & q_2^{(2)} & & q_3^{(1)} & \\ 0 & & e_1^{(3)} & & e_2^{(2)} & & \\ & \vdots & & q_2^{(3)} & & \vdots & \\ & & \vdots & & \vdots & & \end{array},$$

where the quantities are connected by the two **rhombus rules**

$$e_m^{(n)} = q_m^{(n+1)} - q_m^{(n)} + e_{m-1}^{(n+1)}; \quad m = 1, 2, \ldots, \ n = 0, 1, 2, \ldots, \tag{3.5.46}$$

$$q_{m+1}^{(n)} = \frac{e_m^{(n+1)}}{e_m^{(n)}} q_m^{(n+1)}; \quad m = 1, 2, \ldots, \ n = 0, 1, 2, \ldots, \tag{3.5.47}$$

The qd scheme associated with the formal power series (3.5.41) is obtained by taking the entries in the second column to be

$$q_1^{(n)} = c_{n+1}/c_n, \quad n = 0, 1, 2, \ldots, \tag{3.5.48}$$

The remaining elements in the qd scheme can then be generated column by column using the rhombus rules. If the columns $q_{m+1}^{(n)}$, $m = 1, 2, \ldots$ exist, then the continued fraction corresponding to f is given by

$$c = \frac{c_0}{1-} \frac{q_1^{(0)} z}{1-} \frac{e_1^{(0)} z}{1-} \frac{q_2^{(0)} z}{1-} \frac{e_2^{(0)} z}{1-} - \cdots, \tag{3.5.49}$$

Example 3.5.9.

For the power series $c(z) = 0! + 1!z + 2!z^2 + 3!z^3 + \cdots$, the following qd scheme is obtained:

$$\begin{array}{ccccccccc}
 & 1 & & & & & & & \\
0 & & 1 & & & & & & \\
 & 2 & & 2 & & & & & \\
0 & & 1 & & 2 & & & & \\
 & 3 & & 3 & & 3 & & & \\
0 & & 1 & & 2 & & 3 & & \\
 & 4 & & 4 & & 4 & & & \\
0 & & 1 & & 2 & & 3 & &
\end{array}$$

Hence the corresponding continued fraction is

$$c(z) = \frac{1}{1+}\,\frac{z}{1+}\,\frac{z}{1+}\,\frac{2z}{1+}\,\frac{2z}{1+}\,\frac{3z}{1+}.$$

It is sometimes convenient to consider continued fractions in z^{-1}

$$c(z) = \frac{a_1}{1+}\,\frac{a_2 z^{-1}}{1+}\,\frac{a_3 z^{-1}}{1+}, \tag{3.5.50}$$

that corresponds to the formal series

$$p = c_0 + c_1 z^{-1} + c_2 z^{-2} + \cdots.$$

Review Questions

1. Define a continued fraction. Show how the convergents can be evaluated either backwards or forwards.
2. Show how any positive number can be expanded into a continued fraction with integer elements. In what sense are the convergents the best approximations? How accurate are they?
3. The denominators or numerators of the approximants of a continued fraction can be evaluated by Clenshaw's algorithm. Why is that?
4. What is the Padé table? Describe how the Padé approximants can be computed, if they exist. Tell something about singular and almost singular situations that can be encountered, and how to avoid them.
5. Describe the ϵ-algorithm, and tell something about its background and its efficiency.

6. Describe the qd algorithm. What can it be used for?

Problems and Computer Exercises

1. (a) Write a program for the algorithm of Example 3.5.2. Apply it to find a few coefficients of the continued fractions for

$$\tfrac{1}{2}(\sqrt{5}+1), \quad \sqrt{2}, \quad e, \quad \pi, \quad \log 2/\log 3, \quad 2^{j/12}$$

for a few integers j, $1 \le j \le 11$.

(b) Check the accuracy of the convergents. What happens when you apply your program to a rational number, e.g., 729/768 ?

(c) The metonic cycle used for calendrical purposes by the Greeks consists of 235 lunar month, which nearly equal 19 solar years. Show, using the algorithm in Example 3.5.2, that 235/19 is the sixth convergent of the ratio 365.2495/29.53059 of the Lunar phase synodic) period and solar period

2. A matrix formalism for continued fractions.

(a) We use the same notations as in Sec. 3.4.1, but we set, with no loss of generality, $b_0 = 0$. Set

$$P(n) = \begin{pmatrix} p_{n-1} & p_n \\ q_{n-1} & q_n \end{pmatrix}, \qquad A(n) = \begin{pmatrix} 0 & a_n \\ 1 & b_n \end{pmatrix}.$$

Show that $P(0) = I$,

$$P(n) = P(n-1)A(n), \qquad P(n) = A(1)A(2)\cdots A(n-1)A(n), \quad n \ge 1.$$

Comment: This does not minimize the number of arithmetic operations but, in a matrix-oriented programming language, it often gives very simple programs.

(b) Write a program for this with some termination criterion, and test it on a few cases, e.g.,

$$1 + \frac{1}{1+}\,\frac{1}{1+}\,\frac{1}{1+}\ldots; \quad 2 + \frac{1}{3+}\,\frac{1}{2+}\,\frac{1}{3+}\,\frac{1}{2+}\,\frac{1}{3+}\ldots; \quad 2 + \frac{2}{2+}\,\frac{3}{3+}\,\frac{4}{4+}\ldots.$$

As a post-processing, apply in the first two cases, e.g., Aitken acceleration in order to obtain a very high accuracy. Does the result look familiar in the last case? See Problem 3 concerning the exact results in the two other cases.

(c) Write a version of the program with some strategy for scaling $P(n)$ in order to eliminate the risk of overflow and underflow.

Hint: Note that the convergents $x_n = p_n/q_n$ are unchanged if you multiply the $P(n)$ by arbitrary scalars.

(d) Use this matrix form for working out a short proof of (3.5.6).

Hint: What is the determinant of a matrix product?

3. (a) Explain that $x = 1 + 1/x$ for the continued fraction in (3.5.11)?

(b) Compute the periodic continued fraction

$$2 + \frac{1}{3+}\,\frac{1}{2+}\,\frac{1}{3+}\,\frac{1}{2+}\,\frac{1}{3+}\cdots$$

exactly (by paper and pencil). (The convergence is assured by Seidel's Theorem 3.5.2.)

(c) Suggest a generalization of (a) and (b), where you can always obtain a quadratic equation with a positive root.

(d) Show that

$$\frac{1}{\sqrt{x^2-1}} = \frac{1}{x-}\,\frac{\frac{1}{2}}{x-y} \quad \text{where} \quad y = \frac{\frac{1}{4}}{x-}\,\frac{\frac{1}{4}}{x-}\,\frac{\frac{1}{4}}{x-}\cdots.$$

4. (a) Prove the equivalence transformation (3.5.7). Show that the errors of the convergents have alternating signs, if the elements of the continued fraction are positive.

(b) Show how to bring a general continued fraction to the special form of equation (3.5.10).

5. Let $P_{m,m}(z)/Q_{m,m}(z)$ be the diagonal Padé approximants of the exponential function. Show that the coefficients for $P_{m,m}(z)$ satisfy the recursion

$$p_0 = 1, \quad p_{j+1} = \frac{m-j}{(2m-j)(j+1)} p_j, \quad j = 0 : m-1. \tag{3.5.51}$$

(b) Show that for $m = 6$ we have

$$P_{6,6}(z) = 1 + \frac{1}{2}z + \frac{5}{44}z^2 + \frac{1}{66}z^3 + \frac{1}{792}z^4 + \frac{1}{15840}z^5 + \frac{1}{665280}z^6.$$

and $Q_{6,6}(z) = P_{6,6}(-z)$. How many operations are needed to evaluate this approximation for a given z?

(c) Use the error estimate in (3.5.27), neglecting higher order terms, to compute a bound for the relative error of the approximation in (b) when $|z| \in [0, \ln 2]$. What degree of the diagonal Padé approximant is needed for the relative error is required to be of the oder of the unit roundoff $2^{-53} = 1.11 \cdot 10^{-16}$ in IEEE double precision?

6. (a) Write a program for computing a Padé approximant and its error term. Apply it (perhaps after a transformation), for various values of m, n to, e.g., e^z, $\arctan z$, $\tan z$. (Note that two of these examples are odd functions.) Use the algorithm of Example 3.5.2 for expressing the coefficients as rational numbers. For how large m, n can you (in these examples) use your program without severe trouble with rounding errors.

(b) (b) Try to determine for which other functions the Padé table has a similar symmetry as shown in the text for the exponential function e^z.

7. (a) Show that there is at most one rational function $R(z)$, where the degrees of the numerator and denominator do not exceed, respectively, m and n, such that

$$f(z) - R(z) = O(z^{m+n+1}), \quad \text{as} \quad z \to 0,$$

even if the system (3.5.31) is singular. (Note, however, that P_m and Q_n are not uniquely determined, if the system is singular; they have common factors.)

(b) Is it true that if $f(z)$ is a rational function of degrees m', n', then

$$f_{m,n}(z) = f(z), \quad \forall\ m \geq m', \quad n \geq n'?$$

8. Write a program for evaluation the incomplete gamma function. Use the continued fraction (3.5.21) for x greater than about $a + 1$. For x less than about $a + 1$ use the power series for $\gamma(a, x)$.

9. Check that the program sketch for the ϵ-algorithm is equivalent with the scheme with the quantities $\epsilon_k^{(p)}$ given earlier in the text. How do you obtain the boundary values?

Notes and References

Much work on approximations to special functions, e.g., Gauss hypergeometric function and the Kummer function, was done around the end of World War II. This work culminated with the appearance of the classical *Handbook of Mathematical Functions* edited by Milton Abramowitz and Irene A. Stegun [1, 1965]. Chapters 13 and 15 in this handbook contain many useful formulas and tables. Sections 15.1, 15.4, and Table 13.6, show how *many other important functions, elementary as well as advanced special functions, can be expressed in terms of these functions.*

The basic properties of these functions are derived in Lebedev's monograph on Special Functions [30]. Lebedev's compact book will often be referred to, because it provides a good background to the applications of advanced Analysis, that lacks complete proofs in our book. For example, the chapter on the gamma function contains numerous instances of the use of series expansions and analytic continuation that are efficient as well as instructive, important and beautiful. Codes and other interesting information concerning the evaluation of special functions are also found in a modern classic, Numerical Recipes [36, Chapter 5–6].

The idea of using Cauchy's formula and FFT for numerical differentiation seems to have been first suggested by Lyness and Moler [32].

The theory of continued fractions started to develop in the 17th century. The main contributors were Euler, Lambert and Lagrange; see Brezinski [9]. The basic algorithmic aspects of what we today call Padé approximants were established by Frobenius [18]. Padé [34] gave a systematic study of these approximants and introduced the table named after him. The analytic theory of continued fractions has earlier origins and contributors include Chebyshev, A. A. Markov and Stieltjes. Modern related developments are the epsilon algorithm of P. Wynn and the quotient-difference algorithm of Rutishauser. An easy to read introduction to continued fractions and Padé approximations is Baker [2]. Their use in numerical

computations is surveyed in Blanche [4]. A survey of the more recent developments of Padé approximations is given by Gragg [22]. Continued fractions of special functions are found in Abramowitz and Stegun [1]. Codes and further references are given in Numerical Recipes, Press et al. [36, Chapters 5 and 6]. The following example contains a different type of continued fraction. More information about arithmetic continued fractions, from a computational point of view is found in Riesel [37].

More information about the classical methods for polynomial interpolation of equidistant data is found in, e.g., Fröberg [20] and Steffensen [42], in particular § 18 about "the calculus of symbols". For the history of these matters see, e.g., Goldstine [21].

More complete presentation of extrapolation methods is given in the monograph by Claude and Redivo-Zaglia [11], and more recently Sidi [40]. The historical development of the field is nicely surveyed by Brezinski [10].

A rigorous theory of semi-convergent series was developed by Stieltjes and Poincaré in 1886.

Bibliography

[1] Milton Abramowitz and Irene A. Stegun (eds.). *Handbook of Mathematical Functions*. Dover, New York, NY, 1965.

[2] G. A. Baker, Jr. *Essentials of Padé Approximants*. Academic Press, New York, 1975.

[3] Petter Bjørstad, Germund Dahlquist, and Eric Grosse. Extrapolations of asymptotic expansions by a modified Aitken δ^2-formula. *BIT*, 21:56–65, 1981.

[4] G. Blanche. Numerical evaluation of continued fractions. *SIAM Review*, 6:4:383–421, 1964.

[5] G. A. Bliss. *Algebraic Functions*, volume Reprinted 1947, Edwards Brothers, Ann Arbor, MI. Amer. Math. Soc., New York, 1933.

[6] Jon M. Borwein, Peter B. Borwein, and K. Dilcher. Pi, Euler numbers and asymptotic expansions. *Amer. Math. Monthly*, 96:681–687, 1989.

[7] RIchard P. Brent and H. T. Kung. Fast algorithms for manipulating formal power series. *J. ACM*, 25:581–595, 1978.

[8] Claude Brezinski. *Padé-Type Approximations and General Orthogonal Polynomials*. Birkhäuser Verlag, Basel, 1980.

[9] Claude Brezinski. *History of Continued Fractions and Padé Approximants*. Springer-Verlag, Berlin, 1991.

[10] Claude Brezinski. Convergence acceleration during the 20th century. *J. Comput. Appl. Math.*, 122:1–21, 2000.

[11] Claude Brezinski and Michela Redivo-Zaglia. *Extrapolation Methods*. North-Holland, Amsterdam, 1991.

[12] E. W. Cheney. *Introduction to Approximation Theory*. McGraw-Hill, New York, NY, 1966.

[13] C. W. Clenshaw. A note on summation of Chebyshev series. *Math. Tables Aids Comput.*, 9:118–120, 1955.

[14] R. M. Corless, Gaston H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and Donald H. Knuth. On the Lambert W function. *Adv. Comput. Math.*, 5:329–359, 1996.

[15] R. Courant. *Differential and Integral Calculus*, volume I. Blackie & Son, London, 1934. Reprinted 1988 in Classics Library, John Wiley.

[16] Germund Dahlquist. On summation formulas due to Plana, Lindelöf and Abel, and related Gauss–Christoffel rules II. *BIT*, 37:4:804–832, 1997.

[17] Peter Deuflhard and Andreas Hohmann. *Numerical Analysis in Modern Scientific Computing.* Springer, Berlin, second edition, 2003.

[18] G. Frobenius. Über Relationen zwischen den Näherungsbrüchen von Potenzreihen. *J. für Math.*, 90:1–17, 1881.

[19] Carl-Erik Fröberg. *Lärobok i Numerisk Analys.* Svenska Bokförlaget/Bonniers, Stockholm, 1962. In Swedish.

[20] Carl-Erik Fröberg. *Numerical Mathematics.* Benjamin/Cummings, Menlo Park, CA, 1980.

[21] H. H. Goldstine. *The Computer from Pascal to von Neumann.* Princeton University Press, Princeton, NJ, 1972.

[22] W. B. Gragg. The Padé table and its relation to certain algorithms of numerical analysis. *SIAM Review*, 14:1–62, 1972.

[23] Peter Henrici. *Discrete Variable Methods in Ordinary Differential Equations.* John Wiley, New York, 1962.

[24] Peter Henrici. *Elements of Numerical Analysis.* John Wiley, New York, 1964.

[25] Peter Henrici. *Applied and Computational Complex Analysis. Volume 1 Power Series, Integration, Conformal Mapping, Location of Zeros.* Wiley Classics Library, New York, 1988. Reprint of the 1974 original.

[26] Peter Henrici. *Applied and Computational Complex Analysis. Volume 2 Special Functions, Integral Transforms, Asymptotics, Continued Fractions.* Wiley Classics Library, New York, 1991. Reprint of the 1977 original.

[27] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, second edition, 2002.

[28] Felix Klein. *Elementary Mathematics from an Advanced Standpoint.* Dover, New York, 1945. Translation from German original, 1924.

[29] Donald E. Knuth. *The Art of Computer Programming, Vol. 2. Seminumerical Algorithms.* Addison-Wesley, Reading, MA, second edition, 1997.

[30] N. N. Lebedev. *Special Functions and Their Applications.* Dover, New York, 1972. Translation from Russian original.

[31] C. C. Lin and L. A. Segel. *Mathematics Applied to Deterministic Problems in the Natural Sciences*. Macmillan, New York, 1974.

[32] James N. Lyness and Cleve B. Moler. Numerical differentiation of analytic functions. *SIAM J. Numer. Anal.*, 4:202–210, 1967.

[33] N. E. Nörlund. *Vorlesungen über Differenzenrechnung*. Springer-Verlag, Berlin, 1924. Reprinted by Chelsea, New York, 1954.

[34] H. Padé. Sur la représentation approcheée d'un fonction par des fraction rationelles. *Thesis Anal. Ecole Norm. Sup..*, 3:1–93, supplement, 1892.

[35] O. Perron. *Die Lehre von den Kettenbrüchen. Vol. II*. Teubner, Stuttgart, third edition, 1957.

[36] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in Fortran; The Art of Scientific Computing*. Cambridge University Press, Cambridge, GB, second edition, 1992.

[37] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*. Progr. Math. 126, Birkhäuser, Boston, MA, second edition, 1994.

[38] Heinz Rutishauser. Der Quotienten-Differenzen-Algoritmus. *Z. Angew. Math. Phys.*, 5:233–251, 1954.

[39] D. Shanks. Nonlinear transformations of divergent and slowly convergent sequences. *J. Math. Phys.*, 34:1–42, 1955.

[40] Avram Sidi. *Practical Extrapolation Methods. Theory and Applications*. Cambridge University Press, Cambridge, UK, 2003.

[41] A. Smoktunowicz. Backward stability of Clenshaw's algorithm. *BIT*, 42:3:600–610, 2002.

[42] J. F. Steffensen. *Interpolation*. Chelsea, New York, second edition, 1950.

[43] Frank Stenger. *Numerical Methods Based on Sinc and Analytic Functions*. Springer-Verlag, Berlin, 1993.

[44] Gilbert Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, Wellesley, MA, 1986.

[45] E. C. Titchmarsh. *The Theory of Functions*. Oxford University Press, London, second edition, 1939.

[46] John Todd. Notes on numerical analysis i, solution of differential equations by recurrence relations. *Math. Tables Aids Comput.*, 4:39–44, 1950.

[47] H. S. Wall. *Analytic Theory of Continued Fractions*. Van Nostrand, Princeton, NJ, 1948.

[48] Peter Wynn. On a device for computing the $e_m(s_n)$ transformation. *Math. Tables Aids Comput.*, 10:91–96, 1956.

Index

acceleration
  Aitken, 150
acceleration of convergence, 116–161
Adams formula
  explicit, 110
  implicit, 110
Aitken acceleration, 118–123, 150
  a-version, 119
  iterated, 119
algorithm
  Euler's transformation, 127
aliasing, 63
alternating series, 6
Ansatz, 100
asymptotic series, 60

band matrix, 97
bell sum, 54, 56, 65, 114, 159
Bernoulli
  function, 135
  numbers, 12–13, 28
  polynomial, 135
Bernstein's approximation theorem, 47
Bessel functions, 57
Bickley's table, 78
boundary value problem, 64
boundary values)
  for difference equation, 97

Cauchy product, 8
Cauchy–FFT method, 38–43
characteristic equation (polynomial)
  of difference equation, 98
Chebyshev
  expansion, 46
  points, 45
  polynomial, 44–49
    minimax property, 45
  series, 44–49
Clenshaw's algorithm, 48, 64
completely monotonic
  function, 133
  sequence, 52
continued fractions, 161–169
convergence
  acceleration of, 116–161
convergence acceleration, 117–148
  problems, 148
convolution, 8
cubic spline, 112
cut (in the complex plane), 37

difference
  checks, 69
  equation
    linear, 96–101
    nonhomogeneous, 100
  of product, 72
  operator, 67–101
    backward, 68
    forward, 67
  scheme, 68
differential equation, 2
differentiation
  algorithmic, 18
  automatic, 18
  numerical, 92
  symbolic, 18
differentiation formula
  forwards, 84
  higher derivatives, 84
  backwards, 82
Discrete Fourier Transform (DFT), 40
divergent series, 58–61

epsilon algorithm, 123, 174–176
equivalence transformation, 163
error estimate
  asymptotic, 84
error function, 56
Euler
  numbers, 13
Euler transformation, 58
  generalized, 125
Euler's function, 58
Euler's method, 145
Euler's transformation, 124–131
exponential integral, 168

FFT, 40, 113
Fibonacci sequence, 113, 150
Fourier series, 36–38
frozen coefficients, 105, 116
function
  of bounded variation, 38
functionals, 75

Gamma function, 53, 140, 155
  incomplete, 168
Gauss' hypergeometric function, 10, 167
generating function, 101
geometric series, 9
  comparison with, 3
Green's function, 113

Hankel
  determinant, 178
  determinants, 123
  matrix, 172, 176–180
Hilbert matrix, 173

ill-conditioned series, 52–58
integration by parts, 60
  repeated, 8
interpolation formula, 89
irregular errors, 70

J. C. P. Miller formula, 29
Jacobi's identity, 178

Kummer's hypergeometric function, 11

Lagrange's
  remainder term, 8
Laurent series, 36–38
Lin–Segel's balancing procedure, 50
linear functionals, 74
linear operator, 74

matrix
  nilpotent, 19
  semicirculant, 19
  shift, 19
  Toeplitz, 19
matrix representation
  of truncated expansion, 84
  of truncated power series, 19
maximum modulus, 8, 39
minimax property, 45
multi-valued functions, 37

Neville's algorithm, 141–148
normal probability function, 168
Numerov's method, 104, 145, 157

operator
  averaging, 73
  calculus of, 72–96
  central difference, 73
  commutative, 74
  differentiation, 73
  expansions, 67–101
  linear, 74

Padé
  approximation, 169–173
    of exponential function, 170, 173
  table, 169–173
Pascal's triangle, 109
Peano kernel, 84
perturbation
  expansion, 49–51
    regular, 49
  singular, 50
Poisson
  distribution, 55
  summation formula, 117
Poisson distribution, 159, 168

power series, 7–26
    composite function, 17, 22
    division., 12
    inverse function, 22
    reversion, 22

qd algorithm, 176–180
qd scheme, 179

recurrence
    backward, 97
    forward, 97
recurrence relation
    three term, 48
remainder term
    in series, 6
    Lagrange's, 8
repeated averaging, 124
resolve
    grid, 84
Richardson extrapolation, 141–148
    repeated , 141
Riemann–Lebesgue theorem, 38
Romberg's method, 141
root condition, 102
Runge's 2nd order method, 145

Scylla and Charybdis, 40, 42
secant method, 113
seed, 96, 104
semicirculant matrix, 19
semiconvergent series, 58
sequence
    completely monotonic, 52
series
    alternating, 6
    asymptotic, 60
    convergence acceleration, 116–161
    divergent, 58–61
    geometric, 9
    ill-conditioned, 52–58
    semiconvergent, 58, 65
    Taylor, 7
    with positive terms, 58
shift matrix, 19
shift operator, 67
Stirling's formula, 53, 139
subtabulation, 111
summation algorithms, 116–161
summation by parts, 72
    repeated, 107
superposition principle, 69

tail of a series, 3
Taylor series, 7
    symbolic form of, 76
termination criterion, 3
thinned sequence, 121
thinning, 121, 129–131
    geometric, 121
three term recurrence relation, 48
Toeplitz matrix, 19
    method, 19
translation operator, 67
trapezoidal error, 141
tridiagonal, 97

z-transform, 101

Contents

4 Interpolation and Approximation **1**

4.1 The Interpolation Problem 1
4.1.1 Introduction 1
4.1.2 Various Bases for $\mathcal{P}_n$ 4
4.1.3 Discrete Least Squares Approximation 6
4.1.4 The Runge Phenomenon 7
Review Questions 9
Problems and Computer Exercises 10
4.2 Interpolation Formulas and Algorithms 11
4.2.1 Newton's Interpolation Formula 11
4.2.2 Lagrange's Interpolation Formula 19
4.2.3 Iterative Linear Interpolation 24
4.2.4 Conditioning of the Interpolation Problem 25
4.2.5 Interpolation by Rational Functions 27
Review Questions 31
Problems and Computer Exercises 31
4.3 Generalizations and Applications 33
4.3.1 Interpolation using Values of Derivatives 33
4.3.2 Inverse interpolation 38
4.3.3 Numerical differentiation 39
4.3.4 Fast Algorithms for Vandermonde Systems 41
4.3.5 Multidimensional Interpolation 45
Review Questions 46
Problems and Computer Exercises 47
4.4 Piecewise Polynomial Interpolation 48
4.4.1 Bernstein Polynomials 48
4.4.2 Parametric Bézier Curves 50
4.4.3 Splines 54
4.4.4 Cubic Spline Interpolation 58
4.4.5 Computing with B-Splines 65
Review Questions 75
Problems and Computer Exercises 76
4.5 Approximation and Function Spaces 78
4.5.1 Distance and Norm 79

4.5.2 Operator Norms and the Distance Formula 83
4.5.3 Inner Product Spaces and Orthogonal Systems . . . 89
4.5.4 Solution of the Approximation Problem 92
4.5.5 Orthogonal Polynomials and Least Squares Approximation 95
4.5.6 Statistical Aspects of the Method of Least Squares . 103
Review Questions 106
Problems and Computer Exercises 106
4.6 Trigonometric Interpolation and Fourier Transforms 108
4.6.1 Basic Formulas and Theorems 110
4.6.2 Periodic Continuation of a Function 116
4.6.3 The Fourier Integral Theorem 117
4.6.4 Sampled Data and Aliasing 120
Review Questions 122
Problems and Computer Exercises 122
4.7 The Fast Fourier Transform 124
4.7.1 The Fast Fourier Algorithm 124
4.7.2 FFTs and Discrete Convolutions 130
4.7.3 Real Data and Fast Trigonometric Transforms . . . 131
4.7.4 The General Case FFT 134
Review Questions 135
Problems and Computer Exercises 136
4.8 Complex Analysis in Interpolation 138
4.8.1 Interpolation of Analytic Functions 138
4.8.2 Analysis of a Generalized Runge Phenomenon . . . 140
4.8.3 The Sampling Theorem 145
Problems and Computer Exercises 146

Bibliography **151**

Index **155**

Chapter 4

Interpolation and Approximation

4.1 The Interpolation Problem

4.1.1 Introduction

Polynomials are used as the basic means of approximation in nearly all areas of numerical analysis. We have previously encountered two types of interpolation

- Piecewise linear interpolation that is commonly used in tables, when the requirements of accuracy are modest. A more modern application is in Computer Graphics.
- Interpolation of the values of a function in n *equidistant* points by a function in $\mathcal{P}_n$. Recall that in Sec. 3.2.2, $\mathcal{P}_n$ was defined as the space of polynomials in one variable of degree *less than* n; n is the number of data required to specify a polynomial in $\mathcal{P}_n$; the dimension of the linear space $\mathcal{P}_n$ is n.[1]

  The formulas with equidistant points, in particular Stirling's interpolation formula, given in Sec. 3.2, is mainly used piecewise for small values of n; see Problem 2.

The first type of interpolation will be generalized in Sec. 4.4, where we shall study interpolation by **piecewise polynomials**. Of particular importance are **splines**, which are piecewise polynomials, where a few derivatives are required to be continuous at the joints of the pieces.

In the first two sections we shall go deeper into the following **polynomial interpolation problem** for *non-equidistant*, distinct points:

Find a polynomial $p \in \mathcal{P}_n$ such that

$$p(x_i) = f(x_i), \quad i = 1 : n, \quad x_i \neq x_j \text{ for } i \neq j. \tag{4.1.1}$$

Recall that, by Theorem 3.2.1, the interpolation polynomial $p(x)$ *is uniquely determined* for a given **grid**, $(x_1, x_2, \ldots, x_n)$. This theorem is general, although the rest

[1] Some authors use similar notations, e.g., $\mathbf{P}_n$ or Π_n, to denote the $n+1$-dimensional space of polynomials of degree *less than or equal to* n.

of Sec. 3.2 dealt with interpolation polynomials in the equidistant case only, and their application to numerical differentiation and integration.[2] Also note that the formulation and the solution of this problem are *independent of the ordering of the points* x_i.

A set of polynomials $\mathbf{p} = \{p_1(x), p_2(x), \ldots, p_n(x)\}$, such that any polynomial $p \in \mathcal{P}_n$ can be expressed as a linear combination

$$p(x) = c_1 p_1(x) + \ldots + c_m p_n(x),$$

is called a **basis** in $\mathcal{P}_n$. The column vector $c = (c_1, c_2, \ldots c_n)^T$ can be viewed as a *coordinate vector* of p in the space $\mathcal{P}_n$, with respect to this basis. The **power basis**, where $p_j(x) = x^{j-1}$, i.e.

$$p(x) = \sum_{j=1}^{n} c_j x^{j-1},$$

is the simplest basis, though not always the best.

The interpolation problem (4.1.1) leads to a linear system of equations

$$c_1 p_1(x_i) + c_2 p_2(x_i) + \ldots + c_m p_n(x_i) = f(x_i), \quad i = 1 : n. \tag{4.1.2}$$

If we introduce the matrix

$$M_{\mathbf{p}} = [p_j(x_i)]_{i,j=1}^{n}, \tag{4.1.3}$$

and the column vector $\tilde{f} = \big(f(x_1), f(x_2), \ldots, f(x_n)\big)^T$,[3] then the linear system becomes

$$M_{\mathbf{p}} c = \tilde{f}. \tag{4.1.4}$$

The proof of Theorem 3.2.1 was based on the fact that this matrix is non-singular in the case of the power basis; in this case $M_{\mathbf{p}} = V^T$, where V is the **Vandermonde matrix**[4]

$$V = [x_j^{i-1}]_{i,j=1}^{n} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & \cdots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{pmatrix}. \tag{4.1.5}$$

In any basis $\mathbf{q} = \{q_1(x), q_2(x), \ldots, q_n(x)\}$ for $\mathcal{P}_n$, the q_j must be linear combinations of the p_k, $k = 1 : n$. This can be expressed in vector-matrix form:

$$\big(q_1(x), q_2(x), \ldots, q_n(x)\big) = \big(p_1(x), p_2(x), \ldots, p_n(x)\big) A, \tag{4.1.6}$$

where A is a constant matrix. *A must be non-singular*; for, if A were singular then there would exist a non-trivial vector v such that $Av = 0$, hence

$$(q_1(x), q_2(x), \ldots, q_n(x))v = (p_1(x), p_2(x), \ldots, p_n(x))Av = 0 \quad \forall x,$$

[2] It is de facto so, although the polynomials are invisible in the derivations of formulas by operator methods.

[3] We try to make a distinction between f that is an element in some function space and $\tilde{f} \in \mathbf{R}^n$.

[4] Alexandre Théophile Vandermonde (1735–1796), member of the French Academy of Sciences, is regarded as the founder of the theory of determinants. What is now referred to as the "Vandermonde matrix" does not seem to appear in his writings!

and $(q_1(x), q_2(x), \dots, q_n(x))$ would thus not be a basis.

Similarly set $M_{\mathbf{q}} = [q_j(x_i)]_{i,j=1}^n$. By putting $x = x_i$, $i = 1 : m$ into (4.1.6), we see that $M_{\mathbf{q}} = M_{\mathbf{p}}A$, and $M_{\mathbf{q}}$ is non-singular for every basis. If we set $p(x) = \sum d_j q_j(x)$, the system (4.1.2) becomes for this basis $M_{\mathbf{q}}d = \tilde{f}$. Then

$$M_{\mathbf{p}}c = \tilde{f} = M_{\mathbf{q}}d = M_{\mathbf{p}}Ad, \qquad c = M_{\mathbf{p}}^{-1}\tilde{f} = Ad. \tag{4.1.7}$$

the matrix A is thus like a coordinate transformation in Geometry; Gander [25] gives the matrix A for the transformation between various several common representations.

Example 4.1.1 (An application to numerical integration).

We shall find a formula for integrals of the form

$$I = \int_0^1 x^{-1/2} f(x)\, dx$$

that is exact for $f \in \mathcal{P}_m$ and uses the values $f(x_i)$, $i = 1 : m$. Such integrals need a special treatment, due to the integrable singularity at $x = 0$.

Set $\mu_j = \int_0^1 x^{-1/2} p_j(x)\, dx$, and introduce the row vector $\mu^A = (\mu_1, \mu_2, \dots, \mu_n)$. Then

$$I \approx \int_0^1 x^{-1/2} p(x)\, dx = \sum_{j=1}^n c_j \mu_j = \mu^A c = \mu^A(p) M_{\mathbf{p}}^{-1} \tilde{f}, \tag{4.1.8}$$

where it is emphasized in the last expression that μ^A depends on the basis. In fact $\mu^A(q) = \mu^A(p)A$, and $M_{\mathbf{q}}^{-1} = A^{-1}M_{\mathbf{p}}^{-1}$; we see that the approximation to I is independent of the basis, as it should, in view of Theorem 3.2.1.

Another approach is the **method of undetermined coefficients**, i.e. to seek a formula

$$I \approx \sum_{i=1}^n b_i f(x_i) \equiv b^T \tilde{f},$$

that is exact when $f(x) = p_j(x)$, $j = 1 : m$; then it is exact for all $p \in \mathcal{P}_m$ These conditions lead to the linear system

$$M_{\mathbf{p}}^T b = \mu. \tag{4.1.9}$$

This may be called the *adjoint* or dual to the system $M_{\mathbf{p}}c = \tilde{f}$. Using the standard basis it reads

$$Vb = \mu.$$

From (4.1.9) we get $b^T = \mu^T M_{\mathbf{p}}^{-1}$, and the final result of this approach becomes

$$I \approx b^T \tilde{f} = \mu(p)^T M_{\mathbf{p}}^{-1} \tilde{f},$$

which is *the same as* (4.1.8), although interpolation was not mentioned in this approach. In view of Theorem (3.2.1) this is no surprise, since the same values of f are used, and both formulas are exact for all $f \in \mathcal{P}_m$.

For numerical applications (for the power basis) see Problem 2. Evidently these two approaches can be used for any linear functional of f.

4.1.2 Various Bases for $\mathcal{P}_n$

There are many ways of specifying polynomials. If the purpose is to compute derivatives or integrals of the interpolation polynomial, the power basis or the **the shifted power basis**, where

$$q_j(x) = (x-a)^{j-1},$$

are usually also convenient. If a shifted power basis is to be used for polynomial approximation on a certain interval, it is often best to choose a near the midpoint of the interval.

The power basis has a bad reputation, which is related the ill-conditioning of the corresponding Vandermonde matrices. Many bounds and asymptotic estimates for the condition number of the Vandermonde matrix

$$V = V(x_1, x_2, \ldots, x_n)$$

are known; see [26, Sec. 1.3], [31, Sec. 22.1]. For example, for equidistant points on $[-1.1]$, i.e. $x_i = -1 + 2(i-1)/(n-1)$, it holds that

$$\kappa_\infty(V) = \|V^{-1}\|_\infty \|V\|_\infty \sim \pi^{-1} e^{\pi/4} (3.1)^n.$$

Hence, for $n = 20$, $\kappa_\infty(V) \approx 1.05 \cdot 10^9$. Other point distributions are even worse, e.g., for the harmonic points $x_i = 1/i$, $i = 1:n$,

$$\kappa_\infty(V) > n^{n+1},$$

which is faster than exponential growth! For the **Chebyshev points** on $[-1, 1]$

$$x_i = \cos\Big(\frac{2i-1}{n}\frac{\pi}{2}\Big), \quad i = 1:n, \tag{4.1.10}$$

i.e. the zeros of $T_{n-1}(x)$, the Vandermonde matrix is better conditioned

$$\kappa_\infty(V) \sim 0.253^{3/4}(1+\sqrt{2})^n.$$

It should be stressed that the condition number of the Vandermonde matrix measures the sensitivity of the coefficients c_i in the polynomial $p(x) = \sum_{j=1}^n c_j x^{j-1}$ to perturbations in the given data f_i. It is possible, that even when these coefficients are inaccurately determined, the interpolation polynomial $p(x)$ does still reproduce the true interpolation polynomial well. For further discussion of these points, see Sec. 4.2.4 and Sec. 4.3.4.

Mathematically, the choice of basis (for a finite-dimensional space) makes no difference. Computationally, working with *rounded values of the coefficients*, the choice of basis can make a great difference. Consider a sequence of polynomials $q_1, q_2, q_3, \ldots$

$$\begin{aligned}
q_1(x) &= a_{11} \\
q_2(x) &= a_{12} + a_{22}x \\
q_3(x) &= a_{13} + a_{23}x + a_{33}x^2 \\
&\ldots \\
q_n(x) &= a_{1m} + a_{2m}x + a_{3m}x^2 + \ldots + a_{mm}x^{n-1} \\
&\ldots
\end{aligned}$$

where $a_{jj} \neq 0$ for all j, is defined to be a **triangle family** of polynomials, i.e. $q_j(x)$ is of $(j-1)$'th degree with a non-zero leading coefficient.[5].

Conversely, for any j, $p_j(x) = x^{j-1}$ can be expressed recursively and uniquely as linear combinations of $q_1(x), q_2(x), \ldots, q_j(x)$, so that we obtain a triangular scheme also for the inverse transformation. So *every triangle family* $\{q_1(x), q_2(x), \ldots, q_m(x)\}$ *is a basis for* $\mathcal{P}_m$.

What we has just seen, is indeed a proof of the well known fact that the inverse of a triangular matrix, (with no zeros in the main diagonal) is also triangular. Among interesting triangle families we can mention the families where $q_{j+1}(x)$ is defined by $(x-c)^j$, $T_j(x)$, and many other families of orthogonal polynomials.

Let $x_1, x_2, \ldots, x_n$ be n distinct points and consider the **Newton polynomials**

$$p_1(x) = 1, \quad p_j(x) = (x - x_1)(x - x_2)\ldots(x - x_{j-1}), \quad j = 2 : n,$$

They define a triangle family with unit leading coefficients, and hence form a basis. The representation

$$p(x) = c_1 p_1 + c_2 p_2(x) + c_3 p_3(x) + \cdots + c_n p_n(x) \tag{4.1.11}$$

is often very convenient. Since $p_j(x_k) = 0$, if $k < j$, the coefficients $c_1, c_2, \ldots, c_n$ satisfy the lower triangular system of equations $Lc = f$, where

$$L = \begin{pmatrix} 1 & & & & \\ 1 & (x_2 - x_1) & & & \\ 1 & (x_3 - x_1) & (x_3 - x_1)(x_3 - x_2) & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & (x_n - x_1) & (x_n - x_1)(x_n - x_2) & \cdots & \prod_{j=1}^{n-1}(x_n - x_j) \end{pmatrix} \tag{4.1.12}$$

Hence the coefficients can be computed recursively, by forward substitution, with much less work than the linear system (4.1.4) would require for the power basis, by standard methods for linear algebra. In the next section we shall see how this basis leads to Newton's interpolation formula, which is one of the best interpolation formulas, with respect to flexibility, computational economy and numerical stability.

If a polynomial $p(x)$ is given in the form (4.1.11) then it can be evaluated using only n multiplications and $2n$ additions, for a given numerical value x, from the nested form

$$\begin{aligned} p(x) = (\cdots(c_m(x - x_{n-1}) + c_{n-1})(x - x_{n-2}) + \\ \cdots + c_3)(x - x_2) + c_2)(x - x_1) + c_1. \end{aligned}$$

(Notice that the case, where all the $x_i = 0$, gives Horner's rule, see Sec. 1.4.2. We have $p(x) = b_1$, where b_1 is computed using the recursion formula:

$$b_n := c_n, \quad b_{i-1} := b_i(x - x_{i-1}) + c_{i-1}, \quad i = n : -1 : 2. \tag{4.1.13}$$

We leave the proof to Problem 4.

[5]In the terminology used in the previous subsection, this triangular matrix equals A^T; this explains the notation for the elements

Other bases are sometimes more advantageous. A **cardinal basis** of $\mathcal{P}_n$ is generated by the polynomial

$$\Phi_n(x) = (x - x_1)(x - x_2) \cdots (x - x_n), \tag{4.1.14}$$

where x_i, $i = 1 : n$, are distinct. The basis used in Lagrange interpolation formula reads

$$\ell_j(x) = \frac{\Phi_n(x)}{(x - x_j)\Phi_n'(x_j)}, \quad j = 1 : n. \tag{4.1.15}$$

Here ℓ_j is a polynomial of degree $n - 1$. By L'Hospital's rule

$$\ell_j(x_i) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{if } i \neq j. \end{cases} \tag{4.1.16}$$

(This is the property that in a more general context characterizes a cardinal basis.) This basis directly displays the solution of the interpolation problem for n distinct points. **Lagrange's interpolation** formula[6] has been widely regarded as being of mainly theoretical interest. In Sec. 4.2.2 two modified forms of Lagrange interpolation formula will be given, which are more computationally attractive.

The matrix approach described in the previous subsections may sometimes be convenient; a Vandermonde matrix is easily generated, when you work in a matrix-oriented command language. If you deal with a modest number of polynomials of low degree, convenience can be given a larger weight than the optimal number of arithmetical operations and a minimized effect of rounding errors. In the latter respects, the matrix approach is inferior to Newton's interpolation formula and the other formulas and algorithms to be discussed later.

The main reason why we started with such non-optimal procedures, is that they are *easily generalizable to other interpolation problems*, e.g., interpolation in other function spaces than $\mathcal{P}_n$ (see Problem 5), or interpolation with other conditions on the function f in addition to function values (see later sections). For such non-standard interpolation problems—that do occur in practice—the matrix approach is helpful also for finding out under what conditions the problem has a unique solution.

4.1.3 Discrete Least Squares Approximation

Let $p_1(x), p_2(x), \ldots, p_n(x)$ be a basis for $\mathcal{P}_n$. A natural extension of the interpolation problem is to determine a polynomial

$$p(x) = \sum_{j=1}^{n} c_j p_j(x) \in \mathcal{P}_n,$$

that, in some sense, best fits to the data $(x_i, f(x_i))$, $i = 1 : m$, where $m > n$. Since the number of equations is larger than the number of parameters, the corresponding linear system $Mc = f$ is **overdetermined**. and can typically be satisfied only

[6] Lagrange published his interpolation formula in 1794.

approximately; see Example 1.2.5, where a straight line could not be made to pass through the five points.

In **discrete least squares approximation** on determines *the coefficient vector c that minimizes the sum of squared residuals*

$$S(c) = \sum_{i=1}^{m} (p(x_i) - f(x_i))^2. \tag{4.1.17}$$

This can in many applications be motivated by statistical arguments; see Theorem 4.5.19. It also leads to rather simple computations. The conditions for the minimization are

$$\frac{\partial S(c)}{\partial c_k} = 2 \sum_{i=1}^{m} p_k(x_i)\big(p(x_i) - f(x_i)\big) = 0, \quad k = 1 : n.$$

These conditions can be written in the form $M^T(Mc - f) = 0$, or

$$M^T M c = M^T f, \tag{4.1.18}$$

Here $M^T M$ is a symmetric $n \times n$ matrix and (4.1.18) is called the **normal equations**; see Sec. 1.6.5. It can be shown that the matrix $M^T M$ is non-singular, and that the system yields the minimum of $S(c)$, unless the columns of M are linearly dependent.

Overdetermination can be used to attain two different types of **smoothing**:

(a) to reduce the effect of random or other irregular errors in the values of the function;

(b) to give the polynomial a smoother behaviour between the grid points.

Note that interpolation is a special case ($n = m$) of this problem. In this case the normal equations are mathematically equivalent to the system $Mc = f$. Since the condition number of $M^T M$ is the square of the condition number of M, the matrix $M^T M$ often is very ill-conditioned even for moderate n. Therefore forming the normal equations cannot be recommended in general. A stable method for discrete least squares polynomial approximation is obtained by using a basis of orthogonal polynomials; see Sec. 4.5.5. Stable methods for more general least squares problems are treated in Volume II, Chapter 8.

4.1.4 The Runge Phenomenon

Equidistant interpolation can give rise to convergence difficulties when the number of interpolation points becomes large. This difficulty is often referred to as **Runge's phenomenon**[7], and we illustrate it in the following example. A more advanced discussion is given in Sec. 4.4.3, by means of complex analysis.

[7] Carl Runge (1856–1927) German mathematician, who held a chair in Applied Mathematics in Göttingen 1904–1925. Runge's example is from 1901.

Example 4.1.2. The graph of the function

$$f = \frac{1}{1+25x^2} = \frac{i}{2}\Big(\frac{1}{i+5x} + \frac{1}{i-5x}\Big),$$

where $i = \sqrt{-1}$, is the continuous curve shown in Figure 4.2.1, is approximated in two different ways by a polynomial of degree 10 in $[-1, 1]$.

The dashed curve has been determined by interpolation on the equidistant grid with $m = 11$ points

$$x_i = -1 + 2(i-1)/(m-1), \qquad i = 1 : m. \tag{4.1.19}$$

The graph of the polynomial so obtained has—unlike the graph of f—a disturbing course between the grid points. The agreement with f near the ends of the interval is especially bad, while near the center of the interval $[-\frac{1}{5}, \frac{1}{5}]$ the agreement is fairly good. Such behavior is typical of *equidistant interpolation of fairly high degree*, and can be explained theoretically.

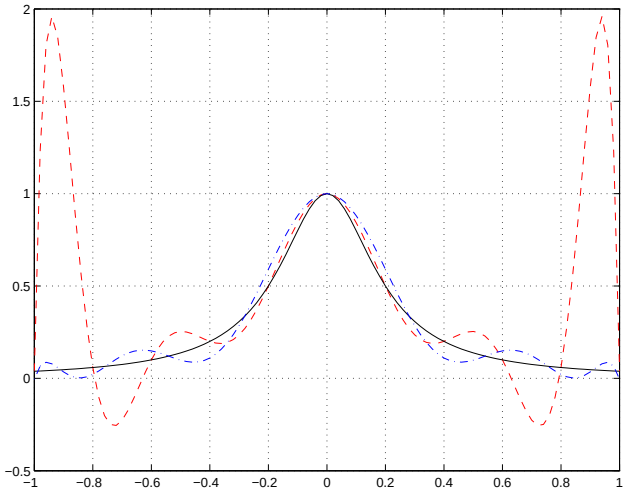


Figure 4.1.1. *Polynomial interpolation of* $1/(1+25x^2)$ *in two ways by the use of 11 points: equidistant points (dashed curve), Chebyshev abscissae (dash-dot curve).*

The *dotted* curve in Figure 4.1.1 has been determined by interpolation at the Chebyshev points

$$x_i = \cos\Big(\frac{2i-1}{m}\frac{\pi}{2}\Big), \quad i = 1 : m, \tag{4.1.20}$$

$(m = 11)$. This procedure is studied more closely in a later section. The agreement with f is now much better than with equidistant interpolation, but still not good. The function is not at all suited for approximation by one polynomial over the

entire interval. Here one would get a much better result using approximation with piecewise polynomials; see Sec. 4.4.

Notice that the difference between the values of the two polynomials is much smaller at the grid points of the equidistant grid than in certain points between the grid points, especially in the outer parts of the interval. This intimates that the values which one gets by equidistant interpolation with a polynomial of high degree can be very sensitive to disturbances in the given values of the function. Put another way, *equidistant interpolation using polynomials of high degree is in some cases an ill-conditioned problem, especially in the outer parts of the interval* $[x_1, x_m]$. The effect is even worse if one extrapolates—i.e. if one computes values of the polynomial outside the grid. However, equidistant interpolation works well near the center of the interval.

Even with equidistant data one can often get a more well-behaved curve by—instead of interpolating—fitting a polynomial of lower degree using the method of least squares. Generally, if one chooses $n < 2\sqrt{m}$, then the polynomial fit is quite well conditioned, but higher values of n should be avoided. In the above example, however, the agreement would still be quite bad, even at the grid points, when the degree is chosen to be so low.

If one intends to approximate a function *in the entire interval* $[-1, 1]$ by a *polynomial* and can choose the points at which the function is computed or measured, then one should choose the Chebyshev points. Using these points, interpolation is a fairly *well-conditioned* problem in the entire interval. The risk of disturbing surprises between the grid points is insignificant. One can also conveniently fit a polynomial of lower degree than $n - 1$, if one wishes to smooth errors in measurement; see Sec. 4.5.5.

Example 4.1.2 shows how important it is to study the course of the approximating curve $p^*(x)$ between the points which are used in the calculation before one accepts the approximation. When one uses procedures for approximation for which one does not have a complete theoretical analysis, one should make an *experimental perturbational calculation*. In the above case such a calculation would very probably reveal that the interpolation polynomial reacts quite strongly if the values of the function are disturbed by small amounts, say $\pm 10^{-3}$. This would give a basis for rejecting the unpleasant dashed curve in the example, even if one knew nothing more about the function than its values at the equidistant grid points.

Review Questions

1. The interpolation problem in $\mathcal{P}_n$ leads to a linear system $V^T c = \tilde{f}$, where V is a Vandermonde matrix. Write down the expression for the element v_{ij}.
2. What is meant by the method of undetermined coefficients? Give an example!
3. What is meant by a triangle family $q_1(x), q_2(x), \ldots, q_n(x)$ of polynomials? Are all such families a basis for $\mathcal{P}_n$?
4. What property characterizes a cardinal basis for $\mathcal{P}_n$?

Problems and Computer Exercises

1. (a) Study experimentally interpolation in $\mathcal{P}_n$, $n = 2 : 2 : 16$ for $f(x) = (3+x)^{-1}$, $x \in [-1, 1]$. Use the linear system $V^T c = \tilde{f}$ and the power basis. Study both equidistant points and Chebyshev points

$$x_i = -1 + 2\frac{i-1}{n-1}, \quad x_i = \cos\Big(\frac{2i-1}{n}\frac{\pi}{2}\Big), \quad i = 1 : n$$

respectively. Plot the error curve, $y = |f(x) - p(x)|$ in semi-logarithmic scale. For the larger values of m, make also experiments to illuminate the effects from random perturbations of the function values to the values of $p(x)$.

(b) Make also a few experiments with a random vector $\tilde{f}$, for $n = 16$ and $n = 8$, in order to compare the grid data and the order of magnitude of $p(x)$ between the grid points.

2. Prove the validity of (4.1.13).

3. *A warning for polynomial extrapolation of empirical functions, or . . . ?*

(a) Write a program $c = \mathit{polyapp}(x, y, n)$ that finds the coefficient vector c for a polynomial in $p \in \mathcal{P}_n$, in a shifted power basis, such that $y_i \approx p(x_i)$, $i = 1 : m$, $m \geq n$, in the least squares sense, or study a program that does almost this.[8]

(b) The following data shows the development of the Swedish GDP, quoted (with permission) from a table made by a group associated with the Swedish Employer's Confederation. (The data are expressed in prices of 1985 and scaled so that the value for 1950 is 100.)

1950	1955	1960	1965	1970	1975	1980	1985	1990
100.0	117.7	139.3	179.3	219.3	249.1	267.5	291.5	326.4
1952	1957	1962	1967	1972	1977	1982	1987	1992
104.5	124.6	153.5	189.2	226.4	247.7	270.2	307.6	316.6

(a) For the upper pairs of data, compute and plot $p(x), x \in [1950, 2000]$ (say). Mark the given data points. Do this for $m = 9$, and for (say) $n = 9$, and then for $n = 8 : -2 : 2$. Store the results, so that comparisons can be made afterwards.

Hint: If you use `polyfit` first subtract 1970 from the years.

(b) Do the same for the lower pairs of data. Organize the plots, so that interesting comparisons become convenient, e.g. how well were the data points of one of the sets interpolated by the results from the other set?

(c) Make forecasts for 1995 and 2000 with both data sets. Then, use a reduced data set, e.g., for the years 1982 and earlier (so that $m = 7$), and and compare the forecasts for 1987 and 1992 with the given data. (Isn't it a reasonable test for every suggested forecast model to study its ability to predict the present

[8] The Matlab command `polyfit` does *almost* this.

from the past?)

(d) See if you obtain better results with the logarithms of the GDP values.

4.2 Interpolation Formulas and Algorithms

4.2.1 Newton's Interpolation Formula

Let $x_1, x_2, \ldots, x_n$ be n distinct points and let p^* in $\mathcal{P}_n$ be the unique solution of the interpolation problem for $f(x)$ with the basis of (4.1.11). Suppose that

$$\begin{aligned} f(x) = c_1 + c_2(x - x_1) + \ldots + c_n(x - x_1)(x - x_2)\cdots(x - x_{n-1}) \\ + A_n(x)(x - x_1)(x - x_2)\cdots(x - x_n), \end{aligned} \tag{4.2.1}$$

If $f \in \mathcal{P}_n$, we know from Sec. 4.1.2 that such a formula holds with $A_n(x) \equiv 0$. We shall see that it is correct in general.

For $x = x_1$ we get $c_1 = f(x_1)$. Set

$$[x]f = f(x), \qquad [x_1, x]f = \frac{f(x) - f(x_1)}{x - x_1}.$$

Then

$$\begin{aligned} [x_1, x]f = c_2 + c_3(x - x_2) + \ldots + c_n(x - x_2)\cdots(x - x_{n-1}) \\ + A_n(x)(x - x_2)\cdots(x - x_n), \end{aligned}$$

and $c_2 = [x_1, x_2]f$. We now *define* recursively, for $k \geq 1$, **divided differences**[9]

$$[x_1, x_2, \ldots, x_{k-1}, x_k, x]f = \frac{[x_1, x_2, \ldots, x_{k-1}, x]f - [x_1, x_2, \ldots, x_{k-1}, x_k]f}{x - x_k}. \tag{4.2.2}$$

We obtain, for $k = 2$,

$$\begin{aligned} [x_1, x_2, x]f = c_3 + c_4(x - x_3) + \ldots + c_n(x - x_3)\cdots(x - x_{n-1}) \\ + A_n(x)(x - x_3)\cdots(x - x_n), \end{aligned}$$

and $c_3 = [x_1, x_2, x_3]f$. By induction it follows that

$$c_k = [x_1, x_2, \ldots, x_{k-1}, x_k]f, \qquad k = 1 : m. \tag{4.2.3}$$

For $k = n$ we obtain, $A_n(x) = [x_1, x_2, \ldots x_n, x]f$.

We now introduce a notation that is convenient in the following. Set $\Phi_0(x) = 1$, and for $k = 1 : n$,

$$\Phi_k(x) = \Phi_{k-1}(x)(x - x_k) = (x - x_1)(x - x_2)\cdots(x - x_k). \tag{4.2.4}$$

[9] We prefer the modern notation $[\ldots]f$ to the older notations $f[\ldots]$ or $f(\ldots)$, since it emphasizes that $[\ldots]$ is an operator. Note that the interpretation $[x]f = f(x)$ is consistent with this.

For $f \in \mathcal{P}_n$ we know that (4.2.1) is correct, hence we can trust the coefficients c_k. Moreover, since $p^*(x_j) = f(x_j)$, $j = 1 : n$, it follows that $[x_1, x_2, \dots x_j]p^* = [x_1, x_2, \dots x_j]f$. Hence

$$p(x) = \sum_{j=1}^{n} [x_1, \dots, x_j]p \, \Phi_{j-1}(x), \quad \forall p \in \mathcal{P}_n,$$
$$p^*(x) = \sum_{j=1}^{n} [x_1, \dots, x_j]f \, \Phi_{j-1}(x).$$

For a general function f we do not yet know that (4.2.1) is correct, but after inserting the only possible values of c_k and $A_n(x)$ in (4.2.1), we can conjecture that the following is an identity:

$$f(x) = \sum_{j=1}^{n} [x_1, \dots, x_j]f \, \Phi_{j-1}(x) + [x_1, x_2, \dots x_n, x]f \, \Phi_n(x).$$

We prove this by induction. For $n = 1$, it is true, because the right hand side becomes $f(x_1) + [x_1, x]f \cdot (x - x_1) = f(x) =$ the left hand side. Next suppose that it is true for $n = m$. The difference between the right hand side for $n = m + 1$ and $n = m$ reads

$$\begin{aligned}
&([x_1, \dots, x_{m+1}]f - [x_1, x_2, \dots x_m, x]f) \, \Phi_m(x) + [x_1, x_2, \dots x_{m+1}, x]f \, \Phi_{m+1}(x) \\
&= \big([x_1, \dots, x_{m+1}]f - [x_1, x_2, \dots x_m, x]f + [x_1, x_2, \dots x_{m+1}, x]f \ (x - x_{m+1})\big)\Phi_m(x) \\
&= \big([x_1, \dots, x_{m+1}, x]f \ (x_{m+1} - x) + [x_1, x_2, \dots x_{m+1}, x]f \ (x - x_{m+1})\big)\Phi_m(x) \\
&= 0.
\end{aligned}$$

Hence the conjecture is true for $n = m+1$. We summarize the results in a theorem.

Theorem 4.2.1. *Newton's Interpolation Formula with exact remainder.*

The interpolation problem of determining the polynomial $p \in \mathcal{P}_n$ such that $p(x_i) = f(x_i)$, $i = 1 : n$, where the x_i are distinct points, has the solution

$$p(x) = \sum_{j=1}^{n} [x_1, \dots, x_j]f \, \Phi_{j-1}(x). \tag{4.2.5}$$

where $\Phi_k(x)$ is defined by (4.2.4)*. The formula*

$$f(x) = \sum_{j=1}^{n} [x_1, \dots, x_j]f \, \Phi_{j-1}(x) + [x_1, x_2, \dots x_n, x]f \, \Phi_n(x) \tag{4.2.6}$$

is an identity, i.e. the exact remainder equals

$$f(x) - p(x) = [x_1, x_2, \dots x_n, x]f \, \Phi_n(x). \tag{4.2.7}$$

These formulas are valid also for complex x_i and x.

Note that to obtain the interpolation polynomial of the next higher degree with Newton's formula, we need only add a term similar to the last term, but involving a new divided difference of one higher order.

In particular, if $f \in \mathcal{P}_n$ then it follows from (4.2.7) that

$$[x_1, x_2, \ldots, x_n, x]f = 0, \quad \forall x.$$

For $x = x_{n+1}$, this equation is, by Theorem 3.2.1, the only non-trivial relation of the form $\sum_{j=1}^{n+1} a_j f(x_j) = 0$ that holds for all $f \in \mathcal{P}_n$,.

Theorem 4.2.2.

For every n, the divided difference $[x_1, x_2, \ldots, x_n]f$ is the coefficient of x^{n-1} in the interpolation polynomial $p^* \in \mathcal{P}_n$. A divided difference is a symmetric function of its arguments.

Proof. The first statement follows from (4.2.3). The second statement then holds, because the interpolation polynomial is uniquely determined and independent of how the points are ordered. □

Assume $k > i$. By the definition of divided differences,

$$[x_{i+1}, \ldots, x_{k-1}, x_k, x]f = \frac{[x_{i+1}, \ldots, x_{k-1}, x]f - [x_{i+1}, \ldots, x_{k-1}, x_k]f}{x - x_k}.$$

Now set $x = x_i$ and use the symmetry property (Theorem 4.2.2). We obtain the formula

$$[x_i, x_{i+1}, \ldots, x_{k-1}, x_k]f = \frac{[x_i, x_{i+1}, \ldots, x_{k-1}]f - [x_{i+1}, \ldots, x_{k-1}, x_k]f}{x_i - x_k}. \quad (4.2.8)$$

This formula can be used recursively to compute the divided differences. The computation is conveniently arranged in a table shown below for $n = 5$ (recall that $[x_i]f = f(x_i)$).

$$\begin{array}{lllll}
x_1 & [x_1]f & & & \\
 & & [x_1, x_2]f & & \\
x_2 & [x_2]f & & [x_1, x_2, x_3]f & \\
 & & [x_2, x_3]f & & [x_1, x_2, x_3, x_4]f \\
x_3 & [x_3]f & & [x_2, x_3, x_4]f & \\
 & & [x_3, x_4]f & & [x_2, x_3, x_4, x_5]f \\
x_4 & [x_4]f & & [x_3, x_4, x_5]f & \\
 & & [x_4, x_5]f & & \\
x_5 & [x_5]f & & &
\end{array} \qquad [x_1, x_2, x_3, x_4, x_5]f$$

This table is called a **divided-difference table**. Note that the points $x_1, x_2, x_3 \ldots$ need not be arranged in increasing order of magnitude. *Each entry in the table is computed from the two entries above and below in the previous column.* Hence the complete table can be constructed, e.g., column by column or diagonal by diagonal.

The divided differences which occur in Newton's interpolation formula (4.2.5) are those in the downward diagonal of the table. However, since the points $x_1, \ldots, x_n$ can be arbitrarily ordered, we can, for example, also introduce the points in Newton's interpolation formula in *backward* order $x_n, \ldots, x_1$. This gives the **backward form** for the interpolating polynomial

$$p^*(x) = f(x_n) + \sum_{j=1}^{n-1} [x_n, x_{n-1} \ldots, x_{n-j}] f \, (x - x_{n-j+1}) \cdots (x - x_n). \tag{4.2.9}$$

The divided differences in this formula lie on the *upward* diagonal starting at f_n in the table.

Theorem 4.2.3. *The Remainder Term for Interpolation*

Let f be a given real function, with $f^{(n)}(x)$ continuous in $\text{int}(x, x_1, x_2, \ldots, x_n)$. *Denote by p^* the polynomial of degree $n-1$ for which $p(x_i) = f(x_i)$, $i = 1 : n$. Then*

$$f(x) - p^*(x) = [x_1, x_2, \ldots x_n, x] f \, \Phi_n(x) = \frac{f^{(n)}(\xi_x)}{n!} \Phi_n(x), \tag{4.2.10}$$

$\Phi_n(x) = \prod_{i=1}^{n} (x - x_i)$, *for some point* $\xi_x \in \text{int}(x, x_1, x_2, \ldots, x_n)$, *and*

$$[x_1, x_2, \ldots x_n, x_{n+1}] f = \frac{f^{(n)}(\xi)}{n!}, \quad \xi \in \text{int}(x_1, \ldots, x_{n+1}). \tag{4.2.11}$$

Proof. Following a proof due to Cauchy, we introduce a new variable z, and set

$$G(z) = f(z) - p^*(z) - [x_1, x_2, \ldots x_n, x] f \, \Phi_n(z).$$

We know by Theorem 4.2.1 that

$$f(x) - p^*(x) = [x_1, x_2, \ldots x_n, x] f \, \Phi_n(x). \tag{4.2.12}$$

hence $G(x) = 0$. Then $G(z) = 0$ for $z = x, x_1, x_2 \ldots, x_n$. From repeated use of Rolle's theorem it follows that there exists a point $\xi_x \in \text{int}(x, x_1, x_2, \ldots, x_n)$, such that $G^{(n)}(\xi_x) = 0$. Since $p^{*(n)}(z) = 0$ and $\Phi_n^{(n)}(z) = n!$ for all z, $G^{(n)}(z) = f^{(n)}(z) - [x_1, x_2, \ldots x_n, x] f \, n!$. If we now put $z = \xi_x$, we obtain

$$[x_1, x_2, \ldots x_n, x] f = \frac{f^{(n)}(\xi_x)}{n!}. \tag{4.2.13}$$

Put this into the definition of $G(z)$, and set $z = x$. Since $G(x) = 0$, the first statement follows. The second statement follows from (4.2.13) for $x = x_{n+1}$. □

In this theorem x_i, x, $f(x)$, etc. must be *real*, while (4.2.12), i.e. Newton's interpolation formula with the exact remainder term, is valid also in the complex plane. Notice the similarity to the remainder term in Taylor's formula. We shall see that this can be considered as a limiting case when all the points x_i coincide. Notice also that the right hand side of (4.2.10) is zero at the grid points—as it should be.

We are now in a position to give a short proof of the important formula (3.3.6) that we now formulate as a theorem.

Theorem 4.2.4. *Assume that $f \in C^k$. Then*

$$\Delta^k f(x) = h^k f^{(k)}(\zeta), \qquad \zeta \in [x, x + kh]. \tag{4.2.14}$$

If $f \in \mathcal{P}_k$ then $\Delta^k f(x) = 0$. Analogous results hold, mutatis mutandis, for backward and central differences.

Proof. Combine the result in Theorem 4.2.5 with (4.2.11), after appropriate substitutions. □

To form the Newton interpolation polynomial we only need one diagonal of the divided-difference table, and it is not necessary to store the entire table.

Algorithm 4.2.1 *Computing the Newton Coefficients*

The following program replaces (overwrites) the function values $f_1, f_2, \ldots, f_n$, where $f_i = f(x_i)$, $i = 1 : n$. by the downward diagonal of divided differences

$$f_i = [x_1, x_2, \ldots, x_i]f, \quad i = 1 : n$$

of the divided difference table. At step j the jth column of the table is computed. Note that it is necessary to proceed from the bottom of each column to avoid overwriting data needed later! The algorithm uses $n(n-1)/2$ divisions and $n(n-1)$ subtractions.

```
for j = 1 : n − 1
  for i = n : −1 : j + 1
    f_i := (f_i − f_{i−1})/(x_i − x_{i−j});
  end
end
```

The Newton interpolation polynomial is then given by (4.2.9).

If it is not known in advance how many interpolation points that are needed to achieve the required accuracy one interpolation point can be added at a time:

Algorithm 4.2.2 *Divided Difference Table*

The following algorithm computes the difference table one diagonal at a time. In the ith step the entries $f_i, [x_{i-1}, x_i]f, \ldots, [x_1, x_2, \ldots, x_i]f$ on the upward diagonal of the divided-difference table overwrites the function values $f_i, f_{i-1}, \ldots, f_1$.

```
for i = 2 : n
  for j = i : −1 : 2
    f_j := (f_j − f_{j−1})/(x_i − x_{j−1});
  end
end
```

For the evaluation of the Newton polynomial at a point $x = z$, we use the simple Horner-like scheme (4.1.13)

$$p(x) = c_1 + \sum_{j=2}^{n} c_j \Phi_{j-1}(x), \quad \phi_{j-1}(x) = \prod_{i=1}^{j-1}(x - x_i).$$

We have $p(x) = b_1$, where b_1 is computed using the recursion formula:

$$b_n := c_n, \quad b_i := b_{i+1}(z - x_i) + c_i, \quad i = n-1 : -1 : 1. \tag{4.2.15}$$

It is straightforward to show that the computed result is the exact value corresponding to slightly perturbed divided differences; cf. Problem 2.3.6.

The auxiliary quantities $b_n, \ldots, b_2$ are of independent interest, since we have

$$p(x) = b_1 + (x - z)\left(b_2 + \sum_{j=2}^{n-1} b_{j+1}\phi_{j-1}(x)\right). \tag{4.2.16}$$

Repeated applications of the Horner scheme are useful in the evaluation of derivatives of a Newton polynomial.

Example 4.2.1.

Compute the interpolation polynomial for the following table:

$$\begin{array}{lllll} x_1 = 1 & \mathbf{0} & & & \\ & & \mathbf{2} & & \\ x_2 = 2 & 2 & & \mathbf{1} & \\ & & 5 & & \mathbf{0} \\ x_3 = 4 & 12 & & 1 & \\ & & 8 & & \\ x_4 = 5 & 20 & & & \end{array}$$

(The entries appearing in the Newton forward interpolation formula are boldface.) We get two alternative representations

$$\begin{aligned} p(x) &= 0 + 2(x-1) + 1(x-1)(x-2) + 0(x-1)(x-2)(x-4) \\ &= 20 + 8(x-5) + 1(x-5)(x-4) + 0(x-5)(x-4)(x-2) \\ &= x^2 - x, \end{aligned}$$

where the second is obtained from (4.2.9). (Note that for these particular data the unique interpolation polynomial in $\mathcal{P}_4$ actually belongs to the subspace $\mathcal{P}_3$.)

The remainder term in interpolation is according to Theorem 4.2.3 equal to

$$\prod_{i=1}^{n}(x - x_i) f^{(n)}(\xi_x)/n!.$$

Here ξ_x depends on x, but one can say that the error curve behaves for the most part like a polynomial curve $y = c\prod_{i=1}^{n}(x - x_i)$. A similar curve is also typical for error curves arising from least squares approximation. This contrasts sharply with the error curve for Taylor approximation, whose behavior is described approximatively by $y = c(x - x_0)^n$. It is natural to ask what the optimal placing of the interpolation points $x_1, \ldots, x_n$ should be in order to minimize the maximum magnitude of $\Phi_n(x) = \prod_{i=1}^{n}(x - x_i)$ in the interval the formula is to be used. For the interval $[-1, 1]$ the answer is given directly by the minimax property (Lemma 3.2.3) of the Chebyshev polynomials—choose $\Phi_n(x) = T_n(x)/2^{n-1}$. Thus the interpolation points should be taken as the zeros of $T_n(x)$. (In case of an interval $[a, b]$ one makes the linear substitution $x = \frac{1}{2}(a + b) + \frac{1}{2}(b - a)t$.) We have already seen examples of the use of Chebyshev interpolation in the discussion of the Runge phenomenon in Sec. 4.1.4.

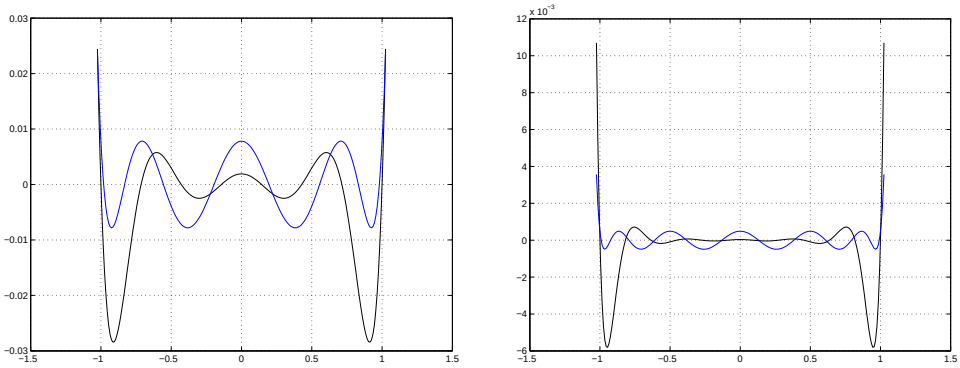


Figure 4.2.1. *Error of interpolation in $\mathcal{P}_n$ for $f(x) = x^n$, using equidistant points and Chebyshev points; $n = 8$ (left) $n = 12$ (right).*

Example 4.2.2.

Use the same notations as before. For $f(x) = x^n$ the interpolation error becomes $f(x) - p^*(x) = \Phi_n(x)$, because $f^{(n)}(x)/n! \equiv 1$. Figure 4.2.1 shows the interpolation error with n equidistant points on $[-1, 1]$ and with n Chebyshev points on the same interval, i.e.

$$x_i = -1 + 2\frac{i-1}{n-1}, \quad x_i = \cos\Big(\frac{2i-1}{n}\frac{\pi}{2}\Big),$$

respectively, for $n = 6$ and $n = 12$. The behaviour of the error curves are rather typical for functions where $f^{(n)}(x)$ is slowly varying. Also note that the error increases rapidly, when x leaves the interval $\text{int}(x_1, x_2, \ldots, x_n)$. In the equidistant case, the error is quite large also in the outer parts of the interval.

Example 4.2.3.

Set $f(x;z) = 1/(z-x)$; x is the variable, z is a parameter; both may be complex. The following elementary, though remarkable, expansion can be proved directly by induction (Problem 3a).

$$\begin{aligned}\frac{1}{z-x} &= \frac{1}{z-x_1} + \frac{x-x_1}{(z-x_1)(z-x_2)} + \ldots + \frac{(x-x_1)(x-x_2)\cdots(x-x_{n-1})}{(z-x_1)(z-x_2)\cdots(z-x_n)} \\ &\quad + \frac{(x-x_1)\cdots(x-x_n)}{(z-x_1)\cdots(z-x_n)(z-x)} \\ &= \sum_{j=1}^{n} \frac{\Phi_{j-1}(x)}{\Phi_j(z)} + \frac{\Phi_n(x)}{\Phi_n(z)(z-x)}. \end{aligned} \tag{4.2.17}$$

When we match this with Newton's interpolation formula we find that

$$[x_1, x_2, \ldots, x_j] f(x;z) = \frac{1}{\Phi_j(z)}, \qquad [x_1, x_2, \ldots, x_j, x] f(x;z) = \frac{1}{\Phi_j(z)(z-x)}. \tag{4.2.18}$$

These formulas can also be proved by induction, by working algebraically with $1/(z-x)$ in the divided difference table (Problem 3). See also Problem 3.2.2a for the equidistant case.

An interesting feature is that these formulas do *not* require that the points x_i are distinct. (They are consistent with the extension to non-distinct points that will be made in Sec. 4.3.) Everything is continuous except if $z = x_i$, $i = 1:n$, or, of course if $z = x$, see Sec. 4.3. If all the x_i coincide, we obtain a geometric series with a remainder.

This is more than a particular example. Since $1/(z-x)$ is the kernel of Cauchy's integral (and several other integral representations), this expansion is easily generalized to arbitrary analytic functions.

For given interpolation points the divided differences in Newton's interpolation formula depends on the ordering in which the points x_i are introduced. Mathematically all orderings give the same unique interpolation polynomial. However, *the condition number for the coefficients c in the Newton polynomial can depend strongly on the ordering of the interpolation points.* For simple everyday interpolation problems the increasing order $x_1 < x_2 < \cdots < x_n$ will give satisfactory results. If the point $\tilde{x}$ where the polynomial is to be evaluated is known, then an ordering such that

$$|\tilde{x} - x_1| \le |\tilde{x} - x_2| \le \cdots \le |\tilde{x} - x_n|$$

can be recommended. (In the equidistant case this corresponds to using Stirling's or Bessel's formula.)

Another suitable choice in case convergence is slow and an interpolation polynomial of high order is used, is the **Leja ordering** defined by

$$x_1 = \max_{1\le i\le n} |x_i|, \qquad \prod_{k=1}^{j-1} |x_j - x_k| = \max_{i\ge j} \prod_{k=0}^{j-1} |x_i - x_k|, \quad j = 2:n-1. \tag{4.2.19}$$

Note also that the barycentric Lagrange interpolation formula, to be introduced in Sec. 4.2.2, has very good stability properties.

Let $\mathcal{K}$ be a compact set in the complex plane with a connected complement. Any sequence of points $\xi_1, \xi_2, \ldots$ which satisfies the conditions

$$|\xi_1| = \max_{\xi\in\mathcal{K}} |\xi|, \qquad \prod_{k=1}^{j-1} |\xi_j - \xi_k| = \max_{\xi\in\mathcal{K}} \prod_{k=0}^{j-1} |\xi - \xi_k|, \quad j = 2, 3, \ldots. \tag{4.2.20}$$

are **Leja points** for $\mathcal{K}$. The points may not be uniquely defined by (4.2.20). For a real interval $[a, b]$ the Leja points are distributed similarly to the Chebyshev points. The main advantage of the Leja points is that it is easy to add new Leja points successively to an already computed sequence of Leja points.

Theorem 4.2.5. *For equidistant points $x_i = x_1 + (i-1)h$, $f_i = f(x_i)$, it holds that*

$$[x_i, x_{i+1}, \ldots, x_{i+k}]f = \frac{\Delta^k f_i}{h^k k!}. \tag{4.2.21}$$

Proof. By induction, with the use of equation (4.2.8). The details are left to the reader. □

We have noted above that, in the notation for the equidistant case, $\nabla^k f_n \approx h^k f^{(k)}$, while in the divided difference notation $f[x_n, x_{n-1}, \ldots, x_{n-k}] \approx f^{(k)}/k!$. For the basis functions of the interpolation formulas, we have, respectively,

$$\binom{x}{k} = O(1), \quad (x - x_n)(x - x_{n-1}) \cdots (x - x_{n-k+1}) = O(h^k),$$

provided that $x - x_{n-j} = O(h)$, $j = 0 : k - 1$.

For many applications the quantities used in the equidistant case have a more appropriate order of magnitude. In some applications to differential equations, there may even be a risk for overflow or underflow, when divided differences are used. F. Krogh [35] introduced a scaling for the divided differences with the same advantage; in the equidistant case these scaled divided differences are identical to $\nabla^k f_n$. The main application so far has been to multistep methods for ordinary differential equations.

4.2.2 Lagrange's Interpolation Formula

A basis that is often advantageous to use is the cardinal basis of $\mathcal{P}_n$ generated by the polynomial

$$\Phi_n(x) = (x - x_1)(x - x_2) \cdots (x - x_n), \tag{4.2.22}$$

where x_i, $i = 1 : n$ are n distinct real numbers. The basis reads,

$$\ell_j(x) = \frac{\Phi_n(x)}{(x - x_j)\Phi_n'(x_j)} = \prod_{\substack{i=1 \\ i\neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}, \quad j = 1 : n \tag{4.2.23}$$

Here ℓ_j, the **Lagrange polynomials** of degree $n-1$, satisfy

$$\ell_j(x_i) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j; \\ 0 & \text{if } i \neq j. \end{cases}$$

Theorem 4.2.6 (Lagrange's interpolation formula).
The unique interpolation polynomial $p \in \mathcal{P}_n$ interpolating the function $f(x)$ at the distinct points x_i, $i = 1 : n$, can be written

$$p(x) = \sum_{j=1}^{n} f(x_j)\ell_j(x), \tag{4.2.24}$$

where

$$\ell_j(x) = \prod_{\substack{i=1 \\ i\neq j}}^{n} \frac{(x - x_i)}{(x_j - x_i)}, \qquad j = 1 : n, \tag{4.2.25}$$

Quite often it is asserted that the Lagrange form is a bad choice for practical computations[10], since for each new value of x the functions $\ell_i(x)$ have to be recomputed at a cost $\mathcal{O}(n^2)$. Further, adding a new data point x_{n+1}, f_{n+1} will require a new computation from scratch. It is concluded that the expression (4.2.24) is not as efficient as the Newton formula.

The above assertions are, however, not well-founded. *The Lagrange representation can easily be rewritten in two more attractive forms, which both are eminently suitable for computation*; see Berrut and Trefethen [3]. Taking out the common factor $\Phi_n(x)$ in (4.2.24) and introducing the **support coefficients**

$$w_j = 1 \Big/ \prod_{\substack{i=1 \\ i\neq j}}^{m} (x_j - x_i), \quad j = 1 : n, \tag{4.2.26}$$

Lagrange interpolation formula can be written in the **modified form**

$$p(x) = \Phi_n(x) \sum_{j=1}^{n} \frac{w_j}{x - x_j} f(x_j), \tag{4.2.27}$$

Here w_j depend only on the given points x_j, $j = 1 : n$, and can be computed in $n(n-1)$ operations. This is twice the work required to compute the divided differences for Newton's interpolation formula. Then, to evaluate $p(x)$ from (4.2.27) for a new value of x we only need to compute $\Phi_n(x)$ and $w_j/(x - x_j)$, $j = 1 : n$, which requires $\mathcal{O}(n)$ operations.

[10] Steffensen [51, p. 25] "For actual numerical interpolation Lagrange's formula is, however, as a rule not very suitable.

The product factor $\Phi_n(x)$ in (4.2.27) can be eliminated as follows. Since the interpolation formula is exact for $f(x) \equiv 1$, we have

$$1 = \Phi_n(x) \sum_{j=1}^{n} \frac{w_j}{x - x_j}.$$

Substituting this in (4.2.27)

$$p(x) = \frac{\displaystyle\sum_{j=1}^{n} \frac{w_j}{x - x_j} f(x_j)}{\displaystyle\sum_{j=1}^{n} \frac{w_j}{x - x_j}}, \quad \text{if} \quad x \neq x_j, \quad j = 1 : n, \tag{4.2.28}$$

which is the **barycentric form** of Lagrange's interpolation formula. This expresses the value $p(x)$ as a weighted mean of the values f_i. (Note that the coefficients $w_j/(x-x_j)$ need not be positive, so the term "barycentric" is not quite appropriate.)

The barycentric formula (4.2.28) has a beautiful symmetric form and is "eminently suitable for machine computation" (Henrici [29, p. 237]) Unlike Newton's interpolation formula, it does not depend on the order in which the nodes are ordered. The numerical stability of the two modified Lagrange interpolation formulas is, contrary to what is often stated, very good. Note that interpolation property of $p(x)$ is preserved even if the coefficients w_i are perturbed, but then $p(x)$ is usually no longer a polynomial but a rational function.

There seems to be a stability problem for the formula (4.2.28) when x is very close to one of the interpolation points x_i. In this case $w_i/(x - x_i)$ will be very large and not accurately computed because of the cancellation in the denominator. However, this is in fact no problem, since there will be exactly the same error in the denominator. Further, in case $\Delta_i = \mathrm{fl}\,(x - x_i)$ is exactly zero, we simply put $\Delta_i = u$ (the unit roundoff).

The Lagrange representation of the interpolation formula can be as efficiently updated. as as Newton's formula. Suppose the support coefficients $w_i^{(k-1)}$, $i = 1 : k-1$ for the points $x_1, \ldots, x_{k-1}$ are known. Adding the point x_k the first $k-1$ new support coefficients can be calculated from

$$w_i^{(k)} = w_i^{(k-1)}/(x_i - x_k), \quad i = 1 : k-1,$$

using $(k-1)$ divisions and subtractions. Finally we have $w_k^{(k)} = 1/\prod_{i=1}^{k-1}(x_k - x_i)$. The computation of the support coefficients is summarized in the following program:

```
w_1 = 1;
for k = 2 : n
   w_k = 1;
   for i = 1 : k - 1
      w_i := w_i/(x_i - x_k);
      w_k = w_k/(x_k - x_i);
   end
end
```

Note that the support coefficients w_i do not depend on the function to be interpolated. Once they are known interpolating a new function f only requires $\mathcal{O}(n)$ operations. This contrasts with Newton's interpolation formula, which requires the calculation of a new table of divided differences for each new function.

Suppose that we use interpolation points in an interval $[a, b]$ of length $2C$. Then as $n \to \infty$ the scale of the weights will grow or decay exponentially at the rate C^{-n}. If n is large or C is far from 1, the result may underflow or overflow even in IEEE double precision. In such cases there may be a need to rescale the support coefficients.

The computation of the support coefficients can be done in only $\frac{1}{2}n(n-1)$ by using the relation (see Problem 5 and [46, Sec. 3.2.1])

$$\sum_{i=1}^{n} w_i = 0, \quad n > 1;$$

to compute $w_n = \sum_{i=1}^{n-1} w_i$. However, using this identity destroys the symmetry and can lead to stability problems for large n. Serious cancellation in the sum will occur whenever $\max_i |w_i|$ is much larger than $|w_n|$. Hence the use of this identity is not recommended in general.

For various important distributions of interpolating points, it is possible to compute the support coefficients w_i analytically.

Example 4.2.4.

For interpolation at the equidistant points x_1, $x_i = x_1 + (i-1)h$, $i = 2 : n$, the support coefficients are

$$\begin{aligned} w_i &= 1 \big/ \left((x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)\right) \\ &= (-1)^{n-i} \big/ \left(h^{n-1}(i-1)!\,(n-i)!\right) = \frac{(-1)^{n-i}}{h^{n-1}(n-1)!} \binom{n-1}{i} \end{aligned}$$

In the barycentric formula (4.2.28) a common factor in the coefficients w_i cancels and we may use instead the modified support coefficients

$$w_i^* = (-1)^{i+1} \binom{n-1}{i}. \tag{4.2.29}$$

For a given n these can be evaluated in only $2n$ operations using the recursion

$$w_1^* = n - 1, \qquad w_i^* = w_{i-1}^* \frac{n-i}{i}, \quad i = 2 : n.$$

Example 4.2.5.

Explicit support coefficients are also known for the Chebyshev points of the first and second kind on $[-1, 1]$. For the Chebyshev points

$$x_i = \cos \frac{(2i-1)}{n} \frac{\pi}{2}, \quad i = 1 : n,$$

they are

$$w_i = (-1)^i \sin \frac{(2i-1)}{n} \frac{\pi}{2}. \tag{4.2.30}$$

For the Chebyshev points of the second kind,

$$x_i = \cos \frac{(i-1)}{(n-1)}\pi, \quad i = 1:n$$

they are

$$w_i = (-1)^i \delta_j, \quad \delta_j = \begin{cases} 1/2 & \text{if } i = 1 \text{ or } i = n, \\ 1, & \text{otherwise} \end{cases}. \tag{4.2.31}$$

Note that all but two weights are equal! This will be considered from another point of view in Sec. 4.6.

For an interval $[a, b]$ the Chebyshev points can be generated by a linear transformation. The corresponding weights w_i then gets multiplied by $2^n(b-a)^n$. However, this factor cancels out in the barycentric formula, and there is no need to include it. Indeed, by *not* doing the risk of overflow or underflow, when $|b-a|$ is far from 1 and n is large, is avoided.

The two examples above show that with equidistant or Chebyshev points only $\mathcal{O}(n)$ operations are needed to get the weights w_i. For these cases the barycentric formula seems superior to all other interpolation formulas.

Lagrange interpolation formula can be used to compute the inverse of the Vandermonde matrix V in (4.1.5) in $\mathcal{O}(n^2)$ operations. If we set $V^{-1} = W = (w_{ij})_{i,j=1}^n$, then $WV = I$, the ith row of which can be written

$$\sum_{j=1}^{n} w_{ij} x_k^j = \delta_{ik}, \quad k = 1:n.$$

This is an interpolation problem that is solved by the Lagrange basis polynomial

$$\ell_i(x) = \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)} = \sum_{j=1}^{n} w_{ij} x^j, \quad j = 1:n. \tag{4.2.32}$$

This shows that V is nonsingular if and only if the points x_i are distinct.

The elements w_{ij} can be computed as follows. First we compute the coefficients of the polynomial

$$\Phi_n(x) = (x - x_1)(x - x_2) \cdots (x - x_n) = \sum_{j=1}^{n+1} a_j x^{j-1}.$$

This can be done by the recursion:

$$a_1 = -x_1; \quad a_2 = 1;$$
$$\textbf{for } k = 2:n$$

```
    a_{k+1} = 1;
    for i = k : -1 : 2
      a_i = a_{i-1} - x_k a_i;
    end
    a_1 = -x_k a_1;
  end
```

Next the coefficients of polynomials

$$q_i(x) = \Phi_n(x)/(x - x_i), \quad i = 1 : n.$$

are computed by synthetic division. Finally, the Lagrange polynomials are obtained from $\ell_i(x) = q_i(x)/q_i(x_i)$, where the scalars $q_i(x_i)$ are computed by Horner's rule. The cost of computing the n^2 elements in W by this algorithm is only $6n^2$ operations.

4.2.3 Iterative Linear Interpolation

There are other recursive algorithms for interpolation. Of interest are those based on *successive linear interpolations*. The basic formula is given in the following theorem.

Theorem 4.2.7.

Assume that the two polynomials $p_{n-1}(x)$ and $q_{n-1}(x)$, both in $\mathcal{P}_{n-1}$ interpolate $f(x)$ at the points $x_1, \ldots, x_{n-1}$, and $x_2, \ldots, x_n$, respectively. If the n points $x_1, x_2, \ldots, x_{n-1}, x_n$ are distinct then

$$p_n(x) = \frac{(x - x_1)q_{n-1}(x) - (x - x_n)p_{n-1}(x)}{x_n - x_1}.$$

is the unique polynomial in $\mathcal{P}_n$ that interpolates $f(x)$ at the m points $x_1, x_2, \ldots, x_{n-1}, x_n$.

Proof. Since $q_{n-1}(x)$ and $p_{n-1}(x)$ both interpolate $f(x)$ at the points $x_2, \ldots, x_{n-1}$ and

$$\frac{(x - x_1) - (x - x_n)}{x_n - x_1} = 1,$$

it follows that also $p_n(x)$ interpolates $f(x)$ at these points. Further, $p_n(x_1) = p_{n-1}(x_1)$ and hence interpolates $f(x)$ at x_1. A similar argument shows that, $p_n(x)$ interpolates $f(x)$ at $x = x_n$. Hence $p_n(x)$ is the unique polynomial interpolating $f(x)$ at the distinct points $x_1, x_2, \ldots, x_n$. □

Neville's and **Aitken's** algorithms both use Theorem 4.2.7 repeatedly to construct successively higher order interpolation polynomials. Let $p_{i,k}$ denote the polynomial interpolating at the k points $x_{i-k+1}, \ldots, x_i$. In Neville's interpolation algorithm one puts

$$p_{i,1} = f(x_i), \quad i = 1 : n$$

and compute for $i = 2 : n$

$$p_{i,k+1} = \frac{(x - x_{i-k})p_{i,k} - (x - x_i)p_{i-1,k}}{x_i - x_{i-k}}, \qquad k = 1 : i - 1, \tag{4.2.33}$$

where $p_{i,k+1}$ interpolates at the points $x_{i-k}, \ldots, x_{i-1}, x_i$. The calculations can be arranged in a table, which for $n = 4$ has the form

$$\begin{array}{lllll} x_1 & f(x_1) = p_{1,1} & & & \\ & & p_{2,2} & & \\ x_2 & f(x_2) = p_{2,1} & & p_{3,3} & \\ & & p_{3,2} & & p_{4,4} \\ x_3 & f(x_3) = p_{3,1} & & p_{4,3} & \\ & & p_{4,2} & & \\ x_4 & f(x_4) = p_{4,1} & & & \end{array}$$

Here any entry is obtained as a linear combination of the nearest two entries in the preceding column. Note that it is easy to add a new interpolation point in this scheme. To proceed only the last lower diagonal needs to be retained. If it is known in advance that a fixed number k of points are to be used, then one can instead generate the table column by column. When one column has been evaluated then the preceding may be discarded.

These formulas are better than Newton's only in the case that $f(x)$ is to be evaluated for the same values of $x - x_i$ for several functions (sequences) f. In this case one should compute

$$t_{ik} = \frac{x - x_{i-k}}{x_i - x_{i-k}}$$

once and for all.

We saw in Sec. 3.3.5 its application to the extrapolation to $x = 0$ of a polynomial given at a few positive arguments, a typical example, where it is efficient and widely used.

Aitken's scheme is similar to Neville's, but uses another sequence of interpolants. Let $p_{i,k}$, $i \geq k$, denote the polynomial interpolating at the k points $x_1, \ldots, x_{k-1}$ and x_i. Set $p_{i,1} = f(x_i)$, as above, and compute for $i = 2 : n$

$$p_{i,k+1} = \frac{(x - x_{i-1})p_{i,k} - (x - x_i)p_{k,k}}{x_k - x_{i-1}}, \qquad k = 1 : i - 1, \tag{4.2.34}$$

The table can again be generated column by column. To be able to add a new point the whole upper diagonal $p_{i,i}$, $i = 1 : k$, must be saved.

4.2.4 Conditioning of the Interpolation Problem

Consider the problem of finding a polynomial $p_f = p_n(x) \in \mathcal{P}_n$ that interpolates given values f_j at distinct points x_j, $j = 1 : n$. With the terminology of Sec. 2.4.3 the input data are f_j, $j = 1 : n$, and the output data is the value of the polynomial p_f evaluated at some fixed point $\tilde{x}$.

Definition 4.2.8.

The condition number of p_f at fixed $\tilde{x}$ and fixed interpolation points but varying data f_j, $j = 1 : n$ is

$$cond\,(\tilde{x}, f) = \lim_{\epsilon \to 0} \sup \left\{ \frac{|p_{f+\Delta f}(\tilde{x}) - p_f(\tilde{x})|}{\epsilon |p_f(\tilde{x})|} : |\Delta f| \le \epsilon |f| \right\} \tag{4.2.35}$$

Lemma 4.2.9. N. J. Higham [32]

Let $\ell_j(x)$ be the Lagrange basis functions. Then the condition number in Definition 4.2.8 *is, for $p_f(\tilde{x}) \neq 0$,*

$$cond\,(\tilde{x}, f) = \frac{\sum_{j=1}^{n} |\ell_j(\tilde{x}) f_j|}{|p_f(\tilde{x})|} \ge 1, \tag{4.2.36}$$

and for any Δf with $|\Delta f| \le \epsilon |\Delta f|$ we have

$$\frac{|p_{f+\Delta f}(\tilde{x}) - p_f(\tilde{x})|}{\epsilon |p_f(\tilde{x})|} \le cond\,(\tilde{x}, f)\epsilon.$$

Proof. Using the Lagrange basis,

$$p_{f+\Delta f}(\tilde{x}) - p_f(\tilde{x}) = \sum_{j=1}^{n} \ell_j(\tilde{x}) \Delta f_j,$$

It follows immediately that the expression in (4.2.36) is an upper bound for the condition number and it is clearly at least 1. Equality is attained for $\Delta f_j = \epsilon\, sign\,(\ell_j(\tilde{x}))|f_j|$. The inequality follows trivially. □

Assume that the interpolation points x_j, $j = 1 : n$ lie in $[-1, 1]$. Consider $cond\,(x, 1)$, the condition number of interpolating the function $f(x) = 1$ at these points. By Lemma 4.2.9 we have

$$cond\,(x, 1) = \sum_{j=1}^{n} |\ell_j(x)|.$$

This quantity is related to the so called **Lebesgue constant** defined by

$$\Lambda_n = \sup_{x \in [-1,1]} \sum_{j=1}^{n} |\ell_j(x)| \ge cond\,(x, 1). \tag{4.2.37}$$

For equally spaced point, however, Λ_n grows at a rate proportional to $2^n/(n \log n)$; see Cheney and Light [12, Chap. 3].

Theorem 4.2.10 (N. Higham [32]).

Assume that x_i, f_i and x are floating point numbers. Then the computed value $\bar{p}(x)$ of the interpolation polynomial using the modified Lagrange formula (4.2.27) *satisfies*

$$\bar{p}(x) = \Phi_n(x) \sum_{i=1}^{n} \frac{w_i}{x - x_i} f(x_i) \prod_{j=1}^{5(n+1)} (1 + \delta_{ij})^{\pm 1}, \tag{4.2.38}$$

where $|\delta_{ij}| \leq u$.

Thus the formula (4.2.27) *computes the exact value of an interpolating polynomial corresponding to slightly perturbed function values $f(x_i)$. Hence this formula is* backward stable *in the sense of Definition* 2.4.9.

From this theorem and Lemma 4.2.9 the forward error bound

$$\frac{|p_n(\tilde{x}) - \bar{p}_n(\tilde{x})|}{|p_n(\tilde{x})|} \leq \gamma_{5n+5} cond\,(\tilde{x}, f). \tag{4.2.39}$$

can be obtained.

The barycentric formula is not backward stable. A forward error bound similar to (4.2.39) but containing an extra term proportional to $cond\,(\tilde{x}, 1) = \sum_{j=1}^{n} |\ell_j(x)|$ can be shown. Hence the barycentric formula can be significantly less accurate than the modified Lagrange formula (4.2.27) only for a poor choice of interpolation points.

4.2.5 Interpolation by Rational Functions

Rational approximation is often superior to polynomial approximation in the neighborhood of a point at which the function has a singularity. The rational interpolation problem is to determine a rational function

$$f_{m,n}(z) = \frac{P_m(z)}{Q_n(z)} \equiv \frac{\sum_{j=0}^{m} p_j z^j}{\sum_{j=0}^{n} q_j z^j}, \tag{4.2.40}$$

so that

$$f_{m,n}(x_i) = f_i, \quad i = 0 : m + n. \tag{4.2.41}$$

A *necessary condition* for (4.2.41) to hold clearly is that

$$P_m(x_i) - f_i Q_n(x_i) = 0, \quad i = 0 : m + n. \tag{4.2.42}$$

or for $i = 0 : m + n$,

$$p_0 x_i + p_1 x_i + \cdots + p_m x_i^m - f_i(q_0 x_i + q_1 x_i + \cdots + q_n x_i^n) = 0, \tag{4.2.43}$$

This is a homogeneous linear system of $(m + n + 1)$ equations for the $(m + n + 2)$ coefficients in $P_{m,n}$ and $Q_{m,n}$. Such a system always has a nontrivial solution. The coefficients are determined only up to a common factor $\rho \neq 0$.

In contrast to polynomial interpolation a solution to the rational interpolation problem may not exist as shown in the following example:

Example 4.2.6.

Assume that we want to interpolate the first four of the points

x	0	1	2	3	4
y	2	3/2	4/5	1/2	6/17

by a rational function

$$f_{2,1} = \frac{p_0 + p_1 x + p_2 x^2}{q_0 + q_1 x}.$$

Then we must solve the homogeneous linear system

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix} - \begin{pmatrix} 2 & 0 \\ 3/2 & 3/2 \\ 4/5 & 8/5 \\ 1/2 & 3/2 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \end{pmatrix} = 0.$$

Setting $p_2 = 1$ we find the solution $p_0 = 8$, $p_1 = -6$, $q_0 = 4$, $q_1 = -2$. The corresponding rational function

$$f_{2,1} = \frac{8 - 6x + x^2}{4 - 2x} = \frac{(4-x)(2-x)}{2(2-x)}$$

has the common factor $(2 - x)$ and is *reducible* to $f_{2,1} = (4 - x)/2$. The original form is indeterminate $0/0$ at $x = 2$ while the reduced form does not take on the prescribed value at $x = 2$.

An algorithm similar to Newton's algorithm can be used for finding rational interpolants in continued fraction form. Set $v_0(x) = f(x)$, and use a sequence of substitutions

$$v_k(x) = v_k(x_k) + \frac{x - x_k}{v_{k+1}(x)}, \quad k = 0, 1, 2, \ldots. \tag{4.2.44}$$

The first two substitutions give

$$f(x) = v_0(x) = v_0(x_1) + \frac{x - x_0}{v_1(x)} = v_0(x_0) + \cfrac{x - x_0}{v_1(x_1) + \cfrac{x - x_1}{v_2(x)}}.$$

In general this gives a continued fraction

$$f(x) = a_0 + \cfrac{x - x_0}{a_1 + \cfrac{x - x_1}{a_2 + \cfrac{x - x_2}{a_3 +}}} \ldots = a_0 + \frac{x - x_1}{a_1 +} \, \frac{x - x_2}{a_2 +} \, \frac{x - x_3}{a_3 +} \ldots, \tag{4.2.45}$$

where $a_k = v_k(x_k)$, and we have used the compact notation introduced in Sec. 3.5.1. This becomes an identity if the expansion is terminated by replacing a_n in the last denominator by $a_n + (x - x_n)/v_{n+1}(x)$. If we set $x = x_k$, $k \le n$, then the fraction terminates before the the residual $(x - x_n)/v_{n+1}(x)$ is introduced. This means that

setting $1/v_{k+1} = 0$ will give a rational function which agrees with $f(x)$ at the points x_i, $i = 0 : k \le n$, assuming that the constants $a_0, \ldots, a_k$ exist. These continued fractions give a sequence of rational approximations $f_{k,k}$, $f_{k+1,k}$, $k = 0, 1, 2, \ldots$.

Introducing the notation

$$v_k(x) = [x_0, x_1, \ldots, x_{k-1}, x]\phi \tag{4.2.46}$$

we have $a_k = [x_0, x_1, \ldots, x_{k-1}, x_k]\phi$. Then by (4.2.44) we have

$$[x]\phi = f(x), \qquad [x_0, x]\phi = \frac{x - x_0}{[x]\phi - [x_0]\phi} = \frac{x - x_0}{f(x) - f(x_0)},$$
$$[x_0, x_1, x]\phi = \frac{x - x_1}{[x_0, x]\phi - [x_0, x_1]\phi},$$

and in general

$$[x_0, x_1, \ldots, x_{k-1}, x]\phi = \frac{x - x_{k-1}}{[x_0, \ldots, x_{k-2}, x]\phi - [x_0, \ldots, x_{k-2}, x_{k-1}]\phi}. \tag{4.2.47}$$

Therefore we also have

$$a_k = \frac{x - x_{k-1}}{[x_0, \ldots, x_{k-2}, x_k]\phi - [x_0, \ldots, x_{k-2}, x_{k-1}]\phi}. \tag{4.2.48}$$

We call the quantity defined by (4.2.48) the kth **inverse divided difference** of $f(x)$. Note that certain inverse differences can become infinite if the denominator vanishes. They are, in general, *symmetrical only in their last two arguments*[11]

The inverse divided differences of a function $f(x)$ can conveniently be computed recursively and arrange in a table similar to the divided difference table.

$$\begin{array}{llllll}
x_1 & f(x_1) & [x_1]\phi & & & \\
 & & & [x_1, x_2]\phi & & \\
x_2 & f(x_2) & [x_2]\phi & & [x_1, x_2, x_3]\phi & \\
 & & & [x_2, x_3]\phi & & [x_1, x_2, x_3, x_4]\phi \\
x_3 & f(x_3) & [x_3]\phi & & [x_2, x_3, x_4]\phi & \\
 & & & [x_3, x_4]\phi & & \\
x_4 & f(x_4) & [x_4]\phi & & &
\end{array}$$

Here the upper diagonal elements are the desired coefficients in the expansion (4.2.45).

Example 4.2.7.

Assume that we want to interpolate the points given in Example 4.2.7. Form-

[11] The **reciprocal differences** of Thiele [55] are symmetric functions of all their arguments; see Hildebrand [33, pp. 406ff].

ing the inverse differences we get the table

x_i	f_i	ϕ_1	ϕ_2	ϕ_3	ϕ_4
0	2				
		-2			
1	3/2		3		
		5/3		0	
2	4/5		∞		-5
		-2		$-1/5$	
3	1/2		-7		
		$-17/7$			
4	6/17				

This gives a sequence of rational approximations. If we terminate the expansion

$$f_{2,2} = 2 + \frac{x}{-2+}\,\frac{x-1}{3+}\,\frac{x-2}{0+}\,\frac{x-3}{-5}.$$

after a_3 we recover the solution of the previous example. Note that the the degeneracy of the approximation is shown by the entry $a_3 = 0$. Adding the last fraction gives the (degenerate) approximation

$$f_{2,2} = \frac{2+x}{1+x^2}.$$

It is verified directly that this rational function interpolates all the given points.

The formulas using inverse or reciprocal differences are are useful if one wants to determine the coefficients of the rational approximation, and use it for to compute approximations for several arguments. If one is only wants the value of the rational interpolating function for a single argument, then it is more convenient to use an alternative algorithm of Neville-type. If we consider the sequence of rational approximations of degrees (m, n)

$$(0,0),\quad (0,1),\quad (1,1),\quad (1,2),\quad (2,2),$$

the following recursive algorithm results (Stoer and Bulirsch [52, Sec. 2.2]):
For $i = 0, 1, 2, \ldots$, set $T_{i,-1} = 0$, $T_{i,0} = f_i$, and

$$T_{ik} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\dfrac{x - x_{i-k}}{x - x_i}\left[1 - \dfrac{T_{i,k-1} - T_{i-1,k-1}}{T_{i,k-1} - T_{i-1,k-2}}\right] - 1}, \quad 1 \le k \le i. \tag{4.2.49}$$

As in Neville interpolation the calculations can be arranged in a table of the form

$$
\begin{array}{cccccc}
(m,n) = & (0,0) & (0,1) & (1,1) & (1,2) & \cdots \\
\hline
 & f_1 = T_{1,0} & & & & \\
0 & & T_{2,1} & & & \\
 & f_2 = T_{2,0} & & T_{3,2} & & \\
0 & & T_{3,1} & \longrightarrow & T_{4,3} & \\
 & f_3 = T_{3,0} & & T_{4,2} & \vdots & \ddots \\
0 & & T_{4,1} & \vdots & & \\
 & f_4 = T_{4,0} & \vdots & & & \\
\vdots & \vdots & & & &
\end{array}
$$

Here any entry is determined by a rhombus rule from three entries in the preceding two columns. Note that it is easy to add a new interpolation point in this scheme.

Review Questions

1. Prove the theorem which says that the interpolation problem for polynomials has a unique solution.
2. When is linear interpolation sufficient?
3. Derive Newton's interpolation formula.
4. Derive Newton's interpolation formula for the equidistant case, starting from Newton's general interpolation formula. How is this formula easily remembered?
5. Discuss how various sources of error influence the choice of step length in numerical differentiation.
6. Derive the Lagrange interpolation formula. Show how it can be rewritten in barycentric form. When is the latter form more efficient to use?

Problems and Computer Exercises

1. (a) Compute $f(3)$ by quadratic interpolation in the following table:

x	1	2	4	5
$f(x)$	0	2	12	21

   Use the points $1, 2$, and 4, and the points $2, 4$, and 5, and compare the results.
   (b) Compute $f(3)$ by cubic interpolation.

2. Compute $f(0)$ using one of the interpolation formulas treated above on the following table:

x	0.1	0.2	0.4	0.8
$f(x)$	0.64987	0.62055	0.56074	0.43609

The interpolation formula is here used for *extrapolation*. Use also Richardson extrapolation and compare the results.

3. *Error in linear interpolation*

(a) Suppose we want to compute by linear interpolation the value $y(x)$ at a point $x = x_0 + \theta h$, $h = x_1 - x_0$. Using (4.2.10) show that for $0 \le \theta \le 1$ the remainder $R(x) = f(x) - p(x)$ satisfies

$$|R(x)| \le \frac{h^2}{8} M_2, \tag{4.2.50}$$

(b) Show that if the values f_0 and f_1 are given to t correct decimal digits then the round-off error R_T in linear interpolation $p(x) = (1-\theta)f_0 + \theta f_1$, for $0 \le \theta \le 1$ satisfies $|R_T| =\le \frac{1}{2}10^{-t}$. Show further that if $h^2 M_2 \le 4 \cdot 10^{-t}$, then the total error in $p(x)$ is bounded by 10^{-t} twice the round-off error in the given values of f.

(c) Motivate the rule of thumb that linear interpolation suffices if $|\Delta^2 f_n|/8$ is a tolerable truncation error.

4. Work out the details of Example 4.2.3 (about divided differences etc. for $1/(z-x)$).

5. (a) Consider the two polynomials $p(x)$ and $q(x)$, both in $\mathcal{P}_n$, which interpolate $f(x)$ at the points $x_1, \ldots, x_n$, and $x_2, \ldots, x_{n+1}$, respectively. Assume that $\{x_i\}_{i=1}^{n+1}$ is an increasing sequence, and that $f^{(n)}(x)$ has constant sign in the interval $[x_1, x_{n+1}]$. Show that $f(x)$ is contained between $p(x)$ and $q(x)$ for all $x \in [x_1, x_{n+1}]$.

(b) Suppose that $f(x) = f_1(x) - f_2(x)$, where both $f_1^{(n)}(x)$ and $f_2^{(n)}(x)$ have the same constant sign in $[x_1, x_{n+1}]$. Formulate and prove a kind of generalization of the result in (a).

6. Using the barycentric formula (4.2.27) the interpolation polynomial can be written

$$p(x) = \sum_{i=1}^{n} w_i f(x_i) \prod_{\substack{j=1 \\ j \ne i}}^{m} (x - x_j).$$

Show by taking $f(x) \equiv 1$ and equating the coefficients for x^{n-1} on both sides that the support coefficients satisfy $\sum_{i=1}^{n} w_i = 0$.

7. Show that, if the points x_i are distinct,

$$[x_1, x_2, \ldots, x_m] f = \sum_{i=0}^{m} \frac{f(x_i)}{\Phi_m'(x_i)},$$

where $\Phi_m(x)$ is defined in (4.2.26).

Hint: Compare the coefficients of x^{n-1} in Newton's and Lagrange's expressions for the interpolation polynomial.

8. (a) Check Table 4.2.1 and the conclusions about the optimal step length in the text, Investigate how the attainable accuracy varies with u.

(b) Study the analogous question for $f''(x_0)$ using the formula

$$f''(x_0) \approx \left(1 - \frac{\delta^2}{12} + \frac{\delta^4}{90} - \frac{\delta^6}{560} + \frac{\delta^8}{3,150} - \dots\right) \frac{\delta^2 f_0}{h^2}.$$

9. Prove the validity of Algorithm 4.2.3

10. Given a backwards (upwards) diagonal in the table of divided differences (scaled or unscaled), $\langle X; m, i\rangle Y$, $i = 1 : k$. Find a recurrence formula for the computation of the next diagonal of the difference scheme for the interpolation polynomial $P(x; m, k)Y$, i.e. if $\langle X; m+1, k\rangle Y = \langle X; m, k\rangle Y$ (why?), find $\langle X; m+1, i\rangle Y$, $i = k-1, k-2, \dots, 0$.

Hint: Look up the equidistant case in Example 3.2.6.

11. (Bulirsch and Rutishauser (1968))

(a) The function $\cot x$ has a singularity at $x = 0$. Use values of $cotx$ for $x = 1°$, $2°$, …, $5°$. and rational interpolation of order (2,2) to determine an approximate value of $\cot x$ for $x = 2.5°$, and its error.

(b) Use polynomial interpolation for the same problem. Compare the result with that in (a).

4.3 Generalizations and Applications

4.3.1 Interpolation using Values of Derivatives

The general **Hermite interpolation** problem is the following: Given n distinct points $\{x_i\}_{i=1}^n$, and numbers $r_i \geq 1$, Find a polynomial $p(x)$ of degree $m-1$, where $\sum_{i=1}^n r_i = m$, so that $p(x)$ and its first $r_i - 1$ derivatives agree with those of $f(x)$ at x_i, i.e.

$$p(x)^{(j)}(x_i) = f(x)^{(j)}(x_i), \quad j = 0 : r_i - 1, \quad \sum_{i=1}^{n} r_i = m, \tag{4.3.1}$$

$i = 1 : n$. (We use here the notation $f^{(0)}(x)$ for $f(x)$.)

Hermite interpolation can be viewed as the result of passages to the limit in interpolation at m points, where for $i = 1 : n$ r_i interpolation points coalesce into the point x_i. We say that the point x_i has **multiplicity** r_i. For example, the Taylor polynomial in $\mathcal{P}_m$

$$p(x) = \sum_{j=0}^{m-1} \frac{f^{(j)}(x_1)}{j!}(x - x_0)^j \tag{4.3.2}$$

interpolates $f(x)$ at the point x_1 with multiplicity m (or x_1 is repeated m times).

Note that (4.3.1) are precisely m conditions on $p(x)$, so we can expect that the Hermite interpolation problem is uniquely solvable.

Theorem 4.3.1.
The problem of finding a polynomial $p \in \mathcal{P}_m$ that satisfies the Hermite interpolation conditions

$$p^{(j)}(x_i) = f^{(j)}(x_i), \quad i = 1:n, \quad j = 0:r_i - 1, \tag{4.3.3}$$

where $r_i \geq 1$, $\sum r_i = m$, has a unique solution.

Proof. The conditions are expressed by a system of m linear equations for the coefficients of $\hat{p}$, with respect to some basis. This has a unique solution for any right hand side, unless the corresponding homogeneous problem has a non-trivial solution. Suppose that a polynomial $p \in \mathcal{P}_m$ comes from such a solution of the homogeneous problem, i.e.

$$p^{(j)}(x_i) = 0, \quad i = 1:n, \quad j = 0:r_i - 1.$$

Then, x_i must be a zero of multiplicity r_i of $p(x)$, hence $p(x)$ must have at least $\sum r_i = m$ zeros (counting the multiplicities). But this is impossible, because the degree of p is less than m. This contradiction proves the theorem. □

Since Hermite interpolation is a boundary case of ordinary interpolation the remainder term for interpolation given in Theorem 4.2.3 applies. Hence, assuming that f is a real function, with continuous derivatives of order at least m, the error in Hermitian interpolation is given by

$$f(x) - p(x) = \frac{f^{(m)}(\xi_x)}{m!}\Phi_n(x), \quad \Phi_n(x) = \prod_{i=1}^{n}(x - x_i)^{r_i}. \tag{4.3.4}$$

for some point $\xi_x \in \text{int}(x, x_1, x_2, \ldots, x_n)$.

Example 4.3.1.
Consider the problem of finding a polynomial $p(x) \in \mathcal{P}_4$ that interpolates the function f and its first derivative f' at the two points x_0 and x_1, and also its second derivative at x_0. In the notations of Sec. 4.1.1 the linear system for the coefficient vector c becomes $V^T c = \tilde{f}$, where $\tilde{f} = (f(x_1), f'(x_1), f''(x_1), f(x_2), f'(x_2))^T$, and

$$V = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ x_1 & 1 & 0 & x_2 & 1 \\ x_1^2 & 2x_1 & 2 & x_2^2 & 2x_2 \\ x_1^3 & 3x_1^2 & 6x_1 & x_2^3 & 3x_2^2 \\ x_1^4 & 4x_1^3 & 12x_1^2 & x_2^4 & 4x_2^3 \end{pmatrix} \tag{4.3.5}$$

is a **confluent Vandermonde matrix**. Note that the second, third, and fifth column of V is obtained by "differentiating" the previous column. From Theorem 4.3.1 we conclude that such confluent Vandermonde matrices are nonsingular.

There are explicit formulas, analogous to Lagrange's formula, for Hermite interpolation; see [52, Sec. 2.1.5]. The Hermite interpolation polynomial is written in the form

$$p(x) = \sum_{i=1}^{n} \sum_{k=0}^{r_i-1} f^{(k)}(x_i) L_{ik}(x), \tag{4.3.6}$$

where $L_{ik}(x)$ are **generalized Lagrange polynomials**. These can be defined starting from the auxiliary polynomials

$$l_{ik}(x) = \frac{(x-x_i)^k}{k!} \prod_{\substack{j=1 \\ j\neq i}}^{n} \left(\frac{x-x_j}{x_i-x_j} \right)^{r_j}, \quad i = 1:n, \quad k = 0: r_j - 1.$$

Next, put

$$L_{i,r_i-1} = l_{i,r_i-1}, \quad i = 1:n,$$

and form recursively,

$$L_{ik}(x) = l_{ik}(x) - \sum_{\nu=k+1}^{r_i-1} l_{ik}^{(\nu)}(x_i) L_{i,\nu}(x), \quad k = r_i - 2 : -1 : 0.$$

It can be showed by induction that

$$L_{ik}^{(\sigma)}(x_i) = \begin{cases} 1, & \text{if } i = j \text{ and } k = \sigma; \\ 0, & \text{otherwise.} \end{cases}$$

Hence the L_{ik} are indeed the appropriate polynomials.

Example 4.3.2.

An important special case is when $r_i = 2$, $i = 1 : n$. Then the Hermite interpolating polynomial is the **osculating polynomial**, which agrees with $f(x)$ and $f'(x)$ at $x = x_i$, $i = 1 : n$. In this case we can write

$$p(x) = \sum_{i=1}^{n} (f(x_i) L_{i0}(x) + f'(x_i) L_{i1}(x)).$$

Here $L_{ik}(x)$ can be written in the form

$$L_{i1}(x) = (x - x_i) l_i(x)^2, \qquad L_{i0}(x) = (1 - 2l_i'(x_i)(x - x_i)) l_i(x)^2,$$

where $l_i(x)$, $i = 1 : n$, are the elementary Lagrange polynomials.

An interpolation problem that contains points of multiplicity greater than one can be obtained from the case with distinct points by a passage to the limit. Newton's interpolation formula is suitable for handling this case. Since by Theorem 4.2.2 a divided difference is a symmetric function of its arguments these can be permuted before taking the limit. We can therefore assume, without loss of generality, that

equal arguments are placed together, and that the values x_i for different groups of arguments are different.

Assume that $f^{(r)}(x)$ is continuous. Then by Theorem 4.2.3,

$$[x_1, x_2, \dots x_{r+1}]f = f^{(r)}(\xi)/r!, \quad \xi \in \text{int}(x_1, x_2, \dots, x_{r+1}).$$

If we let $x_i \to x$, $i = 1 : r+1$, then $[x_1, x_2, \dots x_{r+1}]f \to f^{(r)}(x)/r!$. Hence the natural definition of a divided difference with r equal arguments reads

$$[x, x, \dots, x]f = f^{(r)}(x)/r!, \quad r+1 \text{ equal arguments.} \tag{4.3.7}$$

One may also establish the following more general formula

$$[x_1, \dots, x_k, x \dots, x]f = \frac{1}{r!}\frac{d^r}{dx^r}[x_1, \dots, x_k, x]f \quad r+1 \text{ equal arguments.} \tag{4.3.8}$$

We now give a representation of the divided differences which allows several multiplicities.

Theorem 4.3.2.

Assume that $f^{(n-1)}(x)$ is continuous in $[a, b]$, $x_1, \dots, x_n \in [a, b]$ and x is distinct from any x_i. Then

$$[x, x_1, x_2, \dots, x_n]f = \frac{[x, x_2, \dots, x_n]f - [x_1, x_2, \dots, x_n]f}{x - x_1}, \tag{4.3.9}$$

gives the unique continuous extension of divided differences no matter what multiplicities occur in $x_1, \dots, x_n$.

Proof.

□

This definition and the usual recurrence formula for the divided differences are, under the above assumptions, sufficient for the construction of a table of divided differences in the case of multiple points, e.g.,

$$[x_0, x_0]f = \lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_0),$$
$$[x_0, x_0, x_1]f = \frac{[x_0, x_0]f - [x_0, x_1]f}{x_0 - x_1} = \frac{f'(x_0) - [x_0, x_1]f}{x_0 - x_1}.$$

It can be shown that if $f \in C^k$, the divided differences belong to $C^{k+1-\max r_i}$, and that the interpolation polynomial has this kind of differentiability with respect to the x_i, nota bene if the "groups" do not coalesce further.

Example 4.3.3. Consider the interpolation problem in Example 4.3.1. For this we construct the generalized divided-difference table, where $x_1 \neq x_0$.

$$\begin{array}{llllll}
x_0 & f_0 & & & & \\
 & & f_0' & & & \\
x_0 & f_0 & & \frac{1}{2}f_0'' & & \\
 & & f_0' & & [x_0,x_0,x_0,x_1]f & \\
x_0 & f_0 & & [x_0,x_0,x_1]f & & [x_0,x_0,x_0,x_1,x_1]f \\
 & & [x_0,x_1]f & & [x_0,x_0,x_1,x_1]f & \\
x_1 & f_1 & & [x_0,x_1,x_1]f & & \\
 & & f_1' & & & \\
x_1 & f_1 & & & &
\end{array}$$

The interpolating polynomial now reads

$$\begin{aligned}
p(x) &= f_0 + (x-x_0)f_0' + (x-x_0)^2\frac{1}{2}f_0'' + (x-x_0)^3[x_0,x_0,x_0,x_1]f \\
&\quad + (x-x_0)^3(x-x_1)[x_0,x_0,x_0,x_1,x_1]f.
\end{aligned}$$

$$\begin{aligned}
f(x) - p(x) &= [x_0,x_0,x_0,x_1,x_1,x]f(x-x_0)^3(x-x_1)^2 \\
&= f^{(5)}(\xi_x)(x-x_0)^3(x-x_1)^2/5!
\end{aligned}$$

For the simplest Hermite interpolation problem, i.e. **cubic interpolation**, the given data are $f_i = f(x_i)$, $f_i' = f'(x_i)$, $i = 0, 1$. We can write the interpolation polynomial

$$\begin{aligned}
p(x) &= f_0 + (x-x_0)[x_0,x_1]f + (x-x_0)(x-x_1)[x_0,x_0,x_1]f \\
&\quad + (x-x_0)^2(x-x_1)[x_0,x_0,x_1,x_1]f.
\end{aligned}$$

Set $x_1 = x_0 + h$ and $x = x_0 + \theta h$, and denote the remainder $f(x) - p(x)$ by R_T. Then one can show (Problem 1) that

$$\begin{aligned}
p(x) &= f_0 + \theta\Delta f_0 + \theta(1-\theta)(hf_0' - \Delta f_0) \\
&\quad - \theta^2(1-\theta)\Big[(hf_0' - \Delta f_0) + (hf_1' - \Delta f_0)\Big] \\
&= (1-\theta)f_0 + \theta f_1 + \theta(1-\theta)\Big[(1-\theta)(hf_0' - \Delta f_0) - \theta(hf_1' - \Delta f_0)\Big] \qquad (4.3.10)
\end{aligned}$$

For $x \in [x_0, x_1]$ we get the error bound

$$|R_T| \leq \frac{h^4}{384}\max_{x\in[x_0,x_1]}|f^{(4)}(x)|. \qquad (4.3.11)$$

In particular, putting $t = 1/2$, we get the useful formula

$$f_{1/2} = \frac{1}{2}(f_0 + f_1) + \frac{1}{8}h(f_0' - f_1') + R_T. \qquad (4.3.12)$$

Sometimes there are gaps in the sequence of derivatives that are numerically known at a point. The problem is then called **Birkhoff interpolation** or **lacunary** interpolation. We illustrate by two examples that such problems can either have a unique solution or lead to a singular system of linear equations. See also Problems. We use the notation of Sec. 4.1.1.

Example 4.3.4. Given $\tilde{f} = \big(f(-1),\, f'(0),\, f(1)\big)^T$. Try to find a polynomial $p \in \mathcal{P}_3$ that satisfies such data. The new feature is that $f(0)$ is missing.

Set up (4.1.4), i.e. $M_{\mathbf{p}}c = \tilde{f}$, in the power basis.

$$M_{\mathbf{p}} = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}.$$

The determinant is evidently zero, so *there is no solution* for most data. An explanation is that $hf' = \mu\delta f$ for all $f \in \mathcal{P}_3$.

Example 4.3.5. Given $\tilde{f} = \big(f(1),\, f(-1),\, f''(1),\, f''(-1)\big)^T$. Try to find a polynomial $p^* \in \mathcal{P}_4$ that satisfies such data. The new feature is that there are no first derivatives. In this case, we obtain for the power basis,

$$M_{\mathbf{p}} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 0 & 0 & 2 & 6 \\ 0 & 0 & 2 & -6 \end{pmatrix}.$$

The determinant is 48, and *this interpolation problem is uniquely solvable.* The coefficient vector of p is $c = M_{\mathbf{p}}^{-1}\tilde{f}$.

The first coordinate of c, i.e. $e_1^T c$, is an approximation to $f(0)$; this is also a linear functional of f. Denote by Rf the remainder functional for this approximation, i.e.

$$Rf = f(0) - e_1^T M_{\mathbf{p}}^{-1}\tilde{f}. \tag{4.3.13}$$

4.3.2 Inverse interpolation

It often happens that one has a sequence of pairs $\{(x_i, y_i)\}$ and want to determine a point where $y(x) = c$. We saw an example as early as in the simulation of the motion of a ball (Sec. 1.4), when we computed the landing point. We there used linear interpolation.

In general a natural approach is to reverse the roles of x and y, i.e. to compute the inverse function $x(y)$ for $y = c$, by means of Newton's interpolation formula with the divided differences $[y_i, y_{i+1}, \ldots y_{i+j}]x$ (unscaled or scaled). This is called **inverse interpolation** It is convenient to order the points so that $\ldots < y_5 < y_3 < y_1 < c < y_2 < y_4 < \ldots$. This approach is successful if the function $x(y)$ is suitable for local approximation by a polynomial.

Sometimes, however, the function $y(x)$ is much better suited for local approximation by a polynomial than the inverse function $x(y)$. Then we can instead, for some m, solve the following equation,

$$y_1 + [x_1, x_2]y \cdot (x - x_1) + \sum_{j=2}^{n-1} [x_1, x_2, \ldots x_{j+1}]y\, \Phi_j(x) = c.$$

Again it is convenient to order the points so that the root α comes in the middle, e.g., so that $\ldots < x_5 < x_3 < x_1 < \alpha < x_2 < x_4 < \ldots$.

We write the equation in the form $x = x_1 + F(x)$, where

$$F(x) \equiv \frac{(c - y_1) - \sum_{j=2}^{n-1} [x_1, x_2, \ldots x_{j+1}]y\, \Phi_j(x)}{[x_1, x_2]y}.$$

Then we can use *iteration*. We ignore the sum to get the first guess x^0; this means the same as linear inverse interpolation. We then iterate, $x^i = x_1 + F(x^{i-1})$, until x^i and x^{i-1} are close enough. A more careful termination criterion will be suggested in Chapter 6, where the effect on the result of errors like the interpolation error is also discussed.

Suppose that $x_i - x_1 = O(h)$, $i > 1$, where h is some small parameter in the context (usually some step size), then $\Phi_j(x) = O(h^j)$, $\Phi_j'(x) = O(h^{j-1})$. The divided differences are $O(1)$, and we assume that $[x_1, x_2]y$ is bounded away from zero. Then the terms of the sum decrease like h^j.

By the discussion of iteration in Sec. 1.2, the convergence ratio is $F'(x)$, and this is here approximately

$$\frac{\Phi_2'(x)[x_1, x_2, x_3]y}{[x_1, x_2]y} = O(h).$$

So, if h is small enough, the iterations converge rapidly. If more than two iterations are needed, Aitken acceleration (Sec. 3.3.2) may be practical.

4.3.3 Numerical differentiation

An important problem in many applications is to approximate the derivative of a function using only given function values. A straightforward solution to this problem is to use the derivative of the corresponding interpolation polynomial as the approximation to the derivative of the function. This can also be done for higher order derivatives.

We shall first study the computation of $f'(x_0)$. By the operator expansion (3.3.50) derived in Sec. 3.3.4 we have

$$f'(x_0) = \Big(1 - \frac{\delta^2}{6} + \frac{\delta^4}{30} - \frac{\delta^6}{140} + \frac{\delta^8}{630} - \ldots\Big)\frac{1}{h}\mu\delta f_0, \quad \mu\delta f_0 = \frac{f_1 - f_{-1}}{2}. \tag{4.3.14}$$

By squaring this we obtain

$$f''(x_0) \approx \left(1 - \frac{\delta^2}{12} + \frac{\delta^4}{90} - \frac{\delta^6}{560} + \frac{\delta^8}{3,150} - \frac{\delta^{10}}{16,632} \pm \ldots\right)\frac{\delta^2 f_0}{h^2}. \tag{4.3.15}$$

Suppose that the function values have errors whose magnitude does not exceed $\frac{1}{2}U$. Then the error bound on $\mu\delta f_0 = \frac{1}{2}(f_1 - f_{-1})$ is also equal to $\frac{1}{2}U$. Similarly one can show that the error bounds in $\mu\delta^{(2k+1)} f_0$, for $k = 1:3$ are $1.5U, 5U, 17.5U$, respectively. Thus one gets the upper bounds $U/(2h)$, $3U/(4h)$, and $11U/(12h)$ for the round-off error R_{XF} if one, two, and three terms in (4.3.14).

The truncation error (called R_T) can be estimated by the first neglected term, where

$$\frac{1}{h}\mu\delta^{2k+1} f_0 \approx h^{2k} f^{(2k+1)}(x_0).$$

It has been mentioned several times (see, e.g., Example 3.3.15 in connection with the use of Richardson extrapolation for numerical differentiation) that irregular errors in the values of $f(x)$ are of much greater importance in numerical differentiation than in interpolation and integration.

Example 4.3.6.

Assume that k terms in the formula above is used to approximate $f'(x_0)$, where $f(x) = \ln x$, $x_0 = 3$, and $U = 10^{-6}$. Then $f^{(2k+1)}(3) = (2k)!/3^{2k+1}$, and for the truncation and round-off errors we get:

k	1	2	3
R_T	$0.0123h^2$	$0.00329h^4$	$0.00235h^6$
R_{XF}	$(1/2h)10^{-6}$	$(3/4h)10^{-6}$	$(11/12h)10^{-6}$

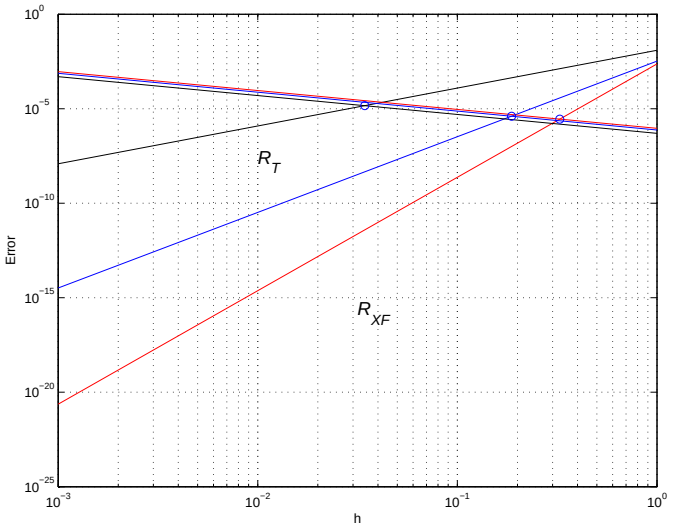


Figure 4.3.1. *Bounds for Truncation error R_T and roundoff error R_{XF} as functions of h for $u = 0.5 \cdot 10^{-6}$.*

In a log-log diagram the plots of R_T and R_{XF} versus h in Figure 4.2.2 are straight lines that illustrate quantitatively the Scylla and Charybdis situation (see explanation in Sec. 3.1.4); the truncation error increases, and the effect of the irregular error decreases with h. One sees how the choice of h, which minimizes the sum of the bounds for the two types of error, depends on u and k, and tells what accuracy can be obtained. The optimal step-lengths for $k = 1, 2, 3$ are $h = 0.0344$, $h = 0.1869$, and $h = 0.3260$, giving error bounds $2.91 \cdot 10^{-5}$, $8.03 \cdot 10^{-6}$, and $5.64 \cdot 10^{-6}$. Note that the optimal error bound with $k = 3$ is not much better than that for $k = 2$.

The effect of the pure rounding errors is important, though it should not be exaggerated. Using IEEE double precision with $u = 1.1 \cdot 10^{-16}$, one can obtain the first two derivatives very accurately by the optimal choice of h. The corresponding figures are $h = 2.08 \cdot 10^{-5}$, $h = 2.19 \cdot 10^{-3}$, and $h = 1.36 \cdot 10^{-2}$, giving the optimal errors bounds $1.07 \cdot 10^{-11}$, $1.52 \cdot 10^{-13}$, and $3.00 \cdot 10^{-14}$, respectively.

It is left to the user (Problem 12) to check and modify the experiments and conclusions indicated in this example. See also the appendix of Chapter 12, where similar questions are discussed in a more general context, namely differentiation for vector-valued functions of vector-valued arguments.

4.3.4 Fast Algorithms for Vandermonde Systems

Given distinct scalars $x_1, x_2, \ldots, x_n$, let V be the Vandermonde matrix

$$V = V(x_1, x_2, \ldots, x_n) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & \cdots & \vdots \\ x_1^{n-1} & x_2^{n-1} & \cdots & x_n^{n-1} \end{pmatrix}. \tag{4.3.16}$$

As shown in Sec. 4.1.1 the solution $a = V^{-T} f$ of the **dual Vandermonde system**

$$V^T a = f \tag{4.3.17}$$

gives the coefficients for the interpolating polynomial in the power basis. This polynomial can be computed, e.g., by Newton's interpolation formula in $O(n^2)$ operations. The related **primal Vandermonde systems**

$$Vy = b \tag{4.3.18}$$

arises in problems of determining approximation of linear functionals (see Example 4.1.1). We would like to have a stable and efficient method also for solving the primal system. One possibility would be to use the algorithm given in Sec. 4.2.2, which computes the inverse V^{-1} in about $6n^2$ operations and then form the product $V^{-1}b = y$.

We shall now derive a more efficient and accurate algorithm for solving primal Vandermonde systems. We start by expressing the solution of the dual problem

in terms of a matrix factorization. Using the power basis the unique polynomial satisfying the interpolation conditions $p(x_i) = f_i$, $i = 1 : n$, is

$$p(x) = (1, x, \ldots, x^{n-1})a,$$

where the coefficient vector a satisfies the linear system $V^T a = f$,

One of the most efficient ways to compute $p(x)$ is by Newton's interpolation formula, which uses the basis polynomials

$$p_1(x) = 1, \qquad p_k(x) = (x - x_1) \cdots (x - x_{k-1}), \quad k = 2 : n.$$

We write the polynomial in the form

$$p(x) = c_1 + c_2 p_2(x) + \cdots + c_n p_n(x).$$

where $c_j = [x_1, \ldots, x_{j-1}]f$. These divided differences can be recursively computed as described in Sec. 4.2.1. Then the coefficient vector a of $p(x)$ in the power basis

$$p(x) = a_1 + a_2 x + \cdots + a_n x^{n-1},$$

can be computed by Horner's rule. This is implemented in the algorithm below. Note that the matrix V^T is never formed and we only need storage for a few vectors. The operation count is $\frac{5}{2}n(n+1)$ flops.

Algorithm 4.3.1 Fast Dual Vandermonde Solver

```
function a = dvand(x,f)
% Newton's method for solving a dual Vandermonde system
% V^T(x_1,x_2,...,x_n)a = f.
  n = length(x);
  a = f;
  for k = 1:n-1
     for j = n:(-1):k+1
        a(j) = (a(j) - a(j-1))/(x(j) - x(j-k));
     end
  end
  for k = n-1:(-1):1
     for j = k:n-1
        a(j) = a(j) - x(k)*a(j+1);
      end
  end
```

To derive a corresponding algorithm for solving primal Vandermonde systems the above algorithm can be interpreted as a factorization of the matrix $(V^T)^{-1}$ into a product of diagonal and lower bidiagonal matrices. Let

$$D_k = \mathrm{d}iag\,(1, \ldots, 1, (x_{k+1} - x_1), \ldots, (x_n - x_{n-k})).$$

and define the matrices

$$L_k(x) = \begin{pmatrix} I_{k-1} & 0 \\ 0 & B_{n-k+1}(x) \end{pmatrix}, \quad k = 1 : n-1, \tag{4.3.19}$$

where

$$B_p(x) = \begin{pmatrix} 1 & & & \\ -x & 1 & & \\ & \ddots & \ddots & \\ & & -x & 1 \end{pmatrix} \in \mathbf{R}^{p\times p}, \tag{4.3.20}$$

Then the dual Vandermonde algorithm can be written in matrix terms as $c = U^T f$, $a = L^T c$, where

$$U^T = D_{n-1}^{-1} L_{n-1}(1) \cdots D_1^{-1} L_1(1), \tag{4.3.21}$$

$$L^T = L_1^T(x_1) L_2^T(x_2) \cdots L_{n-1}^T(x_{n-1}). \tag{4.3.22}$$

Since $a = V^{-T} f = L^T U^T f$, we have $V^{-T} = L^T U^T$.

We can now obtain a fast algorithm for solving a primal Vandermonde system $Vy = b$ as follows. Transposing the matrix factorization of V^{-T} gives $V^{-1} = UL$. Hence $y = V^{-1}b = U(Lb)$ and the solution to the primal system can be computed from $d = Lb$, $y = Ud$. Transposing (4.3.21)–(4.3.22) this gives

$$L = L_{n-1}(x_{n-1}) \cdots L_2(x_2) L_1(x_1)$$

$$U = L_1^T(1) D_1^{-1} \cdots L_{n-1}^T(1) D_{n-1}^{-1}.$$

This leads to an algorithm for solving primal Vandermonde systems. The operation count and storage requirement of this are the same as for dual system algorithm.

Algorithm 4.3.2 Fast Primal Vandermonde Solver

```
function y = pvand(x,b)
% Newton's method for solving a primal Vandermonde system
% V(x_1,x_2,...,x_n)y = b.
  n = length(x);
  y = b;
  for k = 1:n-1
     for j = n:(-1):k+1
        y(j) = y(j) - x(k)*y(j-1);
     end
  end
  for k = n-1:(-1):1
     for j = k+1:n
        y(j) = y(j)/(x(j) - x(j-k));
      end
    for j = k:n -1
```

```
        y(j) = y(j) - y(j+1);
    end
end
```

The above two algorithms are not only fast. Also they can give almost full relative accuracy in the solution of some Vandermonde systems, which are so ill-conditioned that Gaussain elimination with complete pivoting fails to produce a single correct digit. This was first observed by Björck and Pereyra [4], from which the following example is taken.

Example 4.3.7.

Consider a primal Vandermonde system $V_n y = b$, with

$$x_i = 1/(i+2), \qquad b_i = 1/2^{i-1}, \quad i = 1:n.$$

The exact solution can be shown to be

$$y_i = (-1)^{i-1}\binom{n}{i}(1+i/2)^{n-1}.$$

Let $\bar{y}_i$ be the solution computed by the primal Vandermonde algorithm and take as a measure of the relative error

$$e_n = \max_{1\le i\le n} |y_i - \bar{y}_i|/|y_i|.$$

Using a hexadecimal floating point arithmetic with $u = 16^{-13} = 2.22 \cdot 10^{-16}$ the following results were obtained:

n	5	10	15	20	25
e_n/u	4	5	10	54	81

The computed solution has small componentwise relative error, which is remarkable since, e.g., $\kappa(V_{10}) = 9 \cdot 10^{13}$.

A forward error analysis given by Higham [30], explains the surprisingly favorable results. If the points are positive and monotonically ordered

$$0 < x_1 < x_2 \cdots < x_n, \tag{4.3.23}$$

then the error in the solution $\bar{a}$ of a Vandermonde system $Vy = b$ computed by the primal algorithm can be bounded as

$$|\bar{a} - a| \le 5u|V^{-1}|\,|b| + O(u^2). \tag{4.3.24}$$

If the components of the right hand side satisfy $(-1)^n b_i \ge 0$, then $|V^{-1}|\,|b| = |V^{-1}b|$, and this bound reduces to

$$|\bar{a} - a| \le 5u|a| + O(u^2), \tag{4.3.25}$$

i.e. the solution is computed with small relative error indendent of the conditioning of V. A similar result holds for the dual algorithm. These good results can be shown to be related to the fact that when (4.3.23) holds, the matrix $V(x_1, x_2, \ldots, x_n)$ is **totally positive**, i.e. the determinant of every squares submatrix of V is positive; see [8].

The given algorithms has been generalized to confluent Vandermonde matrices (see Example 4.3.1) and other classes of Vandermonde-like matrices.

4.3.5 Multidimensional Interpolation

Much of the theory of the introduction can be generalized to other interpolation problems than problems with polynomials in one variable, but one cannot be sure that there is unconditionally a unique solution to the problem. It may not be enough to require that the points are distinct.

Example 4.3.8.

The interpolation by a linear function in two variables,

$$p(x_i, y_i; c) = c_1 + c_2 x_i + c_3 y_i = f_i, \quad i = 1:3,$$

leads to the linear system $Vc = f$, where

$$V = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}, \quad c = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}, \quad f = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}.$$

This interpolation problem has exactly one solution if V is nonsingular, i.e. when $\det(V) \neq 0$. But $\frac{1}{2}\det(V)$ is just the area of the triangle with vertices (x_i, y_i), $i = 1:3$. If this area is zero then the three points lie on a line and the problem has either infinitely many solutions, or no solution.

The simplest way to generalize interpolation to functions of several variables is to use repeated one-dimensional interpolation, i.e. to work with one variable at a time. The following formula for **bilinear interpolation**,

$$\begin{aligned} f(x_0 + ph, y_0 + qh) &\approx (1-q)\varphi(y_0) + q\varphi(y_0 + k), \\ \varphi(y) &= (1-p)f(x_0, y) + pf(x_0 + h, y). \end{aligned}$$

is the simplest example. After simplification it can written as

$$\begin{aligned} f(x_0 + ph, y_0 + qh) \approx{}& (1-p)(1-q)f_{0,0} + p(1-q)f_{1,0} \\ &+ (1-p)qf_{0,1} + pqf_{1,1}, \end{aligned} \tag{4.3.26}$$

where we have used the notation $f_{ij} = f(x_0 + ih, y_0 + jk)$, $i, j \in \{0, 1\}$. This formula is exact for functions of the form $f(x, y) = a + bx + cy + dxy$, and from equation (4.2.50) we obtain the error bound,

$$\max_{(x,y)\in R} \frac{1}{2}\big(p(1-p)h^2|f_{xx}| + q(1-q)h^2|f_{yy}|\big), \qquad 0 \leq p, q \leq 1,$$

where $R = \{(x, y) : x_0 \le x \le x_0 + h, y_0 \le y \le y_0 + k\}$. The formula for bilinear interpolation can easily be generalized by using higher order interpolation in the x and/or y direction.

In the following we consider explicitly only the case of two dimension, since corresponding formulas for three and more dimensions are analogous.

A **rectangular grid** in the (x, y)-plane with grid spacings h, k in the x and y directions, respectively, consists of points $x_i = x_0 + ih$, $y_i = y_0 + ik$. In the following we use the notation $f(x_i, y_j) = f_{ij}$.

Central difference approximations for partial derivatives using function values can be obtained by working with one variable at a time,

$$\frac{\partial f}{\partial x} = \frac{1}{2h}(f_{i+1,j} - f_{i-1,j}) + O(h^2), \quad \frac{\partial f}{\partial y} = \frac{1}{2k}(f_{i,j+1} - f_{i,j-1}) + O(k^2).$$

For second order derivatives

$$\frac{\partial^2 f}{\partial x^2} = \frac{1}{h^2}(f_{i+1,j} - 2f_{ij} + f_{i-1,j}),$$

and a similar formula holds for $\partial^2 f/\partial y^2$.

Formulas of higher accuracy can also be obtained by operator techniques, based on an operator formulation of Taylor's expansion (see Theorem 4.6.6,

$$f(x_0 + h, y_0 + k) = \exp\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right) f(x_0, y_0) \tag{4.3.27}$$

From this we obtain

$$\begin{aligned} f(x_0 + h, y_0 + k) = f_{0,0} &+ \Big(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\Big) f_{0,0} \\ &+ \Big(h^2\frac{\partial^2}{\partial x^2} + 2hk\frac{\partial^2}{\partial x \partial y} + k^2\frac{\partial^2}{\partial y^2}\Big) f_{0,0} + O(h^2 + k^2). \end{aligned}$$

An interpolation formula valid for all quadratic functions can be obtained by replacing in Taylor's formula the derivatives by difference approximations valid for quadratic polynomials,

$$\begin{aligned} f(x_0 + ph, y_0 + qh) \approx f_{0,0} &+ \frac{1}{2}p(f_{1,0} - f_{-1,0}) + \frac{1}{2}q(f_{0,1} - f_{0,-1}) \\ &+ \frac{1}{2}p^2(f_{1,0} - 2f_{0,0} + f_{-1,0}) \\ &+ \frac{1}{4}pq(f_{1,1} - f_{1,-1} - f_{-1,1} + f_{-1,-1}) \\ &+ \frac{1}{2}q^2(f_{0,1} - 2f_{0,0} + f_{0,-1}). \end{aligned} \tag{4.3.28}$$

This formula uses function values in nine points. (The proof of the expression for approximating the mixed derivative $\dfrac{\partial^2}{\partial x \partial y} f_{0,0}$ is left as an exercise, Problem 2.

Review Questions

1. What is meant by Hermite interpolation (osculatory interpolation)? Prove the uniqueness result for the Hermite interpolation problem.

2. (a) Write down the confluent Vandermonde matrix for the Hermite cubic interpolation problem.

(b) Express the divided difference $[x_0, x_0, x_1, x_1]f$ in terms of f_0, f_0',and f_1,f_1'.

3. How is bilinear interpolation performed? What is the order of accuracy?

Problems and Computer Exercises

1. (a) Construct the divided difference scheme (unscaled or scaled) for the simplest Hermite interpolation problem, where the given data are $f(x_i)$, $f'(x_i)$, $i = 0, 1$; $x_1 = x_0 + h$. Prove all the formulas concerning this problem that are stated at the end of Sec. 4.3.2.

(b) For $f(x) = (1 + x)^{-1}$, $x_0 = 1$, $x_1 = 1.5$, compute $f(1.25)$ by Hermite interpolation. Compare the error bound and the actual error.

(c) Show that for Hermite interpolation

$$|f'(x) - p'(x)| \leq \frac{h^3}{72\sqrt{3}}\Big(\max_{x\in[x_0,x_1]} |f^{(iv)}(x)| + O(h|f^{(v)}(x)|)\Big).$$

Hint: $\frac{d}{dx}[x_0, x_0, x_1, x_1, x]f = [x_0, x_0, x_1, x_1, x, x]f \leq \ldots$.

2. Given $x_i, y(x_i), y'(x_i)$, $x_i = x_0 + ih$, $i = 1, 2, 3$. Let $p \in \mathcal{P}_6$ be the Hermite interpolation polynomial to these data.

(a) Find the remainder term, and show that the interpolation error for $x \in [x_1, x_3]$ does not exceed $h^6 \max |f^{(6)}(x)|/4860$ in magnitude.

(b) Write a program that computes $p(x_1 + 2jh/k)$, $j = 0 : k$.

COMMENT: This is one of several possible procedures for starting a multistep method for an ordinary differential equation $y' = f(x, y)$. Two steps with an accurate one-step method, provide values of y, y', and this program then produces starting values (y only) for the multistep method.

3. Give a short and complete proof of the uniqueness of the interpolation polynomial for distinct points, by the use of the ideas of the proof of Theorem 4.3.1.

4. Derive an approximate formula for $f'(x_0)$ when the values $f(x_{-1}), f(x_0), f(x_1)$ are given at three *non-equidistant* points. Give an approximate remainder term. Check the formula and the error estimate on an example of your own choice.

5. (a) Given a sequence of function values $f_1, f_2, f_3 \ldots$ at equidistant points $x_j = x_0 + jh$. Assume that $\min f_j = f_n$, and let $p(x)$ be the quadratic interpolation

polynomial determined by f_{n-1}, f_n, f_{n+1}. Show that

$$\min p(x) = f_n - \frac{(\mu\delta f_n)^2}{2\delta^2 f_n}, \quad \text{at} \quad x = x_n - h\frac{\mu\delta f_n}{\delta^2 f_n},$$

and that the error of the minimum value can be bounded by $\max|\Delta^3 f_j|/\sqrt{243}$, where j is in some neighborhood of n. Why and how is the estimate of x less accurate?

(b) Write a handy program that includes the search all local maxima and minima. Sketch or work out improvements of this algorithm, perhaps with ideas of inverse interpolation and with cubic interpolation. And perhaps for non-equidistant data.

6. (a) Compute by bilinear interpolation $f(0.5, 0.25)$ when

$$f(0,0) = 1, \quad f(1,0) = 2, \quad f(0,1) = 3, \quad f(1,1) = 5.$$

(b) Set $c = (c_1, c_2, c_3, c_4, c_5, c_6)^T$,

$$p(x, y; c) = c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 xy + c_6 y^2.$$

Consider the interpolation problem: Given x_i, y_i, f_i, $i = 1:6$; try to find c, so that $p(x_i, y_i; c) = f_i$, $i = 1:6$.
Choose x_i, y_i, f_i, $i = 1:6$ by 18 independent random numbers, solve the linear system $p(x_i, y_i; c) = f_i$, $i = 1:6$, look at $\max|c_i|$. Repeat this (say) 25 times. You have a fair chance to avoid singular cases, or cases where $\max|c_i|$ is very large.

(c) Now choose (x_i, y_i) as 6 distinct points on some circle in $\mathbf{R}^2$, and choose f_i at random. This should theoretically lead to a singular matrix. Explain why, and find experimentally the rank (if your software has convenient commands or routines for that). Find a general geometric characterization of the sextuples of points (x_i, y_i), $i = 1:6$, that lead to singular interpolation problems.

Hint: Brush up your knowledge of conic sections.

7. Derive a formula for $f''_{xy}(0,0)$ using f_{ij}, $|i| \le 1, |j| \le 1$, which is exact for all quadratic functions.

4.4 Piecewise Polynomial Interpolation

4.4.1 Bernstein Polynomials

Parametric curves are often used to find a functional form of a curve (or surface) given geometrically by a set of points. Let $c(t)$, $t \in [0,1]$ a parametric curve connecting two points p_0 and p_1 in $\mathbf{R}^{\mathbf{d}}$, so that $p_0 = c(0)$ and $p_1 = c(1)$. In the simplest case we can take $c(t)$ to be linear and write

$$c(t) = (1-t)p_0 + tp_1.$$

If extended to a set of points $p_0, \ldots, p_n$, $n > 1$ this will not give a smooth curve and is therefore of limited interest. We now generalize this approach and take $c(t)$ to be a polynomial of degree n.

The **Bernstein polynomials**[12] are defined by

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0 : n. \tag{4.4.1}$$

Using the binomial theorem we have

$$1 = ((1-t)+t)^n = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} = \sum_{i=0}^{n} B_i^n(t),$$

that is, the Bernstein polynomials of degree n are nonnegative and give a partition of unity. The Bernstein polynomials of degree n form a basis for the space of polynomials of degree $\le n$.

For $n = 3$ the four cubic Bernstein polynomials are

$$B_0^3 = (1-t)^3, \quad B_1^3 = 3t(1-t)^2, \quad B_2^3 = 3t^2(1-t), \quad B_3^3 = t^3. \tag{4.4.2}$$

are plotted in Figure 4.4.1.

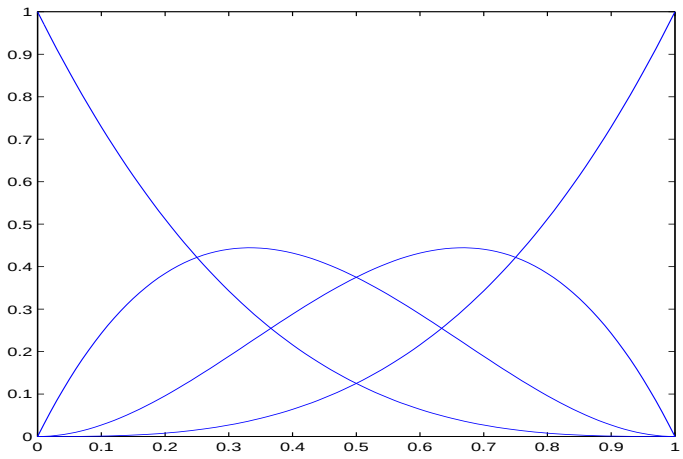


Figure 4.4.1. *Bernstein polynomials.*

Some important properties of the Bernstein polynomials are given in the following theorem.

[12] Sergi NatanovičBernštein (1880–1968) Russian mathematician, who made major contributions to polynomial approximation. In 1911 he introduced the polynomials named after him.

Theorem 4.4.1. *The Bernstein polynomials $B_i^n(t)$ have the following properties:*

1. $B_i^n(t) > 0, \quad t \in (0,1)$ *(nonnegativity);*
2. $B_i^n(t) = B_{n-i}^n(1-t)$ *(symmetry);*
3. *The Bernstein polynomials $B_i^n(t)$ have a unique maximum value at $t = i/n$ on $[0,1]$;*
4. *The Bernstein polynomials satisfy the following recursion formula*

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t), \quad i = 0:n. \tag{4.4.3}$$

Proof. The first three properties follows directly from the definition (4.4.1). The recursion formula is a consequence of the relation

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}$$

between the binomial coefficients. □

Starting with $B_0^0(t) = 1$, and setting $B_{-1}^n(t) = B_{n+1}^n(t) = 0$ this recursion can be used to evaluate the Bernstein polynomials at a given point t.

4.4.2 Parametric Bézier Curves

Given a set of $n+1$ **control points** p_i, $i = 0:n$, the Bézier curve is given by

$$c(t) = \sum_{i=0}^{n} p_i B_i^n(t), \quad t \in [0,1]. \tag{4.4.4}$$

The Bézier curve interpolates the first and last control points p_0 and p_1. It follows directly from the form of (4.4.4) that applying an affine transformation to $c(t)$ can be performed simply by applying the same transformation to the control points. Hence the Bézier curve has the desirable property that it is invariant under translations and rotations.

Bézier curves are a major tool in computer graphics, where usually $p_i \in \mathbf{R}^2$ or $\mathbf{R}^3$. One important application is in computer aided design (CAD) systems, used, e.g., in the auto industry. Often a curve is constructed by smoothly patching together several Bézier curves of lower order.

Example 4.4.1. A quadratic Bézier curve is given by

$$c(t) = (1-t)^2 p_0 + 2t(1-t)p_1 + t^2 p_2, \quad t \in [0,1].$$

Clearly $c(0) = p_0$ and $c(1) = p_2$. For $t = 1/2$ we get

$$c(1/2) = \frac{1}{2}\Big(\frac{p_0 + p_2}{2} + p_1\Big).$$

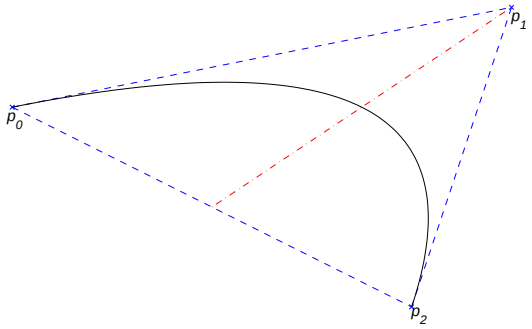


Figure 4.4.2. *Quadratic Bézier curve with control points.*

Hence we can construct the point $c(1/2)$ geometrically as the intersection between the the midpoint of the line between p_0 and p_2 and the point p_1; see Figure 4.4.2.

The **Bézier polygon** is the closed piecewise linear curve connecting the control points p_i and p_{i+1}, $i = 0 : n-1$ and finally p_n and back to p_0. In Figure 4.4.2 this is the polygon formed by the dashed lines. This polygon provides a rough idea about the shape of the Bézier curve.

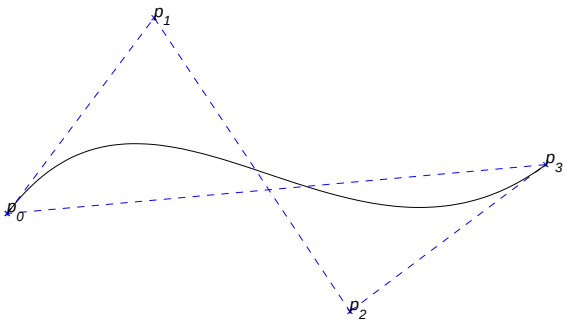


Figure 4.4.3. *Cubic Bézier curve with control points* $P_0, \ldots, P_3$.

Definition 4.4.2. *A set S in $\mathbf{R}^d$ is called convex if for any points $x, y \in S$, the straight line*

$$\{tx + (1-t)y \mid t \in (0,1)\}$$

is also contained in S; The **convex hull** *of a set S in $\mathbf{R}^d$ is the smallest convex subset of $\mathbf{R}^d$, which contains S.*

From the definition (4.4.4) of the Bézier curve it follows that for all $t \in [0,1]$, the curve $c(t)$ is a convex combination of the control points. Therefore $c(t)$ lies within the convex hull of the control points. Often, but not always, the convex hull is the region enclosed Bézier polygon; cf. Figures 4.4.2–3.

The variation of a function in an interval $[a,b]$ is the least upper bound on the sum of the oscillations in the closed subintervals $[a,x_1]$, $[x_1,x_2]$, $\ldots$, $[x_n,b]$, for all possible such subdivisions. The Bézier curve is variation diminishing. In particular if the control points p_i are monotonic, so is $c(t)$. Further, if p_i are convex (concave) so is $c(t)$.

Usually all control points are not known in advance but the curves shape is controlled by moving the control points until the curve has the desired shape. For example, in the quadratic case moving p_1 has a direct and intuitive effect on the curve $c(t)$. An advantage of the Bernstein basis for representing polynomials is that the coefficients (control points) is closely related to the shape of the curve. This is not the case when using a monomial or Chebyshev basis.

Theorem 4.4.3. *The Bézier curve $c(t)$ is tangent to $p_1 - p_0$ and $p_n - p_{n-1}$ for $t = 0$ and $t = 1$, respectively.*

Proof. To show this we compute the derivative of the Bernstein polynomial (4.4.1)

$$\frac{d}{dt}B_i^n(t) = \begin{cases} -nB_0^{n-1}(t), & \text{if } i = 0 \\ n\left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right), & \text{if } 0 < i < n; \\ nB_{n-1}^{n-1}(t), & \text{if } i = n. \end{cases}$$

This follows from

$$\frac{d}{dt}B_i^n(t) = \binom{n}{i}\left(it^{i-1}(1-t)^{n-i} - (n-i)t^i(1-t)^{n-i-1}\right),$$

and using the definition of the Bernstein polynomials. Setting $t = 0$ we find that $\frac{d}{dt}B_i^n(0) = 0$, $i > 1$, and therefore from (4.4.4)

$$\frac{d}{dt}c(t) = n(p_1 - p_0),$$

which shows the statement for $t = 0$. The result for $t = 1$ follows from symmetry. □

More generally, at a boundary point the kth derivative of the Bézier curve depends only on the k closest control points. This fact is useful for smoothly joining together several pieces of Bézier curves.

To evaluate the Bézier curve at $t \in [0,1]$ we use the recursion formula (4.4.3) to obtain

$$c(t) = \sum_{i=0}^{n} p_i B_i^n(t)$$

$$= (1-t)\sum_{i=0}^{n-1} p_i B_i^{n-1}(t) + t\sum_{i=1}^{n} p_i B_{i-1}^{n-1}(t)$$
$$= \sum_{i=0}^{n-1} \big((1-t)p_i + tp_{i+1}\big) B_i^{n-1}(t) = \sum_{i=0}^{n-1} p_i^{(1)}(t) B_i^{n-1}(t)$$

where we have introduced the new auxiliary control points

$$p_i^{(1)}(t) = (1-t)p_i + tp_{i+1}, \quad i = 0 : n-1,$$

as convex combinations (depending on t) of the original control points. Using this result we can successively lower the grade of the Bernstein polynomial until we arrive at $B_0^0 = 1$. This gives a recursion scheme for the auxiliary control points due to de Casteljau:

$$\begin{aligned} p_i^{(0)}(t) &= p_i, \quad i = 0 : n \\ p_i^{(r)}(t) &= (1-t)p_i^{(r-1)}(t) + tp_{i+1}^{(r-1)}(t), \quad i = 0 : n-r. \end{aligned} \tag{4.4.5}$$

It follows

$$c(t) = \sum_{i=0}^{n-r} p_i^{(r)}(t) B_i^{n-r}(t), \quad r = 0 : n \tag{4.4.6}$$

and in particular $c(t) = p_0^{(n)}$.

De Casteljau's algorithm can be arranged in a triangular array

$$\begin{array}{lllll} p_0 = p_0^{(0)} & & & & \\ & p_0^{(1)} & & & \\ p_1 = p_1^{(0)} & & p_0^{(2)} & & \\ & p_1^{(1)} & & & \\ p_2 = p_2^{(0)} & \vdots & p_1^{(2)} & \ddots & \\ \vdots & \vdots & \vdots & \vdots & p_0^{(n)} \\ \vdots & p_{n-2}^{(1)} & \vdots & & \\ p_{n-1} = p_{n-1}^{(0)} & & p_{n-2}^{(2)} & & \\ & p_{n-1}^{(1)} & & & \\ p_n = p_n^{(0)} & & & & \end{array} \tag{4.4.7}$$

Since at each step the new control points are convex combinations of the previous control points, de Casteljau's algorithm is very stable. It uses about n^2 operations and so is less efficient than Horner's algorithm for evaluating a polynomial in the monomial basis.

The kth derivative of $c(t)$ is also available from the de Casteljau scheme. It holds that

$$c'(t) = n(p_1^{n-1} - p_0^{n-1}),$$

$$c''(t) = n(n-1)(p_2^{n-2} - 2p_1^{n-2} + p_0^{n-2}), \ldots,$$

and in general

$$c^{(k)}(t) = \frac{n!}{(n-k)!}\Delta^k p_0^{n-k}, \quad 0 \le k \le n, \tag{4.4.8}$$

where the difference operates on the lower index i.

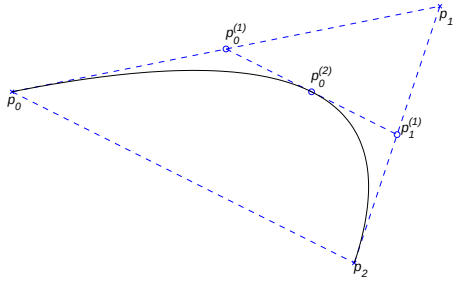


Figure 4.4.4. *Casteljau's algorithm for* $n = 2$, $t = \frac{1}{2}$.

De Casteljau's algorithm is illustrated for the quadratic case in Figure 4.4.4, where the following geometric interpretation can be observed. In the interval $[0, t]$ the Bézier curve is represented by a quadratic spline with control points $p_0, p_0^{(1)}, p_0^{(2)}$. In the remaining interval $[t, 1]$ it is represented by a quadratic spline with control points $p_0^{(2)}, p_1^{(1)}, p_2$. Note that these two sets of control points lies closer to the curve $c(t)$. After a few more subdivisions it will be hard to distinguish the polygon joining the control points form the curve.

4.4.3 Splines

The name **spline** comes from a very old technique in drawing smooth curves in which a thin strip of wood, called a draftsman's spline, is bent so that it passes trough a given set of points, see Figure 4.5.5. The points of interpolation are called **knots** and the spline is secured at the knots by means of lead weights called **ducks**. Before the computer age splines were used in ship building and other engineering designs.

A mathematical model of a spline was given by Daniel Bernoulli (1742) and Euler (1744)[13]. By Hamilton's principle the shape the spline will take is such that its elastic strain energy is minimized. The strain energy of a spline $y = s(x)$, $x \in [a, b]$ in the plane is given by

$$E(s) = \int_a^b \kappa(x)^2 \, dx$$

[13]Euler derived the differential equation satisfied by the spline using techniques now known as calculus of variation and Lagrange multipliers. When Euler did this work Lagrange was still a small child!

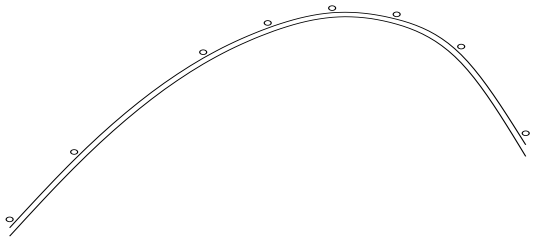


Figure 4.4.5. *The original spline.*

where

$$\kappa(x) = \frac{s''(x)}{(1 + (s'(x))^2)^{3/2}}.$$

is the curvature of the spline. For slowly varying deflections, i.e. when $(s'(x))^2$ is approximately constant, the approximation

$$E(s) =\approx const \cdot \int_a^b s''(x)^2 \, dx,$$

Under this assumption, according to elasticity theory, $s(x)$ is built up of **piecewise** third degree polynomials (cubic polynomials) in such a way that $s(x)$ *and its two first derivatives are everywhere continuous.* Let x_i, $i = 0 : m$ be the points the spline is forced to interpolate. Then the third derivative can have discontinuities at the points x_i. Such a function is called a **cubic spline function**, or shorter, a **cubic spline** The points x_i, $i = 0 : m$, are called **breakpoints** or **knots**.

The mathematical concept of spline functions was introduced in 1946 by Schoenberg in the seminal paper [44]. The importance of the B-spline basis for spline approximation (see Sec. sec4.4.6) was also first appreciated by Schoenberg. These were not used in practical calculations for general knot sequences until the early seventies, when a stable recurrence relation was established independently by de Boor [6] and Cox [17].

Spline functions are now used extensively in computer aided design (CAD), where curves and surfaces have to be represented mathematically, so that they can be manipulated and visualized easily. Important applications occur in computer-aided design, analysis and manufacturing as well as in aircraft and automotive industries. Spline functions can also be used in the numerical treatment of boundary-value problems for differential equations.

With the use of splines, there is no reason to fear equidistant data, as opposed to the situation with higher-degree polynomials. Also, if the function to be approximated is badly behaved somewhere then, using spline approximation with properly chosen knots, the effect of this can be confined locally, allowing good approximation elsewhere in the interval. In the following we restrict ourself to consider curves in the plane. For more information on spline approximations of curves and surfaces the reader is referred to de Boor [7], where also FORTRAN programs for computations with spline functions can be found, and Dierckx [22].

We have seen that it is often not efficient to approximate a given function by a single polynomial over its entire range. On the other hand, polynomials of low degree can give good approximations *locally* in a small interval. Therefore it is natural to consider approximations by piecewise polynomials of different degrees of global continuity.

We first consider the simplest curve interpolating given values $y_i = f(x_i) \in [a, b$ on a grid

$$\Delta = \{a = x_0 < x_1 < \cdots < x_m = b\}$$

is by a broken line $s(x)$, where

$$s(x) = q_i(x) = y_{i-1} + d_i(x - x_i), \quad x \in [x_{i-1}, x_i), \quad i = 1 : m \tag{4.4.9}$$

Here

$$h_i = x_i - x_{i-1}, \qquad d_i = [x_{i-1}, x_i]f(x) = (y_i - y_{i-1})/h_i. \tag{4.4.10}$$

i.e. d_i is the divided difference of f at $[x_{i-1}, x_i]$. If $f \in C^2[a, b]$ then the error satisfies (see (4.2.50)).

$$|f(x) - s(x)| \le \frac{1}{8} \max_i \Big(h_i^2 \max_{x \in [x_{i-1}, x_i]} |f''(x)| \Big). \tag{4.4.11}$$

Hence, we can make the error arbitrary small by decreasing $\max_i h_i$. An important property of interpolation with a piecewise affine function is that it preserves monotonicity and convexity of the interpolated function.

The broken line interpolating function has a discontinuous first derivative at the knots, which makes it unsuitable for many applications. To get better smoothness piecewise polynomials of higher degree need to be used. Although piecewise quadratic approximation is sometimes useful, piecewise *cubic* polynomials with continuous second derivatives are by far the more important (see Figure 4.4.6.

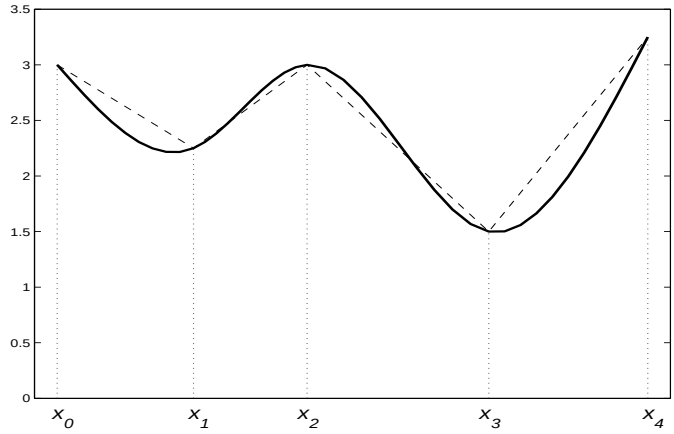


Figure 4.4.6. *Broken line and cubic spline interpolation.*

A cubic polynomial $q_i(x)$ on the interval $[x_{i-1}, x_i)$ is uniquely determined by the values of the function and its first derivative at the end points of the interval. This follows from the more general result on Hermite interpolation in Theorem 4.3.1. By (4.3.11), translated to the notation in (4.4.10), the cubic $q_i(x)$ can be written in the form

$$q_i(x) = \theta y_i + (1-\theta)y_{i-1} + h_i\theta(1-\theta)\left[(k_{i-1}-d_i)(1-\theta) - (k_i - d_i)\theta\right], \quad (4.4.12)$$

where h_i, d_i, $i = 1:m$, are as in (4.4.10),

$$\theta = \frac{x - x_{i-1}}{h_i} \in [0,1), \quad x \in [x_{i-1}, x_i), \quad (4.4.13)$$

is a local variable and $k_i = q_i'(x_i)$.

If the interpolating spline $s(x)$ is to be evaluated at many points, a form that is more efficient to use than (4.4.12) is the piecewise polynomial form (**pp form**)

$$q_i(x) = y_{i-1} + a_{1i}(x - x_{i-1}) + a_{2i}(x - x_{i-1})^2 + a_{3i}(x - x_{i-1})^3, \quad (4.4.14)$$

$i = 1:m$. From (4.4.12) we obtain after some calculation

$$\begin{aligned} a_{1i} &= q_i'(x_{i-1}) = k_{i-1}, \\ a_{2i} &= \tfrac{1}{2}q_i''(x_{i-1}) = (3d_i - 2k_{i-1} - k_i)/h_i, \\ a_{3i} &= \tfrac{1}{6}q_i'''(x_{i-1}) = (k_{i-1} + k_i - 2d_i)/h_i^2. \end{aligned} \quad (4.4.15)$$

Using Horner's scheme $q_i(x)$ can be evaluated from (4.4.14) using only four multiplication.

With piecewise cubic polynomials we can interpolate given function values and first derivatives on the grid Δ. By construction the interpolating piecewise cubic function $s(x)$ will have continuous first derivatives. If $f \in C^4[a,b]$ then it follows from the remainder term (4.3.11) that the error satisfies

$$|f(x) - s(x)| \le \frac{1}{384}\max_i \Big(h_i^4 \max_{x\in[x_{i-1},x_i]} |f^{(iv)}(x)|\Big). \quad (4.4.16)$$

It can be shown (Problem 1c of Sec. 4.3) that also the first derivative of $s(x)$ is a good approximation to $f'(x)$. If $f \in C^5[a,b]$ we have

$$|f'(x) - s'(x)| \le \frac{1}{72\sqrt{3}}\max_i \Big(h_i^3 \max_{x\in[x_i,x_{i+1}]} |f^{(iv)}(x) + O(h_i f^{(5)}(x))|\Big). \quad (4.4.17)$$

Sometimes it is useful to consider the values k_i, $i = 0:m$, as parameters which are used to give the interpolating function the desired shape. In the next section we show that it is possible to choose these parameters such that the interpolating function $s(x)$ also has a continuous *second* derivative.

We shall now formally define a spline function of order $k \ge 1$.

Definition 4.4.4.

Let $\Delta = \{a = x_0 < x_1 < \cdots < x_m = b\}$ be a subdivision of the interval $[a,b]$. A spline function on Δ of order $k \ge 1$ (degree $k-1 \ge 0$), is a real function s with properties:

(a) *For $x \in [x_i, x_{i+1}]$, $i = 0 : m-1$, $s(x)$ is a polynomial of degree $< k$.*

(b) *For $k = 1$, $s(x)$ is a piecewise constant function. For $k \geq 2$, $s(x)$ and its first $k-2$ derivatives are continuous on $[a, b]$, i.e. $s(x) \in C^{k-2}[a, b]$.*

We denote by $S_{\Delta,k}$ the set of all spline functions of order k on Δ. From the definition it follows that if $s_1(x)$ and $s_2(x)$ are spline functions of the same degree, so is $c_1 s_1(x) + c_2 s_2(x)$. Thus $S_{\Delta,k}$ is a *linear space*.

Examples of elements of $S_{\Delta,k}$ are the **truncated power** functions

$$(x - x_j)_+^{k-1}, \quad j = 1 : m-1,$$

and all their linear combinations. Moreover, $\mathcal{P}_k$ is a linear subspace of $S_{\Delta,k}$.[14] Conversely, together these functions span $S_{\Delta,k}$. All we need for the first subinterval is a basis of $\mathcal{P}_k$, e.g., the power basis $\{1, x, \ldots, x^{k-1}\}$. Further, all we need for each additional subinterval $[x_j, x_{j+1})$, $j = 1 : m-1$, is the new basis function $(x - x_j)_+^{k-1}$. One can show that these $k + m - 1$ functions are linearly independent. The dimension of the linear space thus is $k + m - 1$.

4.4.4 Cubic Spline Interpolation

In the following we shall first study cubic spline functions which *interpolate* a given function $f(x)$ at the grid Δ, i.e. the space $S_{\Delta,4}$. By definition a cubic spline consists of cubic polynomials pieced together in such a way that their values and first *two* derivatives coincide at the knots. In contrast to Hermite interpolation, the cubic polynomial in each subinterval will now depend on *all data points*.

Theorem 4.4.5.

Every cubic spline function, with knots $a = x_0 < x_1 < \cdots < x_m = b$, which interpolates the function $y = f(x)$,

$$s(x_i) = f(x_i) = y_i, \quad i = 0 : m,$$

equals for $x \in [x_{i-1}, x_i)$, $i = 1 : m$ a third degree polynomial of the form (4.4.12)*. The $m + 1$ parameters k_i, $i = 0 : m$, satisfy $m - 1$ linear equations*

$$h_{i+1}k_{i-1} + 2(h_i + h_{i+1})k_i + h_i k_{i+1} = 3(h_i d_{i+1} + h_{i+1} d_i), \quad i = 1 : m-1, \tag{4.4.18}$$

where $h_i = x_i - x_{i-1}$, $d_i = (y_i - y_{i-1})/h_i$.

Proof. We require the second derivative of the spline $s(x)$ to be continuous at x_i, $i = 1 : m-1$. We have

$$s(x) = \begin{cases} q_i(x), & x \in [x_{i-1}, x_i), \\ q_{i+1}(x), & x \in [x_i, x_{i+1}), \end{cases}$$

[14] Recall the notation $(x-u)_+^j = \max\{x-u, 0\}$ that was introduced in Sec. 3.2.3 in connection with the Peano kernel.

where $q_i(x)$ is given by (4.4.14)–(4.4.15). Differentiating $q_i(x)$ twice we get $\frac{1}{2}q_i''(x) = a_{2,i} + 3a_{3,i}(x - x_{i-1})$, and putting $x = x_i$

$$\tfrac{1}{2}q_i''(x_i) = a_{2,i} + 3a_{3,i}h_i = (k_{i-1} + 2k_i - 3d_i)/h_i. \tag{4.4.19}$$

Replacing i by $i+1$ we get $\frac{1}{2}q_{i+1}''(x) = a_{2,i+1} + 3a_{3,i+1}(x - x_i)$, and hence

$$\tfrac{1}{2}q_{i+1}''(x_i) = a_{2,i+1} = (3d_{i+1} - 2k_i - k_{i+1})h_{i+1}. \tag{4.4.20}$$

These last two expressions must be equal, which gives the conditions

$$\frac{1}{h_i}(k_{i-1} + 2k_i - 3d_i) = \frac{1}{h_{i+1}}(3d_{i+1} - 2k_i - k_{i+1}), \quad i = 1 : m-1. \tag{4.4.21}$$

Multiplying both sides by $h_i h_{i+1}$ we get (4.4.18). □

The conditions (4.4.18) are $(m-1)$ linearly independent[15] equations for the $(m+1)$ unknowns k_i, $i = 0 : m$. Two additional conditions are therefore needed to uniquely determine the interpolating spline. The four most important choices are discussed below.

(i) If the derivatives at the end points are known we can take

$$k_0 = f'(a), \qquad k_m = f'(b). \tag{4.4.22}$$

The corresponding spline function $s(x)$ is called the **complete cubic spline interpolant**. If k_0 and k_m are determined by numerical differentiation with a truncation error that is $O(h^4)$, we call the spline interpolant **almost complete**. For example, k_0 and k_m may be the sum of (at least) four terms of the expansions

$$Df(x_0) = \frac{1}{h}\ln(1+\Delta)y_0, \qquad Df(x_m) = -\frac{1}{h}\ln(1-\nabla)y_m,$$

into powers of the operators Δ and ∇, respectively.[16]

(ii) A physical spline is straight outside the interval $[a, b]$, i.e. $s''(x) = 0$ for $x \le a$ or $x \ge b$. Thus $q_1''(x_0) = q_m''(x_m) = 0$. From (4.4.20) and (4.4.19) we have

$$\tfrac{1}{2}q_i''(x_{i-1}) = (3d_i - 2k_{i-1} - k_i)/h_i, \quad \tfrac{1}{2}q_i''(x_i) = -(3d_i - k_{i-1} - 2k_i)/h_i.$$

Setting $i = 1$ in the first equation and $i = m$ in the second gives the two conditions

$$\begin{aligned} 2k_0 + k_1 &= 3d_1 \\ k_{m-1} + 2k_m &= 3d_m. \end{aligned} \tag{4.4.23}$$

[15] The equations are strictly row diagonally dominant (see Sec. 7.4.1) and therefore linearly independent

[16] Two terms of the *central* difference expansion in (3.3.45) or one *Richardson* extrapolation, see Example 3.4.16, give higher accuracy, but need extra function values outside the grid Δ.

The corresponding approximating spline is called the **natural spline interpolant**. It should be stressed that when a cubic spline is used for the approximation of a smooth function, these boundary conditions are *not natural*!

(iii) If the end point derivatives are not known, a convenient condition is to require that $s'''(x)$ be continuous across the first and last interior knots x_1 and x_{m-1}. Hence $q_1(x) = q_2(x)$ and $q_{m-1}(x) = q_m(x)$. Then x_1 and x_{m-1} are no longer knots, and these conditions are known as "**not a knot**" conditions. From (4.4.15) we obtain,

$$\tfrac{1}{6}q_i'''(x) = a_{3i} = (k_{i-1} + k_i - 2d_i)/h_i^2, \quad x \in [x_{i-1}, x_i), \quad i = 1:m.$$

Hence the condition $q_1''' = q_2'''$ gives $(k_0 + k_1 - 2d_1)/h_1^2 = (k_1 + k_2 - 2d_2)/h_2^2$, or

$$h_2^2 k_0 + (h_2^2 - h_1^2)k_1 - h_1^2 k_2 = 2(h_2^2 d_1 - h_1^2 d_2).$$

Since this equation would destroy the tridiagonal form of the system, we use (4.4.18), with $i = 1$ to eliminate k_2. This gives the equation

$$h_2 k_0 + (h_2 + h_1)k_1 = 2h_2 d_1 + \frac{h_1(h_2 d_1 + h_1 d_2)}{h_2 + h_1}. \tag{4.4.24}$$

If the right boundary condition is treated similarly we get

$$(h_{m-1} + h_m)k_{m-1} + h_{m-1}k_m = 2h_{m-1}d_m + \frac{h_m(h_{m-1}d_m + h_m d_{m-1})}{h_{m-1} + h_m}. \tag{4.4.25}$$

(iv) If the spline is used to represent a periodic function, then $y_0 = y_m$ and the boundary conditions

$$s'(a) = s'(b), \qquad s''(a) = s''(b), \tag{4.4.26}$$

suffice to determine the spline uniquely. From the first condition it follows that $k_0 = k_m$, which can be used to eliminate k_0 in the equation (4.4.18) for $k = 1$. The second condition in (4.4.26) gives using (4.4.21) $(k_0 + 2k_1 - 3d_1)/h_1 = -(2k_{m-1} + k_m - 3d_m)/h_m$, or after eliminating k_0,

$$2h_m k_1 + 2h_1 k_{m-1} + (h_1 + h_m)k_m = 3(h_m d_1 + h_1 d_m),$$

The spline interpolant has the following best approximation property.

Theorem 4.4.6.

Among all functions g that are twice continuously differentiable on $[a, b]$ and which interpolate f at the points $a = x_0 < x_1 < \cdots < x_m = b$, the natural spline function minimizes

$$\int_a^b \big(s''(t)\big)^2 dt.$$

The same minimum property holds for the complete spline interpolant, if the functions g satisfy $g'(a) = f'(a)$, and $g'(b) = f'(b)$.

Proof. See de Boor [7, 1978, Chapter 5]. □

Due to this property spline functions yield smooth interpolation curves, except for rather thin oscillatory layers near the boundaries if the "natural" boundary conditions $s''(a) = s''(b) = 0$ are far from being satisfied. For the complete or almost complete cubic spline and for cubic splines determined by the "not-a-knot" conditions, these oscillations are much smaller; see Sec. 4.4.4.[17]

Equations (4.4.18) together with any of these boundary conditions give rise to a well-conditioned system of linear equations for determining the derivatives k_i. For the first three boundary conditions the system is **tridiagonal**. As demonstrated in Example 1.3.5 such systems can be solved by Gaussian elimination in $\mathcal{O}(n)$ flops. It can be proved that a sufficient condition for the algorithm given there to be stable is that A is **diagonally dominant**, i.e.

$$|b_1| > |c_1|, \quad |b_k| > |a_{k-1}| + |c_k|, \quad k = 2 : m-1, \quad |b_n| > |a_{m-1}|.$$

It is also stable for the system resulting from the not-a-knot boundary condition although this is not diagonally dominant in the first and last row; see Problem 2b. (Methods for solving general banded linear systems will be studied in more detail in Volume II, Sec. 7.4).

Example 4.4.2. In the case of spline interpolation with constant stepsize $h_i = h$ equation (4.4.18) becomes

$$k_{i-1} + 4k_i + k_{i+1} = 3(d_i + d_{i+1}), \quad i = 1 : m-1. \tag{4.4.27}$$

The "not a knot" boundary conditions (4.4.24)–(4.4.25) become

$$k_0 + 2k_1 = \tfrac{1}{2}(5d_1 + d_2), \qquad 2k_{m-1} + k_m = \tfrac{1}{2}(d_{m-1} + 5d_m). \tag{4.4.28}$$

We obtain a tridiagonal system $Tk = g$, where,

$$\begin{pmatrix} 1 & 2 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 2 & 1 \end{pmatrix} \begin{pmatrix} k_0 \\ k_1 \\ \vdots \\ k_{m-1} \\ k_m \end{pmatrix} = 3 \begin{pmatrix} (5d_1 + d_2)/6 \\ d_1 + d_2 \\ \vdots \\ d_{m-1} + d_m \\ (d_{m-1} + 5d_m)/6 \end{pmatrix}.$$

except for the first and last row, the elements of T are constant along the diagonals. The condition number of T increases very slowly with m; for example, $\kappa(T) < 16$ for $m = 100$.

Consider now the periodic boundary conditions in (iv). Setting $k_m = k_0$ in the last equation we obtain a linear system of equations $Tk = g$ for $k_1, \ldots, k_{m-1}$

[17] When a spline is to be used for the approximate representation of a smooth function, the natural spline is *not* a natural choice.

where

$$T = \left(\begin{array}{ccccc|c} b_1 & c_1 & & & & a_m \\ a_1 & b_2 & c_2 & & & 0 \\ & \ddots & \ddots & \ddots & & \vdots \\ & & a_{m-3} & b_{m-2} & c_{m-2} & 0 \\ & & & a_{m-2} & b_{m-1} & c_{m-1} \\ \hline c_m & 0 & \cdots & 0 & a_{m-1} & b_m \end{array}\right). \tag{4.4.29}$$

Here T is tridiagonal except for its last row and last column, where an extra nonzero element occurs. Such systems, called **arrowhead system**, can be solved with about twice the work of a tridiagonal system; see further Chapter 7.

In some applications one wants to smoothly interpolate given points (x_j, y_j), $j = 0 : m$, where a representation of the form $y = f(x)$ is not suitable. Then we can use a **parametric spline** representation $x = x(\theta)$, $y = y(\theta)$, where the parameter values $0 = \theta_0 \le \theta_1 \le \cdots \le \theta_m$ correspond to the given points. Using the approach described previously two spline functions $s_x(t)$ and $s_y(t)$ can then be determined, that interpolate the points (θ_i, x_i) and (θ_i, y_i), $i = 0 : m$, respectively. The parametrization is usually chosen as $\theta_i = d_i/d$, $i = 1 : m$, where $d_0 = 0$,

$$d_i = d_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}, \quad i = 1 : m.$$

are the the cumulative distance and $d = \sum_{j=1}^{m} d_j$.

For boundary conditions we have the same choices as mentioned previously. In particular, using periodic boundary conditions for $s_x(t)$ and $s_y(t)$ allows the representation of *closed curves* (see Problem 5).

We will now derive estimates of the error in cubic spline interpolation of a function with good smoothness properties, $f \in C^5$ (say). Let $x \in I_i = [x_{i-1}, x_i]$, and set

$$t = (x - x_{i-1})/h_i, \quad y_i = f(x_i), \quad y_i' = f'(x_i).$$

The error can be expressed as the sum of two components:

i. The error $E_H(x)$ due to Hermite interpolation with correct values of $f'(x_{i-1})$, $f'(x_i)$.

ii. The error $E_S(x)$ due to the errors of the slopes $e_i = k_i - y_i'$, $i = 0 : m$.

We shall see that the first part is typically the dominant part. For the error $E_H(x)$ we have from equations (4.4.16)–(4.4.17)

$$\max_{x \in I_i} |E_H(x)| \le \frac{1}{384} \max_{x \in I_i} |h_i^4 f^{(iv)}(x)|, \tag{4.4.30}$$

By (4.4.12) the second part of the error is

$$E_S(x) = h_i t(1-t)\big[e_{i-1}(1-t) - e_i t\big], \quad x = x_{i-1} + t h_i, \quad t \in [0, 1).$$

Since $|1 - t| + |t| = 1$, it follows easily that

$$|E_S(x)| \le \frac{1}{4} \max_{1 \le i \le m} |h_i e_j|, \quad j = i - 1, i. \tag{4.4.31}$$

We shall estimate $|e_j|$ in the case of *constant step size*. Set

$$l_i = 3(d_i + d_{i+1}) - (y'_{i-1} + 4y'_i + y'_{i+1}), \qquad i = 1 : m-1.$$

Then by (4.4.18) $(e_1, \ldots, e_{m-1})$ satisfies

$$\begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_{m-2} \\ e_{m-1} \end{pmatrix} = \begin{pmatrix} l_1 \\ l_2 \\ \vdots \\ l_{m-2} \\ l_{m-1} \end{pmatrix} - \begin{pmatrix} e_0 \\ 0 \\ \vdots \\ 0 \\ e_m \end{pmatrix},$$

or $Ae = l - b$. We write $e = e_I - e_B$, where $Ae_I = l$, $Ae_B = b$. These two systems will be treated differently.

We first estimate e_I and note that, since the matrix A is diagonally dominant, we can use Lemma 6.4.1 to obtain[18]

$$\max_{1\le i<m} |e_{I,i}| \le \frac{1}{\alpha} \max_{1\le i<m} |l_i|, \quad \text{where} \quad \alpha = \min_i \left(|a_{ii}| - \sum_{j\ne i} |a_{ij}| \right) = 2.$$

In order to estimate $\max_{1\le i<m} |l_i|$, note that the defining relation for l_i can be rewritten as

$$\frac{h}{3} l_i = y_{i+1} - y_{i-1} - \frac{h}{3}(y'_{i-1} + 4y'_i + y'_{i+1}).$$

The right hand side here equals the *local error of Simpson's formula* for computing the integral of y' over the interval $[x_{i-1}, x_{i+1}]$, which according to Problem 3.2.12 approximately is $h^5 f^{(5)}(x_i)/90$. It follows that[19]

$$\max_{1\le i<m} |e_{I,i}| \le \frac{1}{60} \max_i h_i^4 |f^{(5)}(x_i)|.$$

By (4.4.31) this shows that the contribution of e_I to E_S is $O(h^5)$ if $f \in C^5$, while $E_H(x)$ is $O(h^4)$. For complete splines $e_0 = e_m = 0$, and for almost complete splines $e_0 = O(h^4)$, $e_m = O(h^4)$. Hence $e_{B,i} = O(h^4)$, and its contribution to E_S is $O(h^5)$. So if h is sufficiently small, *the Hermite interpolation error is, in the whole interval* $[a, b]$*, asymptotically, the dominant source of error for complete and almost complete splines.*[20]

Similar conclusions seem to hold also in the case of variable step size, under the reasonable assumption that $h_{n+1} - h_n = O(h_n^2)$, see Sec. 13.1 (in particular Problem 11), where variable step size is discussed in the context of ordinary differential equations.

Finally we discuss the *effect of the boundary slope errors* for other boundary conditions. The equation $Ae_B = b$ can be written as a difference equation

$$e_{B,i+1} + 4e_{B,i} + e_{B,i-1} = 0, \quad i = 1 : m-1.$$

[18] This is typically an overestimate, almost by a factor of 3, see Problem 3.3.37.

[19] Notice that, if $f \in \mathcal{P}_5$, the slopes k_i becomes exact in complete cubic splines interpolation.

[20] In the literature the usual (rigorous) error bound for a perfect spline, due to Hall and Meyer, is five times as large as the bound for the Hermite error. It is valid with $h = \max h_i$, independent of the position of the knots, for all $f \in C^4$, while we require $f \in C^5$.

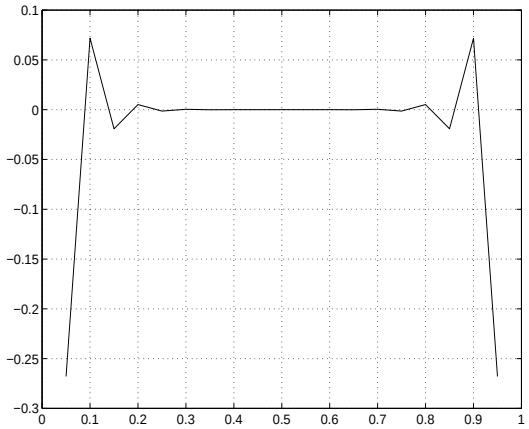


Figure 4.4.7. *Boundary slope errors $e_{B,i}$ for a cubic spline, $e_0 = e_m = -1$; $m = 20$.*

see Sec. 3.4. One can show (Problem 3.4.5) that, for any boundary condition,

$$e_{B,i} \approx u^i e_0 + u^{m-i} e_m, \quad u = \sqrt{3} - 2 \approx -0.268,$$

if u^m is negligible. (Here u and u^{-1} are the roots of the characteristic equation $u^2 + 4u + 1 = 0$.)

Figure 4.4.7 shows (for $m = 20$, $e_0 = e_m = -1$) how rapidly this error component dies out, e.g., $u^4 = 0.005$. At the midpoint $x = 0.5$ the error is $0.3816 \cdot 10^{-5}$.

If $m \gg 1$, $e_0 \neq 0$, and $e_m \neq 0$ it follows that e_B is negligible outside thin *oscillatory boundary layers* near $x = x_0$ and $x = x_m$. The height and thickness of the layers depend on e_0 and e_m. We discuss the left boundary; the right one is analogous. Assume that

$$e_0 = e_{B,0} \neq 0, \qquad e_1 \approx e_{B,1} \approx u e_{B,0} \approx u e_0.$$

We then estimate e_0 by putting

$$k_0 = y_0' + e_0, \qquad k_1 = y_1' + e_1 \approx y_1' + u e_0,$$

into the boundary condition at x_0, i.e. the first equation of (4.4.23) for the natural splines and (4.4.24) for the "not a knot" splines. (Complete splines have no oscillatory boundary layers; $e_B = 0$.) The peak of the contribution of e_B to the spline interpolation error is then obtained by (4.4.12) for $i = 1$, and equals

$$h e_0 \max_{0 \le t \le 1} |t(1 - t(1 - t - ut)| \approx 0.17 h e_0. \tag{4.4.32}$$

For the *natural splines*, this procedure leads to

$$(2 + u)e_0 = 3\frac{1}{h}(y_1 - y_0) - 2y_0' - y_1' - O(h^4)$$

$$= 3\Big(y_0' + \frac{1}{2}hy_0'' + \dots\Big) - 3y_0' - hy_0'' + \dots \sim \frac{1}{2}hy_0''.$$

Since $2 + u = \sqrt{3}$, we obtain $e_0 \approx 0.29hy_0''$, and, by (4.4.32), the peak near $x = x_0$ becomes approximately $0.049h^2|y''|$, i.e. 40% of the *linear* interpolation error (instead of cubic), often clearly visible in a graph of $s(x)$.

For the "not a knot"-splines the procedure leads to

$$(1+2u)e_0 = \frac{5}{2h}(y_1 - y_0) + \frac{1}{2h}(y_2 - y_1) - y_0' - 2y_1' - O(h^4) \approx \frac{h^3}{12}y^{(4)};$$

see Problem 3.2.10. We thus obtain $e_0 \sim 0.180h^3y^{(4)}$, and hence by (4.4.32) the peak near x_0 becomes $0.031h^4y^{(4)}$, typically very much smaller than we found for natural splines. Still it is about 11.5 times as large as the Hermite interpolation error, but since the oscillations die out by the factor $u = 0.29$ in each step, we conclude that *the Hermite interpolation is the dominant error source in cubic "not a knot"-spline interpolation in (say) the interval* $[a + 3h, b - 3h]$.

For natural splines the boundary layers are much thicker, because the peaks are much higher.

Example 4.4.3.

For the function $f(x) = 1/(1 + 25x^2)$, $x \in [-1, 1]$, the maximum norm of the error is 0.022, in interpolation with a natural cubic spline function at the eleven equidistant points $x_i = -1 + 0.2i$, $i = 0 : 10$. This good result contrasts sharply with the unpleasant experience near the boundaries of interpolation with a tenth-degree polynomial shown in Figure 4.2.1. An (almost) perfect cubic spline or a "not a knot"-spline gives even better results near the boundaries.

4.4.5 Computing with B-Splines

It was shown in Sec. 4.4.3 that the set of spline functions of order k, $S_{\Delta,k}$, on the grid

$$\Delta = \{a = x_0 < x_1 < \cdots < x_m = b\}$$

is a linear space of dimension $k + m - 1$. A basis was shown to be

$$\big\{1, x, \dots, x^{k-1}\big\} \cup \big\{(x - x_1)_+^{k-1}, (x - x_2)_+^{k-1}, \dots, (x - x_{m-1})_+^{k-1}\big\}, \qquad (4.4.33)$$

which is the **truncated power basis**.

Example 4.4.4. For $k = 2$ the space $S_{\Delta,k}$ consists of continuous piecewise affine (linear) functions also called linear splines. Then a basis is

$$\{1, x\} \cup \{l_1(x), \dots, l_{m-1}(x)\}, \quad l_i(x) = (x - x_i)_+.$$

Another basis for $S_{\Delta,2}$ is obtained by introducing an artificial exterior knot $x_{-1} \le x_0$. Then it is easy to see that using the functions $l_i(x)$, $i = -1 : m - 1$ every linear

spline on $[x_0, x_m]$ can also be written as

$$s(x) = \sum_{i=-1}^{m-1} c_i l_i(x).$$

The truncated power basis has several disadvantages. The basis functions are not local; e.g., the monomial basis functions $\{1, x, \ldots, x^{k-1}\}$ are nonzero on the whole interval $[a, b]$. Also the basis functions (4.4.33) are almost linearly dependent when the knots are close. Therefore this basis yields an ill-conditioned linear systems for various tasks and is not suited for numerical computations. In the following we will construct a more satisfactory basis for $S_{\Delta,k}$.

In anticipation of the fact that it may be desirable to interpolate at other points than the knots we consider from now on the sequence of knots

$$\Delta = \{\tau_0 \le \tau_1 \le \cdots \le \tau_m\}. \tag{4.4.34}$$

where $\tau_i \le \tau_{i+k}$, $i = 0 : m - k$, i.e. at most k successive knots are allowed to coincide.

We start by considering $k = 1$. The space $S_{\Delta,1}$ consists of piecewise constant functions. As a basis for $S_{\Delta,2}$ we can simply take the functions

$$N_{i,1}(x) = \begin{cases} 1 & x \in [\tau_i, \tau_{i+1}); \\ 0 & \text{otherwise.} \end{cases}, \quad i = 0 : m - 1. \tag{4.4.35}$$

The functions $N_{i,1}(x)$ are arbitrarily chosen to be continuous from the right.

For $k = 2$ we define the **hat functions**[21] by

$$N_{i,2}(x) = \begin{cases} (x - \tau_i)/(\tau_{i+1} - \tau_i), & x \in [\tau_i, \tau_{i+1}], \\ (\tau_{i+2} - x)/(\tau_{i+2} - \tau_{i+1}), & x \in [\tau_{i+1}, \tau_{i+2}), \\ 0, & x \notin (\tau_i, \tau_{i+2}), \end{cases} \quad i = -1 : m - 1. \tag{4.4.36}$$

where we have introduced two **exterior knots** $\tau_{-1} \le \tau_0$ and $\tau_{m+1} \ge \tau_m$ at the boundaries. (In the following we refer to the knots $\tau_0, \ldots, \tau_m$ as **interior knots**.) Note that for $x \in (\tau_i, \tau_{i+1})$ we have $N_{j,2}(x) = 0$, $j \ne i-1, i$. Hence, for a fixed value of x *at most two* hat functions will be nonzero. The exterior knots are usually taken to coincide with the boundary so that $\tau_{-1} = \tau_0$ and $\tau_{m+1} = \tau_m$; see Figure 4.4.8. In this case $N_{-1,1}$ and $N_{m-1,1}$ become "half-hats" with a singularity at τ_0 and τ_m, respectively.

The $(m + 1)$ functions $N_{i,2}(x)$, $i = -1 : m - 1$, are **B-splines** of order two (degree one). At a distinct knot τ_i just one hat function is nonzero, $N_{i+1}(x) = 1$. It follows that the spline function of order $k = 2$ interpolating the points (τ_i, y_i), $i = 0 : m$, can uniquely be written as

$$s(x) = \sum_{i=-1}^{m-1} c_i N_{i,2}(x). \tag{4.4.37}$$

[21] The generalization of hat function to two dimensions is often called tent function. This concept is very important in, e.g., in finite element methods;, see Chap. 14.

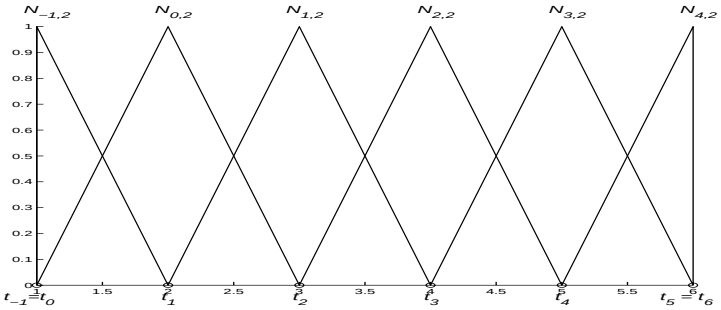


Figure 4.4.8. *The six hat functions $N_{i,2}(x)$, $i = -1 : 4$ $(m + k - 1 = 6)$ with knots $t_{-1} = t_0, t_1, \ldots, t_4, t_5 = t_6$.*

with $c_i = y_{i+1}$. This shows that the restriction of the functions $N_{i,2}(x)$, $i = -1 : m - 1$, to the interval $[\tau_0, \tau_m]$ are $(m + 1)$ linearly independent functions in $S_{\Delta,2}$ and form a basis for $S_{\Delta,2}$.

If we allow two interior knots coalesce, $\tau_i = \tau_{i+1}$, $0 < i < m - 1$, then $N_{i-1,2}(x)$ and $N_{i,2}(x)$ will have a discontinuity at τ_i. This generalizes the concept of a B-spline of order 2 given in Definition 4.4.4 and allows us to model functions with discontinuities at certain knots.

It is easily verified that the functions $N_{i,2}(x)$ can be written as a linear combination of the basis function

$$l_i(x) = (x - \tau_i)_+, \quad i = 1 : m + 1,$$

and it holds that

$$\begin{aligned} N_{i,2}(x) &= \big((x - \tau_{i+2})_+ - (x - \tau_{i+1})_+\big)/(\tau_{i+2} - \tau_{i+1}) \\ &\quad -\big((x - \tau_{i+1})_+ - (x - \tau_i)_+\big)/(\tau_{i+1} - \tau_i) \\ &= \big([\tau_{i+1}, \tau_{i+2}]_t(t - x)_+ - [\tau_i, \tau_{i+1}]_t(t - x)_+ \\ &= (\tau_{i+2} - \tau_i)[\tau_i, \tau_{i+1}, \tau_{i+2}]_t(t - x)_+, \quad i = 1 : m. \end{aligned} \tag{4.4.38}$$

Here $[\tau_i, \tau_{i+1}, \tau_{i+2}]_t$ means the second order divided difference functional[22] operating on a function of t, i.e. the values τ_i etc. are to be substituted for t not for x. Recall that divided differences are defined also for *coincident values* of the argument; see Sec. 4.3.1.

From the definition of the Peano kernel and its basic properties, given in Sec. 3.2.3 it follows that the last expression in (4.4.38) tells us that $N_{i,2}$ is the Peano kernel of a second order divided difference functional multiplied by the constant $\tau_{i+2} - \tau_i$. This observation suggests a definition of B-splines of arbitrary order k and a B-spline basis for the space $S_{\Delta,k}$.

Definition 4.4.7.

[22]The notation is defined in Sec. 4.2.1

Let $\Delta = \{\tau_0 \le \tau_1 \le \cdots \le \tau_m\}$ be an arbitrary sequence of knots such that $\tau_i < \tau_{i+k}$, $i = 0 : m-k$. Then a B-spline of order k equals (apart from a stepsize factor) the Peano kernel of a k-th order divided difference functional; more precisely we define (with the notations used in this chapter)

$$N_{i,k}(x) = (\tau_{i+k} - \tau_i)[\tau_i, \tau_{i+1}, \ldots, \tau_{i+k}]_t (t-x)_+^{k-1}, \tag{4.4.39}$$

where $[\tau_i, \tau_{i+1}, \ldots, \tau_{i+k}]l_x^{k-1}$ denotes the k-th divided difference of the function $l_x^{k-1}(\cdot)$ with respect to the set of points $\tau_i, \tau_{i+1}, \ldots, \tau_{i+k}$.

Since divided differences are defined also for coalescing points (see Sec. 4.4.3), Definition 4.4.7 remains valid for knots that are not distinct.

Example 4.4.5. For $k = 1$ (4.4.38) gives ($\tau_i \ne \tau_{i+1}$)

$$N_{i,1}(x) = (\tau_{i+1} - \tau_i)[\tau_i, \tau_{i+1}]_t (t-x)_+^0.$$

If $\tau_i < x < \tau_{i+1}$, then $(\tau_{i+1} - x)_+^0 = 1$ and $(\tau_i - x)_+^0 = 0$ and hence $N_{i,1} = 1$; otherwise $N_{i,1} = 0$. This coincides with the piecewise constant functions in (4.4.35).

It can be shown that $N_{i,k}(x)$ is defined for all x and is a linear combination of functions $(\tau_j - x)_+^{k-1}$. If the knots are distinct then by Problem 4.2.11,

$$N_{i,k}(x) = (\tau_{i+k} - \tau_i) \sum_{j=i}^{i+k} \frac{(\tau_j - x)_+^{k-1}}{\Phi'_{i,k}(\tau_j)}, \quad \Phi_{i,k}(x) = \prod_{j=i}^{i+k} (x - \tau_j). \tag{4.4.40}$$

This shows that $N_{i,k}$ is a linear combination of functions $(\tau_j - x)_+^{k-1}$, $j = i : i+k$, and thus a spline of order k (as anticipated in the terminology).

The B-spline for equidistant knots is related to the probability density of the sum of k uniformly distributed random variables on $[-\frac{1}{2}, \frac{1}{2}]$. This was known already to Laplace.[23]

Theorem 4.4.8. *The B-splines of order k has the following properties:*

(i) Positivity: $N_{i,k}(x) > 0, \quad x \in (\tau_i, \tau_{i+k})$.

(ii) Compact support: $N_{i,k}(x) = 0, \quad x \notin [\tau_i, \tau_{i+k}]$.

(iii) Summation property: $\sum_i N_{i,k}(x) = 1, \quad \forall x \in [\tau_0, \tau_m]$.

Proof. A proof can be based on the general facts concerning Peano kernels found in Sec. 3.2.3, where also an expression for the B-spline ($k = 3$) is calculated for the equidistant case. (Unfortunately the symbol x means opposite things here and in Sec. 3.2.3.)

[23] Pierre Simon Laplace (1749–1827), French mathematician and astronomer, has also given important contributions to mathematical physics and probability theory.

(i) By (4.2.11) $Rf = [\tau_i, \tau_{i+1}, \ldots, \tau_{i+k}]f = f^{(k)}(\xi)/k!$, $\xi \in (\tau_i, \tau_{i+k})$, and $Rp = 0$, for $p \in \mathcal{P}_k$. It then follows from the corollary of Peano's remainder theorem that the Peano kernel does not change sign in $[\tau_i, \tau_{i+k}]$. It must then have the same sign as $\int K(u)\,du = R(x-a)^k/k! = 1$. This proves a somewhat weaker statement than (i) ($N_{i,k}(x) \geq 0$ instead of $N_{i,k}(x) > 0$).

(ii) This property follows since a Peano kernel always vanishes outside its interval of support of the functional; in this case $[\tau_i, \tau_{i+k}]$. (A more general result concerning the number of zeros is found, e.g., in Powell [38, Theorem 19.1]. Among other things this theorem implies that the jth derivative of a B-spline, $j \leq k-2$, changes sign exactly j times. This explains the "bell-shape" of B-splines.)

(iii) For a sketch of a proof of the summation property [24], see Problem 8. □

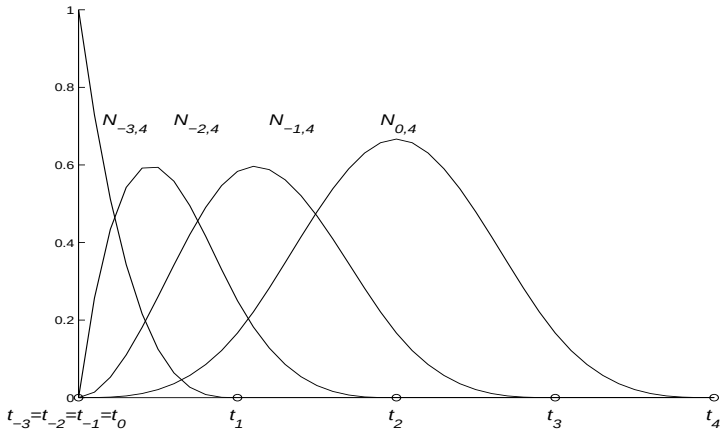


Figure 4.4.9. *The four cubic B-splines nonzero for $x \in (t_0, t_1)$ with coalescing exterior knots $t_{-3} = t_{-2} = t_{-1} = t_0$.*

To get a basis of B-splines for the space $S_{\Delta,k}$, $\Delta = \{\tau_0 \leq \tau_1 \leq \cdots \leq \tau_m\}$, $(m+k-1)$ B-splines of order k are needed. We therefore choose $2(k-1)$ additional knots $\tau_{-k+1} \leq \cdots \leq \tau_{-1} \leq \tau_0$, and $\tau_{m+k-1} \geq \cdots \geq \tau_{m+1} \geq \tau_m$, and B-splines $N_{i,k}(x)$, $i = -k+1 : m-1$.

It is convenient to let the exterior knots coincide with the end points,

$$\tau_{-k+1} = \cdots = \tau_{-1} = \tau_0, \qquad \tau_m = \tau_{m+1} = \cdots = \tau_{m+k-1}.$$

It can be shown that this choice tends to optimize the conditioning of the B-spline basis. Figure 4.4.9 shows the first four cubic B-splines for $k = 4$ (the four last B-splines are a mirror image of these). We note that $N_{-3,4}$ is discontinuous, $N_{-2,4}$ has a non-zero first derivative, and $N_{-2,4}$ a non-zero second derivative at the left boundary.

[24] The B-splines $M_{i,k}$ originally introduced by Curry and Schoenberg were normalized so that $\int_{-\infty}^{\infty} M_{i,k}\,dx = 1$.

Interior knots of multiplicity $r > 1$ are useful when we want to model a function, which has less than $k-2$ continuous derivatives at a particular knot. If $r \leq k$ interior knots coalesce then the spline will only have $k-1-r$ continuous derivatives at this knot.

Lemma 4.4.9. *Let τ_i be a knot of multiplicity $r \leq k$, i.e.*

$$\tau_{i-1} < \tau_i = \cdots = \tau_{i+r-1} < \tau_{i+r}.$$

Then $N_{i,k}$ is at least $(k-r-1)$ times continuously differentiable at τ_i. For $r = k$, the B-spline becomes discontinuous.

Proof. The truncated power $(t-\tau_i)_+^{k-1}$ is $(k-2)$ times continuously differentiable and $[\tau_i, \ldots, \tau_{i+k}]g$ contains at most the $(r-1)$st derivative of g. Hence the lemma follows. □

Consider the spline function

$$s(x) = \sum_{i=-k+1}^{m-1} c_i N_{i,k}(x). \tag{4.4.41}$$

If $s(x) = 0$, $x \in [\tau_0, \tau_m]$, then $s(\tau_0) = s'(\tau_0) = \cdots = s^{(k-1)}(\tau_0) = 0$, and $s(\tau_i) = 0$, $i = 1 : m-1$. From this it can be deduced by induction that in (4.4.41) $c_i = 0$, $i = -k+1 : m-1$. This shows that the $(m+k-1)$ B-splines $N_{i,k}(x)$, $i = -k+1 : m-1$, are linearly independent and form a basis for the space $S_{\Delta,k}$. (A more general result is given in de Boor [7, Theorem IX.1].) Thus any spline function $s(x)$ of order k (degree $k-1$) on Δ can be uniquely written in the form (4.4.41). Note that from the compact support property it follows that for any fixed value of $x \in [\tau_0, \tau_m]$ at most k terms will be nonzero in the sum in (4.4.41), so we have

$$s(x) = \sum_{i=j-k+1}^{j} c_i N_{i,k}(x), \qquad x \in [\tau_j, \tau_{j+1}). \tag{4.4.42}$$

For developing a recurrence relation for B-splines we need the following difference analogue of **Leibniz'**[25] **formula**.

Theorem 4.4.10.
Let $f(x) = g(x)h(x)$, and $x_i \leq x_{i+1} \leq \ldots \leq x_{i+k}$. Then

$$[x_i, \ldots, x_{i+k}]f = \sum_{r=i}^{i+k} [x_i, \ldots, x_r]g \cdot [x_r, \ldots, x_{i+k}]h, \tag{4.4.43}$$

provided that $g(x)$ and $f(x)$ are sufficiently many times differentiable so that the divided differences on the right hand side are defined for any coinciding points x_j.

[25] Gottfried Wilhelm von Leibniz (1646–1716). Leibniz developed his version of calculus at the same time as Newton. Many of the notations he introduced are still used today.

Proof. Note that the product polynomial

$$P(x) = \sum_{r=i}^{i+k} (x - x_i) \cdots (x - x_{r-1})[x_i, \ldots, x_r]g \cdot \sum_{s=i}^{i+k} (x - x_{s+1}) \cdots (x - x_{i+k})[x_s, \ldots, x_{i+k}]h$$

agrees with $f(x)$ at $x_i, \ldots, x_{i+k}$ since by Newton's interpolation formula the first factor agrees with $g(x)$ and the second with $h(x)$ there. If we multiply out we can write $P(x)$ as a sum of two polynomials

$$P(x) = \sum_{r,s=i}^{i+k} \ldots = \sum_{r \le s} \ldots + \sum_{r > s} \ldots = P_1(x) + P_2(x).$$

Since in $P_2(x)$ each term in the sum has $\prod_{j=i}^{i+k}(x - x_j)$ as a factor it follows that $P_1(x)$ will also interpolate $f(x)$ at $x_i, \ldots, x_{i+k}$. The theorem now follows since the leading coefficient of $P_1(x)$, which equals $\sum_{r=i}^{i+k}[x_i, \ldots, x_r]g \cdots [x_r, \ldots, x_{i+k}]h$, must equal the leading coefficient of the unique interpolation polynomial of degree k, which is $[x_i, \ldots, x_{i+k}]f$. □

Theorem 4.4.11. *The B-splines satisfy the recurrence relation*

$$N_{i,k}(x) = \frac{x - \tau_i}{\tau_{i+k-1} - \tau_i} N_{i,k-1}(x) + \frac{\tau_{i+k} - x}{\tau_{i+k} - \tau_{i+1}} N_{i+1,k-1}(x). \tag{4.4.44}$$

Proof. (de Boor [7, pp. 130–131]) The recurrence is derived by applying Leibniz' formula for the k-th divided difference to the product

$$(t - x)_+^{k-1} = (t - x)(t - x)_+^{k-2}.$$

This gives

$$[\tau_i, \ldots, \tau_{i+k}]_t (t - x)_+^{k-1} = (\tau_i - x)[\tau_i, \ldots, \tau_{i+k}]_t (t - x)_+^{k-2} + 1 \cdot [\tau_{i+1}, \ldots, \tau_{i+k}]_t (t - x)_+^{k-2}. \tag{4.4.45}$$

since $[\tau_i]_t(t - x) = (\tau_i - x)$, $[\tau_i, \tau_{i+1}]_t(t - x) = 1$, and $[\tau_i, \ldots, \tau_j]_t(t - x) = 0$ for $j > i + 1$. By the definition of a divided difference

$$(\tau_i - x)[\tau_i, \ldots, \tau_{i+k}]_t = \frac{\tau_i - x}{\tau_{i+k} - \tau_i} \left([\tau_{i+1}, \ldots, \tau_{i+k}]_t - [\tau_i, \ldots, \tau_{i+k-1}]_t \right).$$

Substitute this in (4.4.45), simplify and apply the definition of B-splines. This yields (4.4.44). □

Note that with k multiple knots at the boundaries the denominators in (4.4.44) can become zero. In this case the corresponding nominator also is zero and the term should be set equal to zero.

From Property (ii) in Theorem 4.4.8 we conclude that only k B-splines of order k may be nonzero on a particular interval $[\tau_j, \tau_{j+1}]$. Starting from $N_{i,1}(x) = 1$, $x \in [\tau_i, \tau_{i+1})$ and 0 otherwise, cf. (4.4.35), these B-splines of order k can be *simultaneously* evaluated using this recurrence by forming successively their values for order $1 : k$ in only about $\frac{3}{2}k^2$ flops. This recurrence is extremely stable, since it consists of taking positive (nonnegative) combinations of positive (nonnegative) numbers.

Suppose that $x \in [\tau_i, \tau_{i+1}]$, and $\tau_i \neq \tau_{i+1}$. Then the B-splines of order $k = 1, 2, 3, \ldots$, nonzero at x can be simultaneously evaluated by computing the triangular array

$$\begin{array}{ccccc}
 & & 0 & & \\
 & 0 & & & \cdots \\
 & & N_{i-3,4} & & \\
0 & & N_{i-2,3} & & \cdots \\
 & N_{i-1,2} & & N_{i-2,4} & \\
N_{i,1} & & N_{i-1,3} & & \cdots \\
 & N_{i,2} & & N_{i-1,4} & \\
0 & & N_{i,3} & & \cdots \\
 & 0 & & N_{i,4} & \\
 & & 0 & & \cdots \\
 & & & 0 &
\end{array} \tag{4.4.46}$$

The boundary of zeros in the array is due to the fact that all other B-splines not mentioned explicitly vanish at x. This array can be generated column by column. The first column is known from (4.4.35), and each entry in a subsequent column can be computed as a linear combination with nonnegative coefficients of its two neighbors using (4.4.44). Note that if this is arranged in a suitable order the elements in the new column can overwrite the elements in the old column.

To evaluate $s(x)$, we first determine the index i such that $x \in [\tau_i, \tau_{i+1})$ using, e.g., a linear search or bisection (see Sec. 6.1). The recurrence above is then used to generate the triangular array (4.4.46), which provides $N_{j,k}(x)$, $j = i - k + 1 : i$. in the sum (4.4.42).

Using the B-spline basis we can formulate a more general interpolation problem, where the $n = m + k - 1$ interpolation points, or nodes, x_j do not necessarily coincide with the knots τ_i. We consider determining a spline function $s(x) \in S_{\Delta,k}$, such that

$$s(x_j) = f_j, \quad j = 1 : m + k - 1.$$

Since any spline $s(x) \in S_{\Delta,k}$ can be written as a linear combination of B-splines, the interpolation problem can equivalently be written

$$\sum_{i=-k+1}^{m-1} c_i N_{i,k}(x_j) = f_j, \quad j = 1 : m + k - 1. \tag{4.4.47}$$

These equations form a linear system $Ac = f$ for the coefficients, where

$$a_{ij} = N_{i-k,k}(x_j), \quad i, j = 1 : m + k - 1, \tag{4.4.48}$$

and

$$c = (c_{-k+1}, \ldots, c_{m-1})^T, \qquad f = (f_1, \ldots, f_{m+k-1})^T.$$

The elements $a_{ij} = N_{i-k,k}(x_j)$ of the matrix A can be evaluated by the recurrence (4.4.44). The matrix A will have a banded structure since $a_{ij} = 0$ unless $x_j \in [\tau_i, \tau_{i+k}]$. Hence at most k elements are nonzero in each row of A. (Note that if $x_j = \tau_i$ for some i only $k-1$ elements will be nonzero, which explains why tridiagonal systems were encountered in cubic spline interpolation in earlier sections.)

Schoenberg and Whitney [45, 1953] showed that *the matrix A is nonsingular if and only if its diagonal elements are nonzero*,

$$a_{jj} = N_{j-k,k}(x_j) \neq 0, \quad j = 1 : n,$$

or equivalently if the nodes x_j satisfy

$$\tau_{j-k} < x_j < \tau_j, \quad j = 1 : n. \tag{4.4.49}$$

Further, the matrix can be shown to be **totally nonnegative**, i.e. the determinant of every submatrix is nonnegative. For such systems, if Gaussian elimination is carried out *without pivoting*, the error bound is particularly favorable. This will also preserve the banded structure of A during the elimination.

When the B-spline representation (4.4.41) of the interpolant has been determined it can be evaluated at a given point using the recursion formula (4.4.44). If it has to be evaluated at many points it is more efficient to first convert the spline to its polynomial representation (4.4.14). For hints on how to do that see Problem 9 (b) and (c).

Unless the Schoenberg–Whitney condition (4.4.49) is well-satisfied the system may become ill-conditioned. For splines of even order k the interior nodes

$$\tau_0 = x_0, \quad \tau_{j+1} = x_{j+k/2}, \quad j = 0 : n - k - 1, \quad \tau_m = x_n,$$

is a good choice in this respect. In the important case of cubic splines this means that knots are positioned at each data point except the second and next last (cf. the "not a knot" condition in Sec. 4.4.4.

In some application we are given function values $f_j = f(x_j)$, $j = 1 : n$, that we want to approximate with a spline functions with *much fewer knots* so that $m + k - 1 \leq n$. Then (4.4.47) is an *overdetermined linear system* and the interpolation conditions cannot be satisfied exactly. We therefore consider the linear **least squares spline approximation** problem

$$\min \sum_{j=1}^{n} \left(\sum_{i=-k+1}^{m-1} c_i N_{i,k}(x_j) - f_j \right)^2. \tag{4.4.50}$$

Using the same notation as above this can be written in matrix form

$$\min_c \|Ac - f\|_2^2. \tag{4.4.51}$$

The matrix A will have full column rank equal to $m + k - 1$ if and only if there is a subset of points τ_j satisfying the Schoenberg–Whitney conditions (4.4.49).

If A has full column rank then the least squares solution c is unique and is uniquely determined by the normal equations $A^T Ac = A^T f$. The matrix $A^T A$ is symmetric and positive definite and hence the normal equations can be solved using Cholesky factorization of $A^T A$. A will have at most k nonzero elements in each row and advantage should be taken of the banded form of the matrix $A^T A$; see Figure 4.4.10. More stable methods for solving linear least squares problems (4.4.51) will be introduced in Sec. 8.5.7.

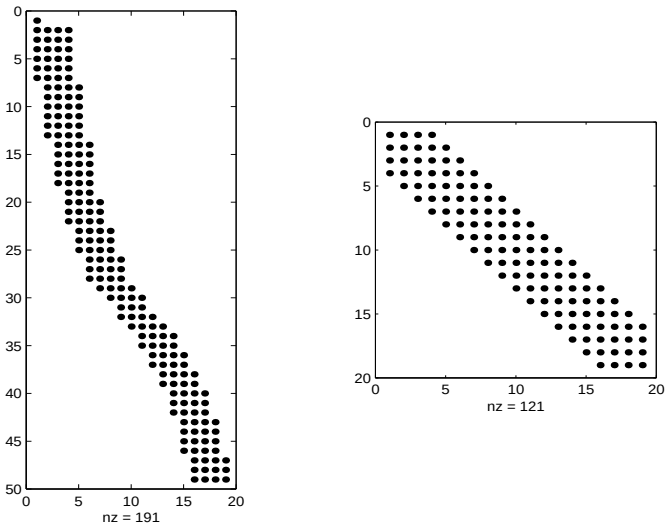


Figure 4.4.10. *Structure of the matrices A and $A^T A$ arising in cubic spline approximation of Titanium data.(nonzero elements showed).*

Example 4.4.6. (de Boor [7]) Consider experimental data describing a property of titanium as a function of temperature. Experimental values for $t_i = 585 + 10i$, $i = 1 : 49$, are given. We want to fit this data using a least squares cubic spline Figure 4.4.11 shows results from using a least squares fitted cubic spline with 9 and 17 knots, respectively. The spline with 9 knots shows oscillations near the points where the curve flattens out and the top of the peak is not well matched. Increasing the number of knots to 17 we get a very good fit.

We have in the treatment above assumed that the set of (interior) knots $\{\tau_0 \leq \tau_1 \leq \cdots \leq \tau_m\}$ is given. In many spline approximation problems it is more realistic to consider the location of knots to be free and try to determine a small set of knots such that the given data can be approximated to a some preassigned accuracy. Several schemes have been developed to treat this problem.

One class of algorithms start with only a few knots and iteratively add more

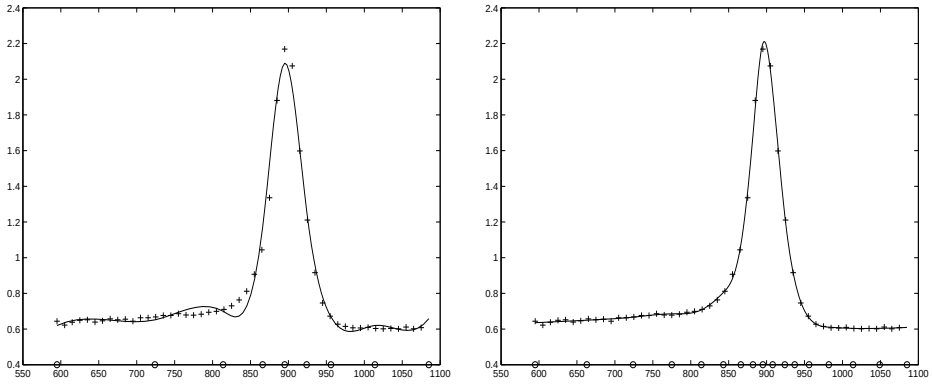


Figure 4.4.11. *Least squares cubic spline approximation of Titanium data; the knots are marked on the axes by a "o"; left: 9 knots;left: 17 knots.*

knots guided by some measure of the error; see de Boor [6, Chapter XII]. The placement of the knots are chosen so that the Schoenberg–Whitney conditions are always satisfied. The iterations are stopped when the approximation is deemed satisfactory. If a node $\tilde{\tau} \in [\tau_j, \tau_{j+1})$ is inserted then the B-spline series with respect to the enlarged set of nodes can cheaply and stably be computed from the old one (see Dierckx [22]).

Other algorithms starts with many knots and successively *remove* knots, which are not contributing much to the quality of the approximation. In these two classes of algorithms one does not seek an optimal knot placement at each step. This is done in a more recent algorithms; see Schwetlick and Schütze [47].

Review Questions

1. What is meant by a cubic spline function? Give an example where such a function is better suited than a polynomial for approximation over the whole interval.

2. (a) What is the dimension of the space $S_{\Delta,k}$ of spline functions of order k on a grid $\Delta = \{x_0, x_1, \ldots, x_m\}$? Give a basis for this space.

(b) Set up the linear system for cubic spline interpolation in the equidistant case for some common boundary conditions. What does the unknown quantities mean, and what conditions are expressed by the equations? About how many operations are required to interpolate a cubic spline function to $m+1$, $m \gg 1$, given values of a function?

3. What error sources have influence on the results of cubic spline interpolation? How fast do the boundary errors die out? How do the results in the interior of the interval depend on the step size (asymptotically)? One of the common types of boundary conditions yield much larger error than the others. Which

one? Compare it quantitatively with one of the others.

4. Approximately how many arithmetic operations are required to evaluate the function values of all cubic B-splines that are nonzero at a given point?

5. Express the restrictions of $f(x) = 1$ and $f(x) = x$ to the interval $[x_0, x_m]$ as linear combinations of the hat functions defined by (4.4.36).

6. The Schoenberg–Whitney conditions give necessary and sufficient conditions for a certain interpolation problem with B-splines of order k. What is the interpolation problem and what are the conditions?

Problems and Computer Exercises

1. Consider a cubic Bézier curve defined by the four control points p_0, p_1, p_2 and p_3. Show that at $t = 1/2$

$$c(1/2) = \frac{1}{4}\frac{p_0 + p_3}{2} + \frac{3}{4}\frac{p_1 + p_2}{2}$$

and interpret this formula geometrically.

2. (G. Eriksson) Approximate the function $y = \cos x$ on $[-\pi/2, \pi/2]$ by a cubic Bézier curve. Determine the four control points in such a way that it interpolates $\cos x$ and its derivative at $-\pi/2, 0$ and $\pi/2$.

Hint Use symmetry and the result of Problem 1 to find the y-coordinate of p_1 and p_2.

3. Suppose that $f(x)$ and the grid Δ are symmetric around the midpoint of the interval $[a, b]$. You can then considerably reduce the amount of computation needed for the construction of the cubic spline interpolant by replacing the boundary condition at $x = b$ by an adequate condition at the midpoint. Which?

(a) Set up the matrix and right hand side for this in the case of constant step size h.

(b) Do the same for a general case of variable step size.

4. (a) Write a program for solving a tridiagonal linear system by Gaussian elimination without pivoting. Assume that the nonzero diagonals are stored in three vectors. Adapt it to cubic spline interpolation with equidistant knots with several types of boundary conditions.

(b) Consider the tridiagonal system resulting from the not-a-knot boundary conditions. Show that after eliminating k_0 between the first two equations and k_m between the last two equations the remaining tridiagonal system for $k_1, \ldots, k_{m-1}$ is diagonally dominant.

(c) Interpolate a cubic spline $s(x)$ through the points $(x_i, f(x_i))$, where

$$f(x) = (1 + 25x^2)^{-1}, \quad x_i = -1 + \frac{2}{10}(i-1), \quad i = 1:11.$$

Compute a natural spline, a complete spline (here $f'(x_1)$ and $f'(x_{11})$ are

needed) and a "not a knot" spline. Compute and compare error curves (natural and logarithmic).

(c) Similar runs as in (b), though for $f(x) = 1/x$, $1 \le x \le 2$, with $h = 0.1$ and $h = 0.05$. Compare the "almost complete", as described in the text, with the complete and the natural boundary condition.

5. If f'' is known at the boundary points, then the boundary conditions can be chosen so that $f'' = s''$ at the boundary points. Show that this leads to the conditions

$$2k_0 + k_1 = 3d_1 - h_1 f''(x_0),$$
$$k_{m-1} + 2k_m = 3d_m + h_m f''(x_m).$$

6. Show that the formula

$$\int_{x_0}^{x_m} s(x)dx = \sum_{i=1}^{m} \Big(\frac{1}{2} h_i (y_{i-1} + y_i) + \frac{1}{12} (k_{i-1} - k_i) h_i^2 \Big),$$

is exact for all cubic spline functions $s(x)$. How does the formula simplify if all $h_i = h$?

Hint: Integrate (4.4.12) from x_{i-1} to x_i.

7. In (4.4.12) the cubic spline $q_i(x)$ on the interval $[x_{i-1}, x_i)$ is expressed in terms of function values y_{i-1}, y_i, and the first derivatives k_{i-1}, k_i.

(a) Show that if $M_i = s''(x_i)$, $i = 0 : m$, are the *second derivatives* (also called **moments**) of the spline function then

$$k_i - d_i = \frac{h_i}{6}(2M_i + M_{i-1}), \qquad k_{i-1} - d_i = -\frac{h_i}{6}(M_i + 2M_{i-1}).$$

Hence $q_i(x)$ can also be uniquely expressed in terms of y_{i-1}, y_i and M_{i-1}, M_i.

(b) Show that, using the parametrization in (a), the continuity of the *first* derivative of the spline function at an interior point x_i gives the equation

$$h_i M_{i-1} + 2(h_i + h_{i+1})M_i + h_{i+1} M_{i+1} = 6(d_{i+1} - d_i).$$

8. (a) Develop an algorithm for solving the arrowhead linear system $Tk = g$ (4.4.29), using Gaussian elimination without pivoting. Show that about twice the number of arithmetic operations are needed compared to a tridiagonal system.

(b) At the end of Sec. 4.4.4 parametric spline interpolation to given points (x_i, y_i), $i = 0 : m$, is briefly mentioned. Work out the details on how to use this to represent a closed curve. Try it out on a boomerang, an elephant, or what have you?

9. (a) Compute and plot a B-spline basis of order $k = 3$ (locally quadratic) and $m = 6$ subintervals of equal length.

Hint: In the equidistant case there is some translation invariance and symmetry, so you do not really need more than essentially three different B-splines.

You need one spline with triple knot at x_0 and a single knot at x_1 (very easy to construct), and two more splines.

(b) Set up a scheme to determine a locally quadratic B-spline, which interpolates given values at the *midpoints* $x_i = (\tau_{i+1}+\tau_i)/2$ ($\tau_{i+1} \neq \tau_i$), $i = 0 : m-1$, and the boundary points τ_0, τ_m. Show that the spline is uniquely determined by these interpolation conditions.

10. Derive the usual formula of Leibniz for the kth derivative from (4.4.43) by a passage to the limit.

11. Use the recurrence (4.4.44)

$$N_{i,k}(x) = \frac{x-\tau_i}{\tau_{i+k-1}-\tau_i} N_{i,k-1}(x) + \frac{\tau_{i+k}-x}{\tau_{i+k}-\tau_{i+1}} N_{i+1,k-1}(x)$$

to show that

$$\sum_i N_{i,k}(x) = \sum_i N_{i,k-1}(x), \quad \tau_0 \le x \le \tau_m,$$

where the sum is taken over all nonzero values. Use this to give an induction proof of the summation property in Theorem 4.4.8.

12. (a) Using the result $\dfrac{d}{dx}(t-x)_+^{k-1} = -(k-1)(t-k)_+^{k-2}$, $k \ge 1$, show the formula for differentiating a B-spline

$$\frac{d}{dx} N_{i,k}(x) = (k-1)\left(\frac{N_{i,k-1}(x)}{\tau_{i+k-1}-\tau_i} - \frac{N_{i+1,k-1}(x)}{\tau_{i+k}-\tau_{i+1}} \right).$$

Then use the relation (4.4.44) to show

$$\frac{d}{dx} \sum_{i=r}^{s} c_i N_{i,k}(x) = (k-1) \sum_{i=r}^{s+1} \frac{c_i - c_{i-1}}{\tau_{i+k-1}-\tau_i} N_{i,k-1}(x),$$

where $c_{r-1} := c_{s+1} := 0$.

(b) Given the B-spline representation of a cubic spline function $s(x)$. Show how to find its polynomial representation (4.4.14) by computing the function values and first derivatives $s(\tau_i), s'(\tau_i)$, $i = 0 : m$.

(c) Apply the idea in (a) recursively to show how to compute all derivatives of $s(x)$ up to order $k-1$. Use this to develop a method for computing the polynomial representation of a spline of arbitrary order k from its B-spline representation.

13. Three different bases for the space of cubic polynomials of degree ≤ 3 on the interval $[0,1]$ are the monomial basis $\{1, t, t^2, t^3\}$, the Bernstein basis $\{B_0^3(t), B_1^3(t), B_2^3(t), B_3^3(t)\}$, and the Hermite basis. Determine the matrices for these basis changes.

4.5 Approximation and Function Spaces

Function space concepts have been introduced successively in this book. Recall, e.g., the discussion of operators and functionals in Sec. 3.2.1, where also the linear space

$\mathcal{P}_n$, the n-dimensional space of polynomials of degree less than n was introduced. This terminology was used and extended in Sec. 4.1, in the discussion of various bases and coordinate transformations in $\mathcal{P}_n$.

For coming applications of functional analysis to interpolation and approximation it is now time for a digression about:

- distances and norms in function spaces;
- a general error bound that we call *the norm and distance formula*;
- inner-product spaces and orthogonal systems.

4.5.1 Distance and Norm

For the study of accuracy and convergence of methods of interpolation and approximation we now introduce the concept of a **metric space**. By this we understand a set of points $\mathcal{S}$, and a real-valued function d, a **distance** defined for pairs of points in $\mathcal{S}$ in such a way that the following axioms are satisfied for all x, y, z in $\mathcal{S}$. (Draw a triangle with vertices at the points x, y, z.)

1. $d(x, x) = 0$, (reflexivity)
2. $d(x, y) > 0$ if $x \neq y$, (posivity)
3. $d(x, y) = d(y, x)$, (symmetry)
4. $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality).

The axioms reflect familiar features of distance concepts used in mathematics and everyday life, such as the absolute value of complex numbers, the shortest distance along a geodesic on the surface of the earth, or the shortest distance along a given road system.[26].

Many other natural and useful relations can be derived from these axioms, e.g.

$$d(x, y) \geq |d(x, z) - d(y, z)|, \qquad d(x_1, x_n) \leq \sum_{i=1}^{n-1} d(x_i, x_{i+1}), \tag{4.5.1}$$

where $x_1, x_2, \ldots, x_n$ is a sequence of points in $\mathcal{S}$; see Problem 1.

Definition 4.5.1.

A sequence of points $\{x_n\}$ in a metric space $\mathcal{S}$ is said to **converge** *to a limit $x^* \in \mathcal{S}$ if $d(x_n, x^*) \to 0$. As $n \to \infty$, we write $x_n \to x^*$ or $\lim_{n\to\infty} x_n = x^*$. A sequence $\{x_n\}$ in $\mathcal{S}$ is called a* **Cauchy sequence**, *if for every $\epsilon > 0$, there is an integer $N(\epsilon)$ such that $d(x_m, x_n) < \epsilon$, for all $m, n \geq N(\epsilon)$. Every convergent sequence is a Cauchy sequence, but the converse is not necessarily true. $\mathcal{S}$ is called a* **complete space** *if every Cauchy sequence in $\mathcal{S}$ converges to a limit in $\mathcal{S}$.*

[26] If $\mathcal{S}$ is a functions space, the points of $\mathcal{S}$ are functions with operands in some other space, e.g., in $\mathbf{R}$ or $\mathbf{R}^n$

It is well known that $\mathbf{R}$ satisfies the characterization of a complete space, but the set of *rational* numbers is not complete. For example, the iteration $x_1 = 1$, $x_{n+1} = \frac{1}{2}(x_n + 2/x_n)$, studied in Example 1.2.1, defines a sequence of rational numbers that converges to $\sqrt{2}$, which is not a rational number.

The distance of a point $x \in \mathcal{S}$ from a subset (subspace) $\mathcal{S}' \subset \mathcal{S}$ is defined by[27]

$$dist\,(x, \mathcal{S}') = \inf_{f \in \mathcal{S}'} d(f, x). \tag{4.5.2}$$

Many important problems in Pure and Applied Mathematics can be formulated as minimization problems. The function space terminology often makes proofs and algorithms less abstract.

Most spaces that we shall encounter in this book are **linear spaces**. Their elements are called vectors, why these spaces also are called **vector spaces** Two operations are defined in these spaces, namely the addition of vectors and the multiplication of a vector by a scalar. They obey the usual rules of algebra.[28] The set of scalars can be either $\mathbf{R}$ or $\mathbf{C}$; the vector space is then called *real* or *complex*, respectively.

We shall be concerned with the problem of **linear approximation**, i.e. a function f is to be approximated using a function f^* that can be expressed as a linear combination

$$f^* = c_1\phi_1(x) + c_2\phi_2(x) + \cdots + c_n\phi_n(x), \tag{4.5.3}$$

of n given linearly independent functions $\phi_1(x), \phi_2(x), \ldots, \phi_n(x)$, where $c_1, c_2, \ldots, c_n$ are parameters to be determined.[29] They may be considered as coordinates of f^* in the functions space spanned by $\phi_1(x), \phi_2(x), \ldots, \phi_n(x)$.

In a vector space the distance of the point f from the origin is called the **norm** of f and denoted by $\|f\|$, typically with some subscript that more or less cryptically indicates the relevant space. The definition of the norm depends namely on the space. The following axioms must be satisfied.

Definition 4.5.2.

A real valued function $\|f\|$ is called a norm on a vector space $\mathcal{S}$, if it satisfies the conditions 1–3 below for all $f, g \in \mathcal{S}$, and for all scalars λ

1. $\|f\| > 0$, *unless* $f = 0$, *(positivity)*
2. $\|\lambda f\| = |\lambda|\|f\|$, *(homogeneity)*
3. $\|f + g\| \leq \|f\| + \|g\|$ *(triangle inequality)*

[27] $\inf x$ denotes the infimum of x, i.e. the greatest lower bound of x. Similarly $\sup x$ is short for supremum, i.e. the least upper bound.

[28] See Appendix A.1 for a summary about vector spaces. In larger texts on linear algebra or functional analysis you find a collection of eight axioms (commutativity, associativity, etc.) required by a linear vector space.

[29] The functions ϕ_j, however, are typically *not* linear. The term "linear interpolation" is from our present point of view rather misleading.

A normed vector space is a metric space; the distance reads

$$d(x, y) = \|x - y\|.$$

If it is also a complete space, it is called a **Banach space**.[30]

The most common norms in spaces of (real and complex) infinite sequences $x = (\xi_1, \xi_2, \xi_3, \ldots)^T$ or spaces of functions on a bounded and closed interval $[a, b]$ and in the spaces $\mathbf{R}^n$ and $\mathbf{C}^n$ are

$$\|x\|_\infty = \max_j |\xi_j|, \qquad \|f\|_\infty = \max_{x\in[a,b]} |f(x)|,$$

$$\|x\|_2 = \Big(\sum_{j=1}^{\infty} |\xi_j|^2\Big)^{1/2}, \qquad \|f\|_2 = \|f\|_{2,[a,b]} = \Big(\int_a^b |f(x)|^2\,dx\Big)^{1/2},$$

$$\|x\|_{2,\omega} = \Big(\sum_{j=1}^{\infty} \omega_j|\xi_j|^2\Big)^{1/2}, \qquad \|f\|_{2,\omega} = \Big(\int_a^b |f(x)|^2\omega(x)\,dx\Big)^{1/2},$$

These norms are called

- the **max(imum) norm** (or the uniform norm);
- the **Euclidean norm** (or the L_2 norm for integrals and l_2 norm for coordinate sequences);
- the **weighted Euclidean norm**. Here $\omega(x)$ is a weight function, assumed to be continuous and strictly positive on the open interval (a, b).

We assume that the integrals

$$\int_a^b |x|^k\omega(x)\,dx$$

exist for all k. Integrable singularities at the end points are permitted; an important example is $\omega(x) = (1 - x^2)^{-1/2}$ on the interval $[-1, 1]$.

The above norms are special cases or limiting cases ($p \to \infty$ gives the max norm) of the l_p or L_p norms and weighted variants of these. They are defined for $p \geq 1$, as follows[31]

$$\|x\|_p = \Big(\sum_{j=1}^{\infty} |\xi_j|^p\Big)^{1/p}, \qquad \|f\|_p = \Big(\int_a^b |f(x)|^p\,dx\Big)^{1/p}. \tag{4.5.4}$$

(The sum in the l_p norm has a finite number of terms, if the space is finite dimensional.)

[30] Stefan Banach (1892–1945) Polish mathematician. professor at the University in Lvov. Banach founded modern Functional Analysis and gave major contributions to the theory of topological vector spaces, measure theory and related topics. In 1939 he was elected President of the Polish Mathematical Society.

[31] The triangle inequality for $\|x\|_p$ is derived from two classical inequalities due to Hölder and Minkowski. Elegant proofs of these are presented, e.g., in Hairer and Wanner [28, p. 327].

From the minimax property of Chebyshev polynomials (Lemma 3.2.3) it follows that the best approximation in the maximum norm to the function $f(x) = x^n$ on $[-1, 1]$ by a polynomial of lower degree is given by $x^n - 2^{1-n}T_n(x)$. The error assumes its extrema in a sequence of $n + 1$ points $x_i = \cos(i\pi/n)$. The sign of the error alternates at these points.

The above property is generalized in the following theorem, which we give without proof. It is the basis of many algorithm for computing approximations in the maximum norm.

Theorem 4.5.3.

Let f be a continuous function on $[a, b]$ and let $\hat{p}$ be the nth degree polynomial which best approximates f in the maximum norm. Then $\hat{p}$ is characterized by the fact that there exists at least $n + 2$ points

$$a \le \zeta_0 < \zeta_1 < \zeta_2 < \cdots < \zeta_{n+1} \le b,$$

where the error $r = \hat{p} - f$ takes on its maximal magnitude with alternating signs; i.e. $|r(\zeta_i)| = \|r\|_\infty$ and

$$r(\zeta_{i+1}) = -r(\zeta_i), \quad i = 0 : n.$$

This characterization constitutes both a necessary and sufficient condition. If $f^{(n+1)}(x)$ has constant sign in $[a, b]$ then $\zeta_0 = a$, $\zeta_{n+1} = b$.

Convergence in a space, equipped with the max norm, means **uniform convergence**. Therefore, *the completeness of the space $C[a, b]$ follows from a classical theorem of Analysis* that tells that the limit of a uniformly convergent sequence is a continuous function. The generalization of this theorem to several dimensions implies the completeness of the space of continuous functions, equipped with the max norm on a closed bounded region in $\mathbf{R}^n$.

Other classes of functions can be normed with the max norm $\max_{x\in[a,b]} |f(x)|$, e.g., $C^1[a, b]$, but this space is not complete; one can subtract a sequence of functions in this space with a limit that is not continuous, but one can often live well with incompleteness.

The notation L_2 norm comes from the function space $L_2[a, b]$, which is the class of functions for which the integral $\int_a^b |f(x)|^2\, dx$ exists, in the sense of Lebesgue;[32] the Lebesgue integral was needed in order to make the space complete. No knowledge of Lebesgue integration is needed for the study of this book, but this particular fact can be interesting as a background. One can apply this norm also to the (smaller) class of continuous functions on $[a, b]$. In this case the Riemannn[33] integral is equivalent. This also yields a normed linear space but *it is not complete*.

[32] Henri Léon Lebesgue (1875–1941) French mathematician. His definition of the Lesbegue integral greatly extended the scope of Fourier analysis.

[33] George Friedrich Bernhard Riemann (1826–1866) German mathematician. He got his Ph. D. 1951 at Göttingen, supervised by Gauss. In his habilitation lecture on Geometry Riemann introduced the curvature tensor and laid the groundwork for Einstein's theory of relativity.

A modification of the L_2 norm that also includes higher derivatives of f is used in the Sobolev spaces, which is a theoretical framework for the study of the practically very important Finite Element Methods (FEM), used in particular for the numerical treatment of partial differential equations; see Vol III. Although $C[0,1]$ is an infinite-dimensional space, the restriction of the continuous functions f to the equidistant grid defined by $x_i = ih$, $h = 1/n$, $i = 0 : n$, constitute an $n+1$ dimensional space, with the function values on the grid as coordinates. If we choose the norm

$$\|f\|_{2,G_h} = \Big(\sum_{i=0}^{1/h} h|f(x_i)|^2\Big)^{1/2},$$

then

$$\lim_{h\to 0} \|f\|_{2,G_h} = \big(\int_0^1 |f(x)|^2\,dx\big)^{1/2} = \|f\|_{2,[0,1]}.$$

Limit processes of this type are common in Numerical Analysis.

Notice that even if $n+1$ functions $\phi_1(x), \phi_1(x), \ldots, \phi_{n+1}(x)$ are linearly independent on the interval $[0,1]$ (say), their restrictions to a grid with n points must be linearly dependent; but if a number of functions are linearly independent on a set $\mathcal{M}$ (a discrete set or continuum), any extensions of these functions to a set $\mathcal{M}' \supset \mathcal{M}$ will also be linearly independent.

The class of functions, analytic in a simply connected domain $\mathcal{D} \subset \mathbf{C}$, normed with $\|f\|_{\mathcal{D}} = \max_{z\in\partial\mathcal{D}} |f(z)|$, is a Banach space denoted by $\mathbf{Hol}(\mathcal{D})$. (The explanation to this term is that analytic functions are also called **holomorphic**.) By the maximum principle for analytic functions $|f(z)| \le \|f\|_{\mathcal{D}}$ for $z \in \mathcal{D}$.

4.5.2 Operator Norms and the Distance Formula

The concepts of **linear operator** and **linear functional** were introduced in Sec. 3.2.2. We here extend to a general vector space $\mathcal{B}$ some definitions for a finite dimensional vector space given in Appendix A.

Next we shall generalize the concept **operator norm** that we have previously used for matrices. Consider an arbitrary bounded linear operator $A : S_1 \mapsto S_2$ in a normed vector space S.

$$\|A\| = \sup_{\|f\|_{S_1}} \|Af\|_{S_2} \tag{4.5.5}$$

Note that $\|A\|$ depends on the vector norm in both S_1 and S_2. It follows that $\|Af\| \le \|A\|\|f\|$. Moreover, whenever the ranges of the operators A_1, A_2 are such that $A_1 + A_2$ and A_1A_2 are defined

$$\|\lambda A\| \le |\lambda|\|A\|, \qquad \|A_1+A_2\| \le \|A_1\|+\|A_2\|, \qquad \|A_1\cdot A_2\| \le \|A_1\|\cdot\|A_2\|. \tag{4.5.6}$$

Similarly for sums with an arbitrary number of terms and for integrals, etc. It follows that $\|A^n\| \le \|A\|^n$, $n = 2, 3, \ldots$.

Example 4.5.1.

Let $f \in C[0,1]$, $\|f\| = \|f\|_\infty$,

$$Af = \sum_{i=1}^{m} a_i f(x_i) \;\Rightarrow\; \|A\| = \sum_{i=1}^{m} |a_i|,$$

$$Af = \int_0^1 e^{-x} f(x)\,dx \;\Rightarrow\; \|A\| = \int_0^1 e^{-x}\,dx = 1 - e^{-1},$$

$$B = \sum_{i=1}^{m} a_i A^i \;\Rightarrow\; \|B\| \le \sum_{i=1}^{m} |a_i| \|A\|^i, \quad (a_i \in \mathbf{C}),$$

$$Kf = \int_0^1 k(x,t) f(t)\,dt \;\Rightarrow\; \|K\|_\infty \le \sup_{x\in[0,1]} \int_0^1 |k(x,t)|\,dt.$$

The proofs of these results are left as a problem. In the last example, approximate the unit square by a uniform grid $(x_i, t_j)_{i,j=1}^{m}$, $h = 1/m$, and approximate the integrals by Riemann sums. Then approximate $\|K\|_\infty$ by the max norm for the matrix with the elements $k_{i,j} = hk(x_i, t_j)$, see Appendix A.8.

The **integral operator** K can thus be considered as an analogue of a matrix. This is a useful point of view also for the numerical treatment of linear integral equations, e.g., equations of the form $f - \lambda K f = g$, g given, see Volume III.

Example 4.5.2.

For the forward difference operator Δ we obtain $\|\Delta\|_\infty = 2$, hence $\|\Delta^k\|_\infty \le 2^k$. In fact $\|\Delta^k\|_\infty = 2^k$, because the upper bound 2^k is attained by the sequence $\{(-1)^n\}_0^\infty$. The same holds for ∇^k, δ^k, and $\mu\delta^k$.

Example 4.5.3.

Let $\mathcal{D}$ be a domain in $\mathbf{C}$, the interior of which contains the closed interval $[a,b]$. Define the mapping D^k: $\mathbf{Hol}\mathcal{D} \Rightarrow C[a,b]$ (with maxnorm), by

$$D^k f(x) = \frac{\partial^k}{\partial x^k} \frac{1}{2\pi i} \int_\partial \mathcal{D} f(z) \frac{1}{(z-x)}\,dz = \frac{1}{2\pi i} \int_\partial \mathcal{D} \frac{k! f(z)}{(z-x)^{k+1}}\,dz.$$

$$sup_{x\in[a,b]} \|D^k f(x)\| \le \max_{z\in\partial\mathcal{D}} |f(z)| \cdot \sup_{x\in[a,b]} \frac{k!}{2\pi} \int_{\partial\mathcal{D}} \frac{|dz|}{|z-x|^{k+1}} < \infty.$$

Note that D^k is in this setting a *bounded* operator, while if we had considered D^k to be a mapping from $C^k[a,b]$ to $C[a,b]$, where both spaces are normed with the max norm in $[a,b]$, D^k would have been an *unbounded* operator.

Many of the procedures for the approximate computation of integrals, derivatives, etc. that encounter in this book, may be characterized as follows. Let A be a linear functional, such that Af cannot be easily computed for an arbitrary function

f, but it can be approximated by another linear functional $\tilde{A}_k$, more easily computable, such that $\tilde{A}_k f = Af \ \forall f \in \mathcal{P}_k$. A general error bound to such procedures was given in Sec. 3.2.3 by the Peano Remainder Theorem, in terms of an integral,

$$\int f^{(k)}(u)K(u)\,du$$

where the Peano kernel $K(u)$ is determined by the functional $R = \tilde{A} - A$.

Now we shall give a different type of error bound for more general approximation problems, where other classes of functions and operators may be involved. Furthermore, no estimate of $f^{(k)}(u)$ is required. It is based on the following almost trivial theorem. It yields, however, often less sharp bounds than the Peano formula, in situations when the latter can be applied.

Theorem 4.5.4. *The Norm and Distance Formula*

Let A, $\tilde{A}$ be a two linear operators bounded in a Banach space $\mathcal{B}$, such that for any vector s in a certain linear subspace S, $\tilde{A}s = As$. Then

$$\|\tilde{A}f - Af\| \le \|\tilde{A} - A\|\,\mathrm{dist}(f, \mathcal{S}) \quad \forall f \in \mathcal{B}.$$

Proof. Set $R = \tilde{A} - A$. For any positive ϵ, there exists a vector $s_\epsilon \in \mathcal{S}$ such that

$$\|f - s_\epsilon\| = \mathrm{dist}(f, s_\epsilon) < \inf_{s\in\mathcal{S})} \mathrm{dist}(f, s) + \epsilon = \mathrm{dist}(f, \mathcal{S}) + \epsilon.$$

Then $\|Rf\| = \|Rf - Rs_\epsilon\| = \|R(f - s_\epsilon)\| \le \|R\|\|f - s_\epsilon\| < (\mathrm{dist}(f, \mathcal{S}) + \epsilon)\|R\|$. The theorem follows, since this holds for every $\epsilon > 0$. □

The following is a common particular case of the theorem. *If A, A_k are* **linear functionals** *such that $A_k p = Ap \ \forall p \in \mathcal{P}_k$, then*

$$|A_k f - Af| \le (\|A_k\| + \|A\|)\mathrm{dist}(f, \mathcal{P}_k). \tag{4.5.7}$$

Another important particular case of the theorem concerns **projections** P from a function space to a finite dimensional subspace, $\mathcal{S}_k$, e.g., *interpolation and series truncation operators*, $A = I$, $\tilde{A} = P$, see the beginning of Sec. 4.5.2. Then

$$\|Pf - f\| \le \|(P - I)f\| \le (\|P\| + 1)\mathrm{dist}(f, \mathcal{S}_k). \tag{4.5.8}$$

The Norm and Distance Formula requires bounds for $\|\tilde{A}\|$, $\|A\|$ and $\mathrm{dist}(f, \mathcal{S})$. We have seen examples above, how to obtain bounds for operator norms. Now we shall exemplify how to obtain bounds for the distance of f from some relevant subspace $\mathcal{S}$, in particular spaces of polynomials or trigonometric polynomials restricted to some real interval $[a, b]$. For the efficient estimation of $\mathrm{dist}(f, \mathcal{S})$ it may be important, e.g., to take into account that f is analytic in a larger domain than $[a, b]$.

Theorem 4.5.5. *Estimation of* $\operatorname{dist}_\infty(f, \mathcal{P}_k)$ *in terms of* $\|f^{(k)}\|_\infty$.

Let $f \in C^k[a,b] \subset \mathbf{R}$, $\|g\| = \max_{t\in[a,b]} |g(t)|$, $\forall g \in C[a,b]$. *Then*

$$\operatorname{dist}_{\infty,[a,b]}(f, \mathcal{P}_k) \le \frac{2\|f^{(k)}\|_{\infty,[a,b]}}{k!}\Big(\frac{b-a}{4}\Big)^k.$$

Proof. Let $p(t)$ be the polynomial which interpolates $f(t)$ at the points $t_j \in [a,b]$, $j = 1:k$. By the remainder term in interpolation, (4.2.6),

$$|f(t) - p(t)| \le \max_{\xi\in[a,b]} \frac{|f^{(k)}(\xi)|}{k!} \prod_{j=1}^{k} |t - t_j|.$$

Set $t = \frac{1}{2}(b+a) + \frac{1}{2}(b-a)x$, and choose $t_j = \frac{1}{2}(b+a) + \frac{1}{2}(b-a)x_j$ where x_j are the zeros of the Chebyshev polynomial $T_k(x)$, i.e. p is the Chebyshev interpolation polynomial for f on the interval $[a,b]$. Set $M = \|f^{(k)}\|/k!$. Then

$$|f(t) - p(t)| \le M \prod_{j=1}^{k} \frac{(b-a)|x - x_j|}{2}, \quad x \in [-1,1].$$

Since the leading coefficient of $T_k(x)$ is 2^{k-1}, and $|T_k(x)| \le 1$, we have, for $t \in [a,b]$,

$$|f(t) - p(t)| \le M\left(\frac{b-a}{2}\right)^k \frac{1}{2^{k-1}}|T_k(x)| \le M\left(\frac{b-a}{2}\right)^k \frac{1}{2^{k-1}},$$

The bound stated for $\operatorname{dist}_\infty(f, \mathcal{P}_k)$ is thus satisfied. □

Example 4.5.4.

By the above theorem, the function e^t can be approximated on the interval $[0,1]$ by a polynomial in $\mathcal{P}_6$ with the error bound $2e\cdot 4^{-6}/6! \approx 2\cdot 10^{-6}$. According to the proof this accuracy is attained by Chebyshev interpolation on $[0,1]$.

If one instead uses the Maclaurin series, truncated to $\mathcal{P}_6$, then the remainder is $e^\theta/(6!) \ge 1.3\cdot 10^{-3}$. Similarly, with the truncated Taylor series about $t = \frac{1}{2}$ the remainder is $e^\theta/(2^6 6!) \ge 2\cdot 10^{-5}$, still significantly less accurate than the Chebyshev interpolation. Economization of power series (see Problem 3.2.4), yields approximately the same accuracy as Chebyshev interpolation.

If we do these things on an interval of length h (instead of the interval $[0,1]$) all the bounds are to multiplied by h^6.

Example 4.5.5. *The use of analyticity in estimates for* $\operatorname{dist}_\infty(f, \mathcal{P}_k)$ *etc.*

Denote by $\mathcal{E}_R$ an ellipse in $\mathbf{C}$ with foci at -1 and 1; R is equal to the sum of the semi-axes. Theorem 3.2.3 gives the following truncation error bound for the Chebyshev expansion for a function $f \in \mathbf{Hol}(\mathcal{E}_R)$ and real-valued on $[-1,1]$.

$$\Big|f(x) - \sum_{j=0}^{k-1} c_j T_j(x)\Big| \le \frac{2\|f\|_{\mathcal{E}_R} R^{-k}}{1 - R^{-1}}, \quad x \in [-1,1].$$

This implies that, on the same assumptions concerning f,

$$\operatorname{dist}_{\infty,[-1,1]}(f,\mathcal{P}_k) \le \frac{2\|f\|_{\mathcal{E}_R} R^{-k}}{1-R^{-1}}.$$

Suppose that $f \in \mathbf{Hol}(\mathcal{D})$, where $\mathcal{D} \supseteq \dfrac{b+a}{2} + \dfrac{b-a}{2}\mathcal{E}_R$. Then transforming from $[-1,1]$ to a general interval $[a,b] \subset \mathbf{R}$, we have

$$\operatorname{dist}_{\infty,[a,b]}(f,\mathcal{P}_k) \le 2\|f\|_{\mathcal{D}} \left(\frac{b-a}{2R}\right)^k \frac{2R}{2R-(b-a)}.$$

Example 4.5.6.

As a first simple example we shall derive an error bound for one step with the trapezoidal rule. Set

$$Af = \int_0^h f(x)\,dx, \qquad A_2 f = \frac{h}{2}\big(f(0)+f(h)\big).$$

We know that $A_2 p = Ap$ if $p \in \mathcal{P}_2$. By Theorem 4.5.5, $\mathrm{d}ist_{\,\infty}(f,\mathcal{P}_2) \le \|f''\|_\infty h^2/16$.

Furthermore, $\|A\|_\infty = \int_0^h dx = h$, $\|A_2\|_\infty = h$, hence by (4.5.7). the requested error bound becomes

$$\|A_2 f - Af\|_\infty \le 2h \cdot \|f''\|_\infty h^2/16 = \|f''\|_\infty h^3/8.$$

This general method does not always give the best possible bounds but, typically, it gives no gross overestimate. For the trapezoidal rule we know by Peano's method (Example 3.4.7) that $\|f''\|h^3/12$ is the best possible estimate, so we now obtained a 50% overestimate of the error.

The norm and distance formula can also be written in the form

$$\operatorname{dist}(f,\mathcal{S}) \ge |Af - \tilde{A}f|/\|A-\tilde{A}\|.$$

This can be used for finding a simple lower bound for $\mathrm{d}ist\,(f,\mathcal{P}_k)$ in terms of an easily computed functional that vanishes on $\mathcal{P}_k$.

Example 4.5.7.

Let $\tilde{A}=0$. The functional $Af = f(1)-2f(0)+f(-1)$ vanishes for $f \in \mathcal{P}_2$. If the maximum norm is used on [-1,1], then $\|A\| = 1+2+1 = 4$. Thus

$$\operatorname{dist}(f,\mathcal{P}_2)_{\infty,[-1,1]} \ge \frac{|Af|}{\|A\|} = \frac{1}{4}|f(1)-2f(0)+f(-1)|.$$

It follows, e.g., that the curve $y = e^x$ cannot be approximated by a straight line in $[-1,1]$ with an error less than $(e-2+e^{-1})/4 \approx 0.271$. (This can also be seen without the use of the Norm and Distance Formula.)

It is harder to derive the following generalization without the Norm and Distance Formula. By Example 4.5.2, $\|\Delta^k\| = 2^k$, $\Delta^k p = 0$ if $p \in \mathcal{P}_k$, hence

$$\operatorname{dist}(f, \mathcal{P}_k)_{\infty,[x_0,x_k]} \geq 2^{-k}|\Delta^k f(x_0)|. \tag{4.5.9}$$

There is another inequality that is usually sharper but less convenient to use. (It follows from the discrete orthogonality property of the Chebyshev polynomials, see Sec. 4.6.)

$$\operatorname{dist}(f, \mathcal{P}_k)_{\infty,[x_0,x_k]} \geq \frac{1}{k}\left|\sum_{j=0}^{k}(-1)^j a_j f\left(\cos\frac{j\pi}{k}\right)\right|, \tag{4.5.10}$$

where $a_j = \frac{1}{2}$, $j = 0, k$ and $a_j = 1$ otherwise. Inequalities of this type can reveal when one had better using *piecewise polynomial approximation* of a function on an interval instead of using a single polynomial over the whole interval. See also Sec. 4.4.

One of the fundamental theorems in approximations theory is Weierstrass'[34] approximation theorem.

Theorem 4.5.6. Weierstrass' Approximation Theorem

For every continuous function f defined on a closed, bounded interval $[a, b]$ it holds that

$$\lim_{n\to\infty} \operatorname{dist}(f, \mathcal{P}_n)_{\infty,[a,b]} = 0.$$

Proof. For an elegant proof due to S. Bernstein using Bernstein polynomials; see Davis [19, Sec. 6.2]. □

The smoother f is, the quicker $\operatorname{dist}(f, P_n)$ decreases, and the narrower the interval is, the less $\operatorname{dist}(f, P_n)$ becomes. In many cases $\operatorname{dist}(f, P_n)$ decreases so slowly toward zero (as n grows) that it is impractical to attempt to approximate f with only one polynomial in the entire interval $[a, b]$.

In infinite-dimensional spaces, certain operators may not be defined everywhere, but only in a set that is everywhere dense in the space. For example, in the space $C[a, b]$ of continuous functions on a bounded interval (with the maximum norm), the operator $A = d/dx$ is not defined everywhere, since there are continuous functions, which are not differentiable. By Weierstrass' Approximation Theorem any continuous function can be approximated uniformly to arbitrary accuracy by a polynomial. In other words: the set of polynomials is everywhere dense in C, and hence the set of differentiable functions is so too. Moreover, even if Au exists, A^2u may not exist. That A^{-1} may not exist, is no novel feature of infinite-dimensional spaces. In $C[a, b]$ the norm of $A = d/dx$ is infinite. This operator is said to be unbounded.

[34] Kurt Theodor Wilhelm Weierstrass (1815–1897) German mathematician, whose lectures at Berlin University attracted students from all over the world. He set high standards of rigor in his work and is known as the father of modern analysis.

4.5.3 Inner Product Spaces and Orthogonal Systems

An abstract foundation for least squares approximation is furnished by the theory of **inner product spaces**, which we now introduce.

Definition 4.5.7.

A normed linear space $\mathcal{S}$ will be called an inner product space, if for each two elements f, g in $\mathcal{S}$ there is a scalar designated by (f, g) with the following properties

1.	$(f+g,h) = (f,h) + (g,h)$	*(linearity)*
2.	$(f,g) = \overline{(g,f)}$	*(symmetry)*
3.	$(\alpha f, g) = \alpha(f,g), \quad \alpha$ *scalar*	*(homogeneity)*
4.	$(f,f) \geq 0, \quad (f,f) = 0,$ *iff* $f = 0$	*(positivity)*

The **inner product** (f, g) *is scalar, i.e. real in a real space and complex in a complex space. The* **norm** *is defined as*

$$\|f\| = (f,f)^{1/2}.$$

We shall show below that the triangle inequality is satisfied. (The other axioms for a norm are obvious.) The standard inner products introduced in § 1.6.2, are particular cases, if we set $(x, y) = y^T x$ in $\mathbf{R}^n$, and $(x, y) = y^H x$ in $\mathbf{C}^n$. A **complete** inner-product space is called a **Hilbert space** and is often denoted $\mathcal{H}$ in this book.

One can make computations using the more general definition of (f, g) given above in the same way that one does with scalar products in linear algebra. Note, however, the *conjugations necessary in a complex space*, e.g.,

$$(f, \alpha g) = \bar{\alpha}(f, g), \tag{4.5.11}$$

because, by the axioms, $(f, \alpha g) = \overline{(\alpha g, f)} = \overline{\alpha(g, f)} = \bar{\alpha}\overline{(g, f)} = \bar{\alpha}(f, g)$. By the axioms it follows by induction that

$$\Big(\phi_k, \sum_{j=1}^{n} c_j \phi_j\Big) = \sum_{j=1}^{n} (\phi_k, c_j \phi_j) = \sum_{j=1}^{n} \bar{c}_j (\phi_k, \phi_j). \tag{4.5.12}$$

Theorem 4.5.8.

The Cauchy–Schwarz inequality in a complex space

$$|(f,g)| \leq \|f\| \, \|g\|.$$

Proof. Let f, g be two arbitrary elements in an inner-product space. Then[35], for every *real* number λ,

$$0 \leq (f + \lambda(f,g)g, f + \lambda(f,g)g) = (f,f) + 2\lambda|(f,g)|^2 + \lambda^2 |(f,g)|^2 (g,g).$$

[35] We found this proof in [41, n°83]. The application of the same idea in a *real* space can be made simpler, see Problem X.

This polynomial in λ with real coefficients cannot have two distinct zeros, hence the discriminant cannot be positive, i.e.

$$|(f,g)|^4 - (f,f)|(f,g)|^2(g,g) \le 0.$$

So, even if $(f,g) = 0$), $|(f,g)|^2 \le (f,f)(g,g)$. □

By the definition and the Cauchy–Schwarz inequality,

$$\begin{aligned}\|f+g\|^2 &= (f+g, f+g) = (f,f) + (f,g) + (g,f) + (g,g)\\ &\le \|f\|^2 + \|f\|\,\|g\| + \|g\|\,\|f\| + \|g\|^2 = (\|f\| + \|g\|)^2.\end{aligned}$$

This shows that the **triangle inequality** is satisfied by the norm defined above.

Example 4.5.8.
The set of all complex infinite sequences $\{x_i\}$ for which $\sum_{i=1}^{\infty} |x_i|^2 < \infty$ and equipped with the inner product

$$(x,y) = \sum_{i=1}^{\infty} x_i \overline{y}_i,$$

constitutes a Hilbert space.

Definition 4.5.9.
Two functions f and g are said to be **orthogonal** *if $(f,g) = 0$. A finite or infinite sequence of functions $\phi_0, \phi_1, \ldots, \phi_n$ constitutes an* **orthogonal system**, *if*

$$(\phi_i, \phi_j) = 0, \quad i \ne j, \text{ and } \|\phi_i\| \ne 0, \quad \forall i. \tag{4.5.13}$$

If, in addition, $\|\phi_i\| = 1 \forall i$, then the sequence is called an **orthonormal system**.

Theorem 4.5.10 (Pythagoras' theorem).
Let $\{\phi_1, \phi_2, \ldots, \phi_n\}$ be an orthogonal system in an inner-product space. Then

$$\Big\| \sum_{j=1}^{n} c_j \phi_j \Big\|^2 = \sum_{j=1}^{n} |c_j|^2 \|\phi_j\|^2.$$

The elements of an orthogonal system are linearly independent.

Proof. We start as in the proof of the triangle inequality:

$$\|f+g\|^2 = (f,f) + (f,g) + (g,f) + (g,g) = (f,f) + (g,g) = \|f\|^2 + \|g\|^2.$$

Using this result and induction the first stement follwos. The second statement then follows because $\sum c_j \phi_j = 0 \;\Rightarrow\; |c_j| = 0, \forall j$. □

Theorem 4.5.11.

A linear operator P is called a **projection** *(or projector) if $P = P^2$. Let $\mathcal{V}$ be the range of the operator P. Then P is a projection if and only $Pv = v$ if for each $v \in \mathcal{V}$.*

Proof. If P is a projection, then $v = Px$ for some $x \in \mathcal{B}$, hence $Pv = P^2x = Px = v$. Conversely, if Q is a linear operator, such that $Qx \in \mathcal{V}$, $\forall x \in \mathcal{B}$, and $v = Qv$, $\forall v \in V$, then Q is a projection, in fact $Q = P$. □

Note that $I - P$ is also a projection, because

$$(I - P)(I - P) = I - 2P + P^2 = I - P.$$

Every vector $x \in \mathcal{B}$ can be written *uniquely* in the form $x = u + w$, where $u = Px$, $w = (I - P)x$, so that $u \in range\,(P)$, $w \in range\,(I - P)$.

Important examples of projections in function spaces are interpolation operators, e.g., the mapping of $C[a, b]$ into $\mathcal{P}_k$ by Newton or Lagrange interpolation, because each polynomial is mapped to itself. The two types of interpolation are the same projection, although they use different bases in $\mathcal{P}_k$. Another example is the mapping of a linear space of functions, *analytic* on the unit circle, into $\mathcal{P}_k$ so that each function is mapped to its *Maclaurin expansion truncated to* $\mathcal{P}_k$. There are analogous projections where, e.g., periodic functions and trigonometric polynomials are involved, or functions in C^3 and cubic splines.

In an inner product space, the **adjoint operator** A^* of a linear operator A is defined by the requirement that

$$(A^*u, v) = (u, Av), \ \forall\ u, v. \tag{4.5.14}$$

An operator A is called **self-adjoint** if $A = A^*$. In $\mathbf{R}^n$, we define $(u, v) = uv^T$, i.e. the standard scalar product. Then $(A^*u)^Tv = u^TAv$, i.e. $u^T((A^*)^Tv = u^TAv$ hence $(A^*)^T = A$. It follows that symmetric matrices are self-adjoint. Similarly, in $\mathbf{C}^n$, we define $(u, v) = uv^H$. It follows that $A^* = A^H$ and that Hermitean matrices are self-adjoint. An operator B is **positive definite** if $(u, Bu) > 0\ \forall\ u \neq 0$.

Example 4.5.9.

An important example of an orthogonal system is the sequence of trigonometric functions $\phi_j(x) = \cos jx$, $j = 0 : m$. These form an orthogonal system, with the either of the two inner products

$$(f, g) = \int_0^\pi f(x)g(x)\,dx \quad \text{(continuous case, } m = \infty\text{)},$$

$$(f, g) = \sum_{i=0}^{m} f(x_\mu)g(x_\mu), \quad x_\mu = \frac{2\mu + 1}{m + 1}\frac{\pi}{2} \quad \text{(discrete case)}.$$

Moreover, it holds that

$$\|\phi_j\|^2 = \frac{1}{2}\pi, \quad j > 0, \quad \|\phi_0\|^2 = \pi, \quad \text{(continuous case)},$$

$$\|\phi_j\|^2 = \frac{1}{2}(m+1), \quad 1 \le j \le m, \quad \|\phi_0\|^2 = m+1. \quad \text{(discrete case)}.$$

These results are closely related to the orthogonality of the Chebyshev polynomials; see Theorem 4.5.17.

Trigonometric interpolation and Fourier analysis will be treated in Sec. 4.6.

There are many other examples of orthogonal systems. Orthogonal systems of polynomials play an important role in approximation and numerical integration. Orthogonal systems also occur in a natural way in connection with eigenvalue problems for differential equations, which are quite common in mathematical physics.

4.5.4 Solution of the Approximation Problem

Orthogonal systems give rise to extraordinary formal simplifications in many situations. We now consider the least squares approximation problem of minimizing the norm of the error function $\|f^* - f\|$ over all functions $f^* = \sum_{j=0}^{n} c_j \phi_j$.

Theorem 4.5.12.

If $\phi_0, \phi_1, \ldots, \phi_n$ are linearly independent, then the least squares approximation problem has a unique solution,

$$f^* = \sum_{j=0}^{n} c_j^* \phi_j, \tag{4.5.15}$$

which is characterized by the orthogonality property that $f^* - f$ is orthogonal to all ϕ_j, $j = 0 : n$. The coefficients c_j^*, which are called **orthogonal coefficients** *(or Fourier coefficients), satisfy the linear system of equations*

$$\sum_{j=0}^{n} (\phi_j, \phi_k) c_j^* = (f, \phi_k). \tag{4.5.16}$$

called normal equations. In the important special case when $\phi_0, \phi_1, \ldots, \phi_n$ form an orthogonal system, the coefficients are computed more simply by

$$c_j^* = (f, \phi_j)/(\phi_j, \phi_j), \quad j = 0 : n. \tag{4.5.17}$$

Proof. Let $(c_0, c_1, \ldots, c_n)$ be a sequence of coefficients with $c_j \neq c_j^*$ for at least one j. Then

$$\sum_{j=0}^{n} c_j \phi_j - f = \sum_{j=0}^{n} (c_j - c_j^*)\phi_j + (f^* - f).$$

If $f^* - f$ is orthogonal to all the ϕ_j, then it is also orthogonal to the linear combination $\sum_{j=0}^{n} (c_j - c_j^*)\phi_j$, and according to the Pythagorean Theorem

$$\Big\| \sum_{j=0}^{n} c_j \phi_j - f \Big\|^2 = \Big\| \sum_{j=0}^{n} (c_j - c_j^*)\phi_j \Big\|^2 + \|(f^* - f)\|^2 > \|(f^* - f)\|^2.$$

Thus if $f^* - f$ is orthogonal to all ϕ_k, then f^* is a solution to the approximation problem. It remains to show that the orthogonality conditions

$$\Big(\sum_{j=0}^{n} c_j^*\phi_j - f, \phi_k\Big) = 0, \quad k = 0:n,$$

can be fulfilled. The above conditions are equivalent to the normal equations in (4.5.16). If $\{\phi_j\}_{j=0}^n$ constitutes an orthogonal system, then the system can be solved immediately, since in each equation all the terms with $j \neq k$ are zero. The formula in (4.5.17) then follows immediately.

Suppose now that we know only that $\{\phi_j\}_{j=0}^n$ are linearly independent. The solution to the normal equations exists and is unique, unless the homogeneous system,

$$\sum_{j=0}^{n}(\phi_j, \phi_k)c_j^* = 0, \quad k = 0:n$$

has a solution $c_0, c_1, \ldots, c_n$, with at least one $c_i \neq 0$. But this would imply

$$\Big\|\sum_{j=0}^{n} c_j\phi_j\Big\|^2 = \Big(\sum_{j=0}^{n} c_j\phi_j, \sum_{k=0}^{n} c_k\phi_k\Big) = \sum_{k=0}^{n}\sum_{j=0}^{n}(\phi_j, \phi_k)c_jc_k = \sum_{k=0}^{n} 0 \cdot c_k = 0,$$

which contradicts that the ϕ_j were linearly independent. □

In the case where $\{\phi_j\}_{j=0}^n$ form an orthogonal system, *the Fourier coefficients c_j^* are independent of n* (see formula (4.5.17)). This has the important advantage that one can increase the total number of parameters without recalculating any previous ones. Orthogonal systems are advantageous not only because they greatly simplify calculations; using them, one can often avoid numerical difficulties with round-off error which may occur when one solves the normal equations for a nonorthogonal set of basis functions.

With every continuous function f one can associate an infinite series,

$$f \sim \sum_{j=0}^{\infty} c_j^*\phi_j, \quad c_j^* = \frac{(f, \phi_j)}{(\phi_j, \phi_j)}.$$

Such a series is called an **orthogonal expansion**. For certain orthogonal systems this series converges with very mild restrictions on the function f.

Theorem 4.5.13.

If f^* is defined by formulas (4.5.15) *and* (4.5.17)*, then*

$$\|f^* - f\|^2 = \|f\|^2 - \|f^*\|^2 = \|f\|^2 - \sum_{j=0}^{n}(c_j^*)^2\|\phi_j\|^2.$$

Proof. Since $f^* - f$ is, according to Theorem 4.5.12, orthogonal to all ϕ_j, $0 \leq j \leq n$, then $f^* - f$ is orthogonal to f^*. The theorem then follows directly from the Pythagorean Theorem and Theorem 4.5.10. □

We obtain as corollary **Bessel's inequality**:

$$\sum_{j=0}^{n}(c_j^*)^2\|\phi_j\|^2 \le \|f\|^2. \tag{4.5.18}$$

The series $\sum_{j=0}^{\infty}(c_j^*)^2\|\phi_j\|^2$ is convergent. If $\|f^* - f\| \to 0$ as $n \to \infty$, then the sum of the latter series is equal to $\|f\|^2$, which is **Parseval's identity**.

Theorem 4.5.14.

If $\{\phi_j\}_{j=0}^m$ are linearly independent on the grid $\{x_i\}_{j=0}^m$, then the interpolation problem of determining the coefficients $\{c_i\}_{j=0}^m$ such that

$$\sum_{j=0}^{m} c_j\phi_j(x_i) = f(x_i), \quad i = 0:m, \tag{4.5.19}$$

has exactly one solution. Interpolation is a special case ($n = m$) of the method of least squares. If $\{\phi_j\}_{j=0}^m$ is an orthogonal system, then the coefficients c_j are equal to the orthogonal coefficients in (4.5.17).

Proof. The system of equations (4.5.19) has a unique solution, since its column vectors are the vectors $\phi_j(G)$, $j = 0:n$, which are linearly independent. For the solution of the interpolation problem it holds that $\|c_j\phi_j - f\| = 0$; that is, the error function has the least possible semi-norm. The remainder of the theorem follows from Theorem 4.5.12. □

The following collection of important and equivalent properties is named the **Fundamental theorem of orthonormal expansions**, by Davis [19, Theorem 8.9.1], whom we follow closely at this point.

Theorem 4.5.15.

Let $\phi_1, \phi_2, \cdots$, be a sequence of orthonormal elements in in a complete inner product space $\mathcal{H}$. The following seven statements are equivalent: [36]

(A) *The ϕ_j is a complete orthonormal system in $\mathcal{H}$.*

(B) *The orthonormal expansion of any element $y \in \mathcal{H}$ converges in norm to y; i.e.*

$$\lim_{n\to\infty}\Big\|y - \sum_{j=1}^{n}(y,\phi_j)\phi_j\Big\|. \tag{4.5.20}$$

(C) *Parseval's identity holds for every $y \in \mathcal{H}$, i.e.*

$$\|y\|^2 = \sum_{j=1}^{\infty}|(y,\phi_j)|^2. \tag{4.5.21}$$

[36] We assume that $\mathcal{H}$ is not finite-dimensional, in order to simplify the formulations. Only minor changes are needed in order to cover the finite-dimensional case.

(C') *The extended Parseval's identity holds for every $x, y \in \mathcal{H}$, i.e.*

$$(x, y) = \sum_{j=1}^{\infty} (x, \phi_j)(\phi_j, y). \tag{4.5.22}$$

(D) *There is no strictly larger orthonormal system containing $\phi_1, \phi_2, \cdots$*

(E) *If $y \in \mathcal{H}$ and $(y, \phi_j) = 0$, $j = 1, 2, \ldots$, then $y = 0$.*

(F) *An element of $\mathcal{H}$ is determined uniquely by its Fourier coefficients, i.e. if $(w, \phi_j) = (y, \phi_j)$, $j = 1, 2, \cdots$, then $w = y$.*

Proof. For the rest of the proof, See Davis [19, pp. 192ff]. □

Theorem 4.5.16.

The converse of statement (F) *holds, i.e. let $\mathcal{H}$ be a complete inner product space, and let a_j be constants such that $\sum_{j=1}^{\infty} |a_j|^2 < \infty$. Then there exists an $y \in \mathcal{H}$, such that $y = \sum_{j=1}^{\infty} a_j \phi_j$ and $(y, \phi_j) = a_j \ \forall\ j$.*

Proof. Omitted. □

4.5.5 Orthogonal Polynomials and Least Squares Approximation

By a family of **orthogonal polynomials** we mean a triangle family of polynomials, which (in the continuous case) is an orthogonal system with respect to a given inner product Expansions of functions in terms of orthogonal polynomials are very useful. They are easy to manipulate, have good convergence properties and usually give a well conditioned representation. The theory of orthogonal polynomials is also of fundamental importance for many problems, which at first sight seem to have little connection with approximation (e.g., numerical integration, continued fractions, and the algebraic eigenvalue problem).

Perhaps the most important example of a family of orthogonal polynomials is the Chebyshev polynomials $T_n(x) = \cos(n \arccos(x))$ introduced in Sec. 3.2.3. These are are orthogonal on $[-1, 1]$ with respect to the weight function $(1 - x^2)^{-1/2}$ and also with respect to a discrete inner product. Their properties can be derived by rather simple methods.

Theorem 4.5.17.

The Chebyshev polynomials have the following two orthogonality properties. Set

$$(f, g) = \int_{-1}^{1} f(x) g(x) (1 - x^2)^{-1/2}\, dx \tag{4.5.23}$$

(the continuous case). Then $(T_0, T_0) = \pi$, *and*

$$(T_j, T_k) = \begin{cases} 0 & \text{if } j \neq k, \\ \pi/2 & \text{if } j = k \neq 0. \end{cases} \tag{4.5.24}$$

Let x_k *be the zeros of* $T_{m+1}(x)$ *and set*

$$(f, g) = \sum_{k=0}^{m} f(x_k)g(x_k), \quad x_k = \cos\Big(\frac{2k+1}{m+1}\frac{\pi}{2}\Big) \tag{4.5.25}$$

(the discrete case). Then $(T_0, T_0) = m + 1$, *and*

$$(T_j, T_k) = \begin{cases} 0 & \text{if } j \neq k, \\ (m+1)/2 & \text{if } j = k \neq 0. \end{cases} \tag{4.5.26}$$

Proof. In the continuous case, let $j \neq k$, $j \geq 0$, $k \geq 0$. From $x = cos\phi$ it follows that $dx = \sin\phi d\phi = (1 - x^2)^{1/2} d\phi$. Hence

$$\begin{aligned}(T_j, T_k) &= \int_0^\pi \cos jx \cos kx\, dx = \int_0^\pi \frac{1}{2}\big(\cos(j-k)x + \cos(j+k)x\big)\, dx \\ &= \frac{1}{2}\Big(\frac{\sin(j-k)\pi}{j-k} + \frac{\sin(j+k)\pi}{j+k}\Big) = 0,\end{aligned}$$

whereby orthogonality is proved.

In the discrete case, set $h = \partial\mu/(m+1)$, $x_\mu = h/2 + \mu h$,

$$(T_j, T_k) = \sum_{\mu=0}^{m} \cos jx_\mu \cos kx_\mu = \frac{1}{2}\sum_{\mu=0}^{m}\big(\cos(j-k)x_\mu + \cos(j+k)x_\mu\big).$$

Using notation from complex numbers ($i = \sqrt{-1}$) we have

$$(T_j, T_k) = \frac{1}{2}\mathrm{Re}\Big(\sum_{\mu=0}^{m} e^{i(j-k)h(1/2+\mu)} + \sum_{\mu=0}^{m} e^{i(j+k)h(1/2+\mu)}\Big). \tag{4.5.27}$$

The sums in (4.5.27) are geometric series with ratios $e^{i(j-k)h}$ and $e^{i(j+k)h}$, respectively. If $j \neq k$, $0 \leq j \leq m$, $0 \leq k \leq m$, then the ratios are never equal to 1, since

$$0 < |\ (j \pm k)h| \leq \frac{2m}{m+1}\pi < \pi.$$

The first sum in (4.5.27) is, then, using the formula for the sum of a geometric series

$$e^{i(j-k)(h/2)}\frac{e^{i(j-k)(m+1)h} - 1}{e^{i(j-k)h} - 1} = \frac{e^{i(j-k)\pi} - 1}{e^{i(j-k)(h/2)} - e^{-i(j-k)(h/2)} - 1} = \frac{(-1)^{j-k} - 1}{2i\sin(j-k)h/2}.$$

The real part of the last expression is clearly zero. An analogous computation shows that the real part of the other sum in (4.5.27) is also zero. Thus the orthogonality property holds in the discrete case also. It is left to the reader to show that the expressions when $j = k$ given in the theorem are correct. □

Example 4.5.10. *Chebyshev interpolation*

Let $p(x)$ denote the Chebyshev interpolation polynomial of degree m. For many reasons it is practical to write this interpolation polynomial in the form

$$p(x) = \sum_{i=0}^{m} c_i T_i(x). \tag{4.5.28}$$

Then using the discrete orthogonality property (4.5.26)

$$c_i = \frac{(f, T_i)}{\|T_i\|^2} = \frac{1}{\|T_i\|^2} \sum_{k=0}^{m} f(x_k) T_i(x_k), \tag{4.5.29}$$

where

$$\|T_0\|^2 = m + 1, \quad \|T_i\|^2 = \tfrac{1}{2}(m+1), \quad i > 0.$$

The recursion formula (3.2.20) can be used for calculating the orthogonal coefficients according to (4.5.29). For computing $p(x)$ with (4.5.28) Clenshaw's algorithm (Theorem 3.2.5) can be used.

Occasionally on is interested in the partial sums of (4.5.28). For example, if the values of $f(x)$ are afflicted with statistically independent errors with standard deviation σ, then the series can be broken off when for the first time

$$\left\| f - \sum_{i=0}^{p} c_i T_i(x) \right\| < \sigma m^{1/2}.$$

We assume in the following that in the continuous case the weight function $w(x) \geq 0$ is such that the **moments**

$$\mu_k = (x^k, 1) = \int_a^b x^k w(x)\, dx. \tag{4.5.30}$$

are defined for all $k \geq 0$, and $\mu_0 > 0$. Note that the inner product (4.5.38) has the property that

$$(xf, g) = (f, xg). \tag{4.5.31}$$

Given a linearly independent sequence of vectors in an inner-product space an orthogonal system can be derived by a process analogous to Gram–Schmidt orthogonalization. The proof below is constructive and leads to a unique construction of the sequence of orthogonal polynomials ϕ_k, $n \geq 1$, with leading coefficients equal to 1.

Theorem 4.5.18.

In an inner product space with the inner product (4.5.38), there is a triangle family of orthogonal polynomials $\phi_k(x)$, $k = 1, 2, 3, \ldots$, such that $\phi_{k+1}(x)$ has exact degree k, and is orthogonal to all polynomials of degree less than k. The family is uniquely determined apart from the fact that the leading coefficients can be given arbitrary positive values.

The monic orthogonal polynomials satisfy the three-term recurrence formula,

$$\phi_{k+1}(x) = (x - \beta_k)\phi_k(x) - \gamma_{k-1}^2\phi_{k-1}(x), \quad k \geq 1, \tag{4.5.32}$$

with initial values $\phi_0(x) = 0$, $\phi_1(x) = 1$. *The recurrence coefficients are given by*

$$\beta_k = \frac{(x\phi_k, \phi_k)}{\|\phi_k\|^2}, \qquad \gamma_{k-1} = \frac{\|\phi_k\|}{\|\phi_{k-1}\|}. \tag{4.5.33}$$

If the weight distribution is symmetric about $x = \beta$, *then* $\beta_k = \beta$ *for all* k.

Proof. By induction: Suppose that the $\phi_j \neq 0$ have been constructed for $1 \leq j \leq k$. We now seek a polynomial ϕ_{k+1} of degree k with leading coefficient equal to 1, which is orthogonal to all polynomials in $\mathcal{P}_k$. Since $\{\phi_j\}_{j=1}^k$ is a triangle family, every polynomial of degree $k-1$ can be expressed as a linear combination of these polynomials. Therefore we can write

$$\phi_{k+1} = x\phi_k - \sum_{i=1}^{k} c_{k,i}\phi_i. \tag{4.5.34}$$

Clearly ϕ_{k+1} has leading coefficient one. The orthogonality condition is fulfilled if and only if

$$(x\phi_k, \phi_j) - \sum_{i=1}^{k} c_{k,i}(\phi_i, \phi_j) = 0, \quad j = 1 : k.$$

But $(\phi_i, \phi_j) = 0$ for $i \neq j$, and thus $c_{k,j}\|\phi_j\|^2 = (x\phi_k, \phi_j)$. From the definition of inner product (4.5.38), it follows that

$$(x\phi_k, \phi_j) = (\phi_k, x\phi_j).$$

But $x\phi_j$ is a polynomial of degree j. Thus if $j < k$, then it is orthogonal to ϕ_k. So $c_{kj} = 0$ for $j < k - 1$. From (4.5.34) it then follows that

$$\phi_{k+1} = x\phi_k - c_{k,k}\phi_k - c_{k,k-1}\phi_{k-1}, \tag{4.5.35}$$

which has the same form as the original assertion of the theorem if, we set

$$\beta_k = c_{k,k} = \frac{(x\phi_k, \phi_k)}{\|\phi_k\|^2}, \qquad \gamma_{k-1}^2 = c_{k,k-1} = \frac{(\phi_k, x\phi_{k-1})}{\|\phi_{k-1}\|^2}. \quad k \geq 1.$$

This shows the first part of (4.5.33).

The expression for γ_{k-1} can be written in another way. If we scalar-multiply (4.5.34) by ϕ_{k+1} we get

$$(\phi_{k+1}, \phi_{k+1}) = (\phi_{k+1}, x\phi_k) - \sum_{i=1}^{k} c_{k,i}(\phi_{k+1}, \phi_i) = (\phi_{k+1}, x\phi_k).$$

Thus $(\phi_{k+1}, x\phi_k) = \|\phi_{k+1}\|^2$, or if we decrease all indices by 1, $(\phi_k, x\phi_{k-1}) = \|\phi_k\|^2$. Substituting this in the expression for γ_{k-1} gives the second part of (4.5.33) ∎

Sometimes it is advantageous to consider corresponding orthonormal polynomials $\hat{\phi}_k(x)$, which satisfy $\|\hat{\phi}_k\| = 1$. Setting $\hat{\phi}_1 = c$, we find

$$\|\hat{\phi}_1\| = \int_a^b c^2 w(x)\,dx = c^2\mu_0 = 1,$$

and thus $\hat{\phi}_1 = 1/\sqrt{\mu_0}$. If we scale the monic orthogonal polynomials according to

$$\phi_k = (\gamma_1 \cdots \gamma_{k-1})\hat{\phi}_k, \quad k > 1, \tag{4.5.36}$$

then we find using (4.5.33) that

$$\frac{\|\hat{\phi}_k\|}{\|\hat{\phi}_{k-1}\|} = \frac{\gamma_1 \cdots \gamma_{k-2}}{\gamma_1 \cdots \gamma_{k-1}} \frac{\|\phi_k\|}{\|\phi_{k-1}\|} = 1.$$

Substituting (4.5.36) in (4.5.32) we obtain a recurrence relation for the orthonormal polynomials

$$\gamma_k\hat{\phi}_{k+1}(x) = (x - \beta_k)\hat{\phi}_k(x) - \gamma_{k-1}\hat{\phi}_{k-1}(x), \quad k \geq 1, \tag{4.5.37}$$

Let $\hat{p}_n$ denote the polynomial of degree n for which

$$\|f - \hat{p}_n\|_\infty = E_n(f) = \min_{p\in\mathcal{P}_{n+1}} \|f - p\|_\infty.$$

Set $p_n = \sum_{j=0}^n c_j\phi_j$, where c_j is the jth Fourier coefficient of f and $\{\phi_j\}$ are the orthogonal polynomials with respect to the inner product

$$(f, g)_w = \int_a^b f(x)g(x)w(x)\,dx, \quad w(x) \geq 0. \tag{4.5.38}$$

If we use the weighted Euclidian norm, $\hat{p}_n$ is of course not a better approximation than p_n. In fact

$$\begin{aligned}\|f - p\|_w^2 &= \int_a^b |f(x) - p_n(x)|^2 w(x)\,dx \\ &\leq \int_a^b |f(x) - \hat{p}_n(x)|^2 w(x)\,dx \leq E_n(f)^2 \int_a^b w(x)\,dx.\end{aligned} \tag{4.5.39}$$

This can be interpreted as saying that a kind of weighted mean of $|f(x) - p_n(x)|$ is less than or equal to $E_n(f)$, which is about as good result as one could demand. The error curve has an oscillatory behavior. In small subintervals $|f(x) - p_n(x)|$ can be significantly greater than $E_n(f)$. This is usually near the ends of the intervals or in subintervals where $w(x)$ is relatively small. Note that from (4.5.39) and Weierstrass approximation theorem it follows that

$$\lim_{n\to\infty} \|f - p\|_{2,w}^2 = 0$$

for every continuous function f. From (4.5.39) one gets after some calculations

$$\sum_{j=n+1}^{\infty} \|\phi_j\|^2 = \|f-p\|_{2,w}^2 \leq E_n(f)^2 \int_a^b w(x)\,dx,$$

which gives one an idea of how quickly the terms in the orthogonal expansion decrease.

In Sec. 4.1.3 we considered the discrete least squares approximation problem

$$f(x) \approx p(x) = \sum_{j=1}^{n} c_j p_j(x) \in \mathcal{P}_n,$$

where $p_1(x), p_2(x), \ldots, p_n(x)$ is a basis for $\mathcal{P}_n$. Then $S(c) = \|p-f\|^2$, the least squares problem to minimize $S(c)$ is equivalent to minimizing the norm of the error function $p-f$. Thus, if we could determine the basis functions so that they constitute an orthogonal system, i.e.

$$(p_i, p_k) = \begin{cases} 0, & i \neq k, \\ \|p_i\|^2 \neq 0, & i = k, \end{cases}$$

then by Theorem 4.5.12 the coefficients of the least squares approximations are

$$c_j = (f, p_j)/(p_j, p_j), \quad j = 1:n. \tag{4.5.40}$$

Example 4.5.11.

For the case $n = 1$, $p_1(x) = 1$, the normal equations reduce to the single equation $(p_0, p_0)c_0 = (f, p_0)$. Hence using the discrete inner product we get

$$c_1 = \frac{1}{\omega} \sum_{i=0}^{m} w_i f(x_i), \quad \omega = \sum_{i=0}^{m} w_i.$$

Here c_1 is said to be a **weighted mean** of the values of the function.

Let $\{x_i\}_{i=1}^m \in (a,b)$ be distinct points and $\{w_i\}_{i=1}^m$ a set of weights and define the weighted discrete inner product of two real-valued functions f and g on the grid $\{x_i\}_{j=1}^m$ by

$$(f,g) = \sum_{i=1}^{m} w_i f(x_i) g(x_i). \tag{4.5.41}$$

From Theorem 4.5.18 it follows that there is a unique associated triangle family of orthogonal polynomials $p_0, p_1, \ldots, p_{m-1}$, with leading coefficients equal to one, that satisfy the three-term recurrence

$$p_{-1}(x) = 0, \quad p_0(x) = 1, \tag{4.5.42}$$

$$p_{k+1}(x) = (x-\alpha_k)p_k(x) - \beta_k p_{k-1}(x), \quad k = 0, 1, 2, \ldots, \tag{4.5.43}$$

where

$$\alpha_k = \frac{(xp_k, p_k)}{\|p_k\|^2}, \qquad \beta_k = \frac{\|p_k\|^2}{\|p_{k-1}\|^2} \quad (k > 0). \tag{4.5.44}$$

If the weight distribution is symmetric about $x = \alpha$, then $\alpha_k = \alpha$ for all k.

For a discrete weight distribution on a grid with m points, the family ends with $p_m(x)$; $p_{m+1}(x)$ becomes zero at each grid point. In the continuous case, the family has infinitely many members.

Using (4.5.33 the coefficients α_k and β_k and the value of the polynomials p_k at the grid points x_i can be recursively computed, in the order

$$\alpha_0, p_1(x_i), \alpha_1, \beta_1, p_2(x_i), \ldots.$$

(Note that β_0 is not needed since $p_{-1} = 0$.) This procedure is called the **Stieltjes procedure**[37]

For $k = m$ the constructed polynomial p_m must be equal to

$$(x - x_0)(x - x_1) \cdots (x - x_m),$$

because this polynomial is zero at all the grid points, and thus orthogonal to all functions. From this it follows that $\|p_{m+1}\| = 0$; thus the computation of α_k cannot be carried out for $k = m + 1$ and the construction stops at $k = m$. This is natural, since there cannot be more than $m + 1$ orthogonal (or even linearly independent) functions on a grid with $m + 1$ points.

There are many computational advantages of using the recurrence relation (4.5.43) for discrete least squares data fitting. In a least squares approximation of the form

$$p(x) = \sum_{k=0}^{n} c_k p_k,$$

the coefficients

$$c_k = (p, p_k)/\|p_k\|^2,$$

are independent of n. In the computation of the coefficients c_k one can make use of the recursion formula (4.5.43). approximations of increasing degree can be recursively generated as follows. Suppose that p_i, $i = 0 : k - 1$, and the least squares approximation p_k of degree k have been computed. In the next step the coefficients β_k, γ_k are computed from (4.5.44) and then p_{k+1} by (4.5.43). The next approximation of f can now be obtained by

$$p_{k+1} = p_k + c_{k+1}p_{k+1}, \qquad c_{k+1} = (f, p_{k+1})/\|p_{k+1}\|^2. \tag{4.5.45}$$

Since p_{k+1} is orthogonal to p_k, an alternative expression for the new coefficient is

$$c_{k+1} = (r_k, p_{k+1})/\|p_{k+1}\|^2, \qquad r_k = f - p_k. \tag{4.5.46}$$

[37] Thomas Jan Stieltjes (1856–1894), was born in the Netherlands. After working with astronomical calculations at the Observatory in Leiden he got a university position in Toulouse, France. His work on continued fractions and the moment problem and invented a new concept of integral.

Assuming unit weights and that the grid is symmetric the coefficients α_k, c_k and the orthogonal functions p_k at the grid points can be using the Stieltjes procedure in about $4mn$ flops. If there are differing weights, then about mn additional operations are needed; similarly, mn additional operations are required if the grid is not symmetric. If the orthogonal coefficients are determined simultaneously for several functions on the same grid, then only about mn additional operations per function are required. (In the above, we assume $m \gg 1$, $n \gg 1$.) Hence the procedure is much more economical than the general methods based on normal equations which require $O(mn^2)$ flops.

Mathematically the two formulas (4.5.45) and (4.5.46) for c_{k+1} are equivalent. In finite precision, as higher degree polynomials p_{k+1} are computed, they will gradually lose orthogonality to previously computed p_j, $j \leq k$. In practice there is an advantage in using (4.5.46) since cancellation then will mostly take place in computing the residual $r_k = f - p_k$, and then the inner product (r_k, p_{k+1}) is computed accurately. Theoretically the error $\|p_k - f\|$ must be a non-increasing function of k. I However, when using the first formula one sometimes finds that *the residual norm increases when the degree of the approximation is increased*! With the modified formula (4.5.46) this is very unlikely to happen; see Problem 8.

Note that for $n = m$ we obtain the (unique) interpolation polynomial for the given points. Often the error decreases rapidly with k and then p_k provides a good representation of f already for small values of k. With some nets e.g., equidistant nets, one should choose n less than $2\sqrt{m}$, since otherwise *the approximation polynomial will have large oscillatory behavior between the grid points.*

When the coefficients c_j in the orthogonal expansion are known, then the easiest way to compute the numerical values of $p(x)$ is to use **Clenshaw's algorithm**; see Theorem 3.2.4.

For equidistant data, the **Gram polynomials** $\{P_{n,m}\}_{n=0}^{m}$ are of interest.[38] These polynomials are orthogonal with respect to the inner product

$$(f, g) = \frac{1}{m} \sum_{i=1}^{m} f(x_i) g(x_i), \quad x_i = -1 + (2i - 1)/m.$$

The weight distribution is symmetric around the origin $\alpha_k = 0$. and for the *monic* Gram polynomials the recursion formula is (see [2])

$$\begin{aligned} &P_{-1,m}(x) = 0, \qquad P_{0,m} = 1, \\ &P_{n+1,m}(x) = xP_{n,m}(x) - \beta_{n,m} P_{n-1,m}(x), \qquad n = 0 : m - 1, \end{aligned}$$

where $(n < m)$

$$\beta_{n,m} = \frac{n^2}{4n^2 - 1} \left(1 - \frac{n^2}{m^2}\right).$$

When $n \ll m^{1/2}$, these polynomials are well behaved. However, when $n \geq m^{1/2}$, the Gram polynomials have very large oscillations between the grid points,

[38] Jørgen Pedersen Gram (1850–1916), Danish mathematician, worked on probability and numerical analysis. Gram is now best remembered for the Gram–Schmidt orthogonalization process.

and a large maximum norm in $[-1, 1]$. This fact is related to the recommendation that when fitting a polynomial to *equidistant data*, one should never choose n larger than about $2m^{1/2}$.

4.5.6 Statistical Aspects of the Method of Least Squares

One of the motivations for the method of least squares is that it effectively reduces the influence of random errors in measurements. Let $f \in \mathbf{R}^m$ be a vector of observations that is related to a parameter vector $c \in \mathbf{R}^n$ by the linear relation

$$f = Ac + \epsilon, \quad A \in \mathbf{R}^{m\times n}, \tag{4.5.47}$$

where A is known matrix of full column rank and $\epsilon \in \mathbf{R}^m$ is a vector of random errors. We assume here that ϵ_i, $i = 1 : m$ has zero mean and covariance equal to σ^2, and that ϵ_i and ϵ_i are **uncorrelated** if $i \neq j$, that is

$$E(\epsilon) = 0, \qquad \text{var}(\epsilon) = \sigma^2 I,$$

(Recall the definitions of mean value and correlation in Sec. 1.5.1.) The parameter c is then a random vector, which we want to estimate in terms of the known quantities A and f.

Let $y^T c$ be a linear functional of the parameter c in (4.5.47). We say that $\theta = \theta(A, f)$ is an unbiased linear estimator of $y^T c$ if $E(\theta) = y^T c$. It is a **best linear unbiased estimator** (BLUE) if θ) has the smallest variance among all such estimators.

The **Gauss–Markov theorem**[39] places the method of least squares on a sound theoretical basis.

Theorem 4.5.19.

Consider a linear model (4.5.47), where ϵ is an uncorrelated random vector with zero mean and variance equal to σI. Then the best linear unbiased estimator of any linear functional $y^T c$ is $y^T \hat{c}$, where

$$\bar{c} = (A^T A)^{-1} A^T f$$

is the least squares estimator obtained by minimizing the sum of squares $\|f - Ac\|_2^2$.

The covariance matrix of the least squares estimate $\hat{c}$ equals

$$\text{var}(\hat{c}) = \sigma^2 (A^T A)^{-1}. \tag{4.5.48}$$

Furthermore, the quadratic form

$$s^2 = \|f - A\hat{c}\|_2^2/(m - n). \tag{4.5.49}$$

is an unbiased estimate of σ^2, i.e. $E(s^2) = \sigma^2$.

[39] This theorem is originally due to C. F. Gauss 1821. His contribution was somewhat neglected until rediscovered by the Russian mathematician A. A Markov in 1912.

Proof. See Zelen [58, pp. 560–561]. □

Suppose that the values of a function have been measured in the points $x_1, x_2, \ldots, x_m$. Let $f(x_p)$ be the measured value, and let $\bar{f}(x_p)$ be the "true" (unknown) function value, which is assumed to be the same as the *expected value* of the measured value. Thus *no systematic errors* are assumed to be present. Suppose further that *the errors in measurement at the various points are statistically independent*. Then we have a linear model $f(x_p) = \bar{f}(x_p) + \epsilon$, where

$$E(\epsilon) = 0, \qquad \text{var}(\epsilon) = \text{d}iag\,(\sigma_1^2, \ldots, \sigma_n^2), \tag{4.5.50}$$

where E denotes expected value and var variance. The problem is to use the measured data to estimate the coefficients in the series

$$f(x) = \sum_{j=1}^{n} c_j \phi(x), \quad n \le m.$$

where $\phi_1, \phi_2, \ldots, \phi_n$ are known functions. According to Theorem 4.5.19 the estimates c_j^*, which one gets by minimizing the sum

$$\sum_{p=1}^{m} w_p \Big(f(x_p) - \sum_{j=1}^{n} c_j \phi_j(x_p) \Big)^2, \quad w_p = \sigma_p^{-2},$$

have a smaller variance than the values one gets by any other linear unbiased estimation method. This minimum property holds not only for the estimates of the coefficients c_j, but also for every linear functional of the coefficients, e.g., the estimate

$$f_n^*(\alpha) = \sum_{j=1}^{n} c_j^* \phi(\alpha)$$

of the value $f(\alpha)$ at an arbitrary point α.

Suppose now that $\sigma_p = \sigma$ for all p and that the functions $\{\phi_j\}_{j=1}^n$ form an *orthonormal system* with respect to the discrete inner product

$$(f, g) = \sum_{p=1}^{m} f(x_p) g(x_p).$$

Then the least squares estimates are $c_j^* = (f, \phi_j)$, $j = 1 : n$. By Theorem 4.5.19 the estimates c_j^* and c_k^* are *uncorrelated if* $j \ne k$ and

$$E\{(c_j^* - \bar{c}_j)(c_k^* - \bar{c}_k)\} = \begin{cases} 0, & \text{if } j \ne k; \\ \sigma^2 & \text{if } j = k, \end{cases}$$

From this it follows that

$$\text{var}\{f_n^*(\alpha)\} = \text{var}\Big\{ \sum_{j=1}^{n} c_j^* \phi_j(\alpha) \Big\} = \sum_{j=1}^{n} \text{var}\{c_j^*\} |\phi_j(\alpha)|^2 = \sigma^2 \sum_{j=1}^{n} |\phi_j(\alpha)|^2.$$

As an average, *taken over the grid of measurement points*, the variance of the smoothed function values is

$$\frac{1}{m}\sum_{j=1}^{n}\operatorname{var}\{f_n^*(x_i)\} = \frac{\sigma^2}{m}\sum_{j=1}^{n}\sum_{i=1}^{m}|\phi_j(x_i)|^2 = \sigma^2\frac{n}{m}.$$

Between the grid points, however, the variance can in many cases be significantly larger. For example, when fitting a polynomial to measurements in *equidistant points*, the Gram polynomial $P_{n,m}$ can be much larger between the grid points when $n > 2m^{1/2}$. Set

$$\sigma_I^2 = \sigma^2\sum_{j=1}^{n}\tfrac{1}{2}\int_{-1}^{1}|\phi(x)|^2\,dx.$$

Thus σ_I^2 is an average variance for $f_n^*(x)$ *taken over the entire interval* $[-1,1]$. The following values for the ratio ρ between σ_I^2 and $\sigma^2(n+1)/(m+1)$ when $m = 42$ were obtained by H. Björk:

$n+1$	5	10	15	20	25	30	35
ρ	1.0	1.1	1.7	26	$7\cdot10^3$	$1.7\cdot10^7$	$8\cdot10^{11}$

These results are related to the recommendation that one should choose $n < 2m^{1/2}$ when fitting a polynomial to equidistant data. This recommendation seems to contradict the Gauss–Markov theorem, but in fact it just means that one gives up the requirement that there be no systematic errors. Still it is remarkable that this can lead to such a drastic reduction of the variance of the estimates.

If the measurement points are the Chebyshev abscissae, then no difficulties arise in fitting polynomials to data. The Chebyshev polynomials have a magnitude between grid points not much larger than their magnitude at the grid points. In this case the choice of n when m is given, is a question of compromising between taking into account the truncation errors (which decreases as n increases) and the random errors (which grow when n increases). If f is a sufficiently smooth function then in the Chebyshev case $|c_j|$ decreases quickly with j. In contrast the part of c_j which comes from errors in measurements varies randomly with a magnitude of about $\sigma(2/(m+1)^{1/2}$, using (4.5.25)) and $\|T_j\|^2 = (m+1)/2$. The expansion should be broken off when the coefficients begin to "behave randomly". An expansion in terms of Chebyshev polynomials can hence be used for *filtering away the "noise" from the signal, even when σ is initially unknown.*

Example 4.5.12.

Fifty-one equidistant values of a certain analytic function were rounded to four decimals. In Figure 4.5.?, a semilog diagram is given which shows haw $|c_i|$ varies in an expansion in terms of the Chebyshev polynomials for this data. For $i > 20$ (approximately) the contribution due to noise dominates the contribution due to signal. Thus it is sufficient to break off the series at $n = 20$.

Figure 4.5.1. *Magnitude of coefficients c_i in a Chebyshev expansion of an analytic function contaminated with roundoff noise.*

Review Questions

1. State the axioms that any norm must satisfy. Define the maximum norm and the Euclidean norm for a continuous function f on a closed interval.
2. Define $\text{dist}(f, P_n)$, and state Weierstrass' approximation theorem.
3. Prove the Pythagorean theorem in an inner product space.
4. Define and give examples of orthogonal systems of functions.
5. Formulate and prove Bessel's inequality and Parseval's identity, and interpret them geometrically.
6. (a) Give some reasons for using orthogonal polynomials in polynomial approximation with the method of least squares.

   (b) Give some argument against the assertion that orthogonal polynomials are difficult to work with.
7. The Gram polynomials are examples of orthogonal polynomials. With respect to what inner product are they orthogonal?

Problems and Computer Exercises

1. Compute $\|f\|_\infty$ and $\|f\|_2$ for the function $f(x) = (1+x)^{-1}$ on the interval $[0, 1]$.
2. Determine straight lines which approximate the curve $y = e^x$ such that
   (a) the discrete Euclidean norm of the error function on the grid $(-1, -0.5, 0, 0.5, 1)$ is as small as possible;

(b) the Euclidean norm of the error function on the interval $[-1, 1]$ is as small as possible.

(c) the line is tangent to $y = e^x$ at the point $(0, 1)$, i.e. the Taylor approximation at the midpoint of the interval.

3. Determine, for $f(x) = \pi^2 - x^2$, the "cosine polynomial" $f^* = \sum_{j=0}^{n} c_j \cos jx$, which makes $\|f^* - f\|_2$ on the interval $[0, \pi]$ as small as possible.

4. (a) Show that on any interval containing the points $-1, -1/3, 1/3, 1$,

$$E_2(f) \geq \frac{1}{8}\Big|f(1) - 3f(1/3) + 3f(-1/3) - f(-1)\Big|.$$

(b) Compute the above bound and the actual value of $E_2(f)$ for $f(x) = x^3$.

5. (a) Let a scalar product be defined by $(f, g) = \int_a^b f(x)g(x)\,dx$. Calculate the matrix of normal equations, when $\phi_j(x) = x^j$, $j = 0 : n$, when $a = 0$, $b = 1$.

(b) Do the same when $a = -1$, $b = 1$. Show how in this case the normal equations can be easily decomposed into two systems, with approximately $(n + 1)/2$ equations in each.

6. Verify the formulas for $\|\phi_j\|^2$ given in Example 4.5.9.

7. (a) Show that $\|f - g\|$ $ge\|f\| - \|g\|$ for all norms. (use the axioms mentioned in Sec. 4.5.1.)

(b) Show that if $\{c_j\}_0^n$ is a set of real numbers and if $\{f_j\}_0^n$ is a set of vectors, then $\|\sum c_j f_j\| \leq \sum |c_j| \|f_j\|$.

8. Let $G \in \mathbf{R}^{n\times n}$ be a symmetric positive definite matrix. Show that an inner product is defined by the formula $(u, v) = u^T G v$. Show that $A^* = G^{-1}A^T G$.

9. In a space of complex-valued twice differentiable functions of t, which vanish at t=0 and t=1, let the inner product be:

$$(u, v) = \int_0^1 u(t)\bar{v}(t)\,dt.$$

What is the adjoint of the operator $A = d/dt$? Is it true that the operator iA is self-adjoint, and that $-A^2$ is self-adjoint and positive definite?

10. a) Show that, in a real inner-product space,

$$4(u, v) = \|u + v\|^2 - \|u - v\|^2.$$

In a complex space this gives only the real part of the inner product. Show that one has to add $\|u - iv\|^2 - \|u + iv\|^2$.

(b) This can be used to reduce many questions concerning inner-products to questions concerning norms. For example, in a general inner product space a **unitary operator** is defined by the requirement that $\|Au\| = \|u\|$ $\forall u$. Show that $(Au, Av) = (u, v)$ $\forall u, v$.

Note, however, that the relation $(u, Au) = (Au, u)$ $\forall u$, which, in a real space, holds for every operator A, does *not* imply that $(u, Av) = (Au, v)$ $\forall u, v$. The latter holds only if A is self-adjoint.

11. Show that $(AB)^* = B^*A^*$. Also show that if C is self-adjoint and positive definite, then A^*CA is so too. (A is not assumed to be self-adjoint.)

12. Show that

$$(A^{-1})^* = ((A^*))^{-1}, \qquad (A^p)^* = ((A^*))^p,$$

for all integers p, provided that the operators mentioned exist. Is it true that C^p is self-adjoint and positive definite, if C is so?

13. Show the following **minimum property** of orthogonal polynomials: Among all $n$the degree polynomials p_n with leading coefficient 1, the smallest value of

$$\|p_n\|^2 = \int_a^b p_n^2(x)w(x)dx, \qquad w(x) \geq 0$$

is obtained for $p_n = \phi_n/A_n$, where ϕ_n is the orthogonal polynomial with leading coefficient A_n associated with the weight distribution $w(x)$.

Hint: Determine the best approximation to x^n in the above norm or consider the expansion $p_n = \phi_n/A_n + \sum_{j=0}^{n-1} c_j\phi_j$.

14. Verify the formulas for $\|T_j\|^2$ given in Theorem 4.5.17.

14. (a) Write a MATLAB function `c = stieltjes(x,y,w,n)` that computes the least squares polynomial fit to data (x_i, f_i) and weights w_i, $i = 1 : n$, using the Stieltjes procedure. Compute the orthogonal polynomials p_j, from the recursion (4.5.43) and the coefficients c_j, $j = 1 : n$, from (4.5.45) or (4.5.45).

(b) (L. F. Shampine) Apply the function in (a) to the case when $f_i = x_i^7$, $w_i = 1/(f_i)^2$, and $n = 20$. Compute and print the error $\|p - f\|$, for $n = 1 : 10$ using the expression (4.5.45) for c_{k+1}. Note that the fits for $k > 7$ should be exact!

(c) Repeat the calculations, now using the modified formula (4.5.45). Compare the error for $n = 1 : 10$ with the results in (a).

4.6 Trigonometric Interpolation and Fourier Transforms

Many natural phenomena, e.g., acoustical and optical, are of a periodic character. For instance, it is known that a musical sound is composed of regular oscillations, partly a fundamental tone with a certain frequency f, and partly overtones with frequencies $2f$, $3f$, $4f$,.... The ratio of the strength of the fundamental tone to that of the overtones is decisive for our impression of the sound. Sounds, which are free from overtones occur, for instance, in electronic music, where they are called pure sine tones.

In an electronic oscillator, a current is generated whose strength at time t varies according to the formula $r\sin(\omega t + v)$, where r is called the amplitude of the oscillation; ω is called the angular frequency, and is equal to 2π times the frequency; v is a constant which defines the state at the time $t = 0$. In a loudspeaker, variations of current are converted into variations in air pressure which, under ideal conditions, are described by the same function. In practice, however, there is always a certain

distortion, overtones occur. The variations in air pressure which reach the ear can, from this viewpoint, be described as a sum of the form

$$\sum_{k=0}^{\infty} r_k \sin(k\omega t + v_k). \tag{4.6.1}$$

The separation of a periodic phenomenon into a fundamental tone and overtones permeates not only acoustics, but also many other areas. It is related to an important, purely mathematical theorem, first given by Fourier[40] According to this theorem, every function $f(t)$ with period $2\pi/\omega$ can, under certain very general conditions, be expanded in a series of the form (4.6.1). (A function has period p if $f(t+p) = f(t)$, for all t.) A more precise formulation will be given later in Theorem 4.6.2.

An expansion of the form of (4.6.1) can be expressed in many equivalent ways. If we set $a_k = r_k \sin v_k$, $b_k = r_k \cos v_k$, then using the addition theorem for the sine function we can write

$$f(t) = \sum_{k=0}^{\infty} (a_k \cos k\omega t + b_k \sin k\omega t), \tag{4.6.2}$$

where a_k, b_k are real constants. Another form, which is often the most convenient, can be found with the help of Euler's formulas,

$$\cos x = \frac{1}{2}(e^{ix} + e^{-ix}), \qquad \sin x = \frac{1}{2i}(e^{ix} - e^{-ix}), \quad (i = \sqrt{-1}).$$

Then one gets

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{ik\omega t}, \tag{4.6.3}$$

where

$$c_0 = a_0, \qquad c_k = \frac{1}{2}(a_k - ib_k), \qquad c_{-k} = \frac{1}{2}(a_k + ib_k), \quad k = 1, 2, 3, \ldots. \tag{4.6.4}$$

In the rest of this chapter we shall use the term **Fourier series** to denote an expansion of the form of (4.6.3) or (4.6.4). We shall call the partial sums of the form of these series **trigonometric polynomials**. Sometimes the term **spectral analysis** is used to describe the above methods.

Fourier series are valuable aids in the study of phenomena which are periodic in time (vibrations, sound, light, alternating currents, etc.) or in space (waves, crystal structure, etc.). One very important area of applications is in digital signal and image processing, which is used in interpreting radar and sonar signals. Another is statistical time series, which are used in communications theory, control theory, and the study of turbulence. For the numerical analyst, Fourier analysis is partly

[40] Jean Baptist Joseph Fourier (1768–1830), French mathematician and engineer. In 1807 Fourier presented before the French Academy his famous theorem.

a very common computational task and partly an important aid in the analysis of properties of numerical methods.

Basic formulas and theorems are derived in Sec. 4.6.1, which relies to a great extent on the theory in Sec. 4.5. Modifications of pure Fourier methods are used as a means of analyzing nonperiodic phenomena; see, e.g., Sec. 4.6.2 (periodic continuation of functions) and Sec. 4.6.3 (Fourier transforms). The approximation of Fourier transforms using sampled data and discrete Fourier transforms are treated in Sec. 4.6.4. FFT (Fast Fourier Transforms) algorithms have had an enormous impact and have caused a complete change of attitude toward what can be done using discrete Fourier methods. Sec. 4.7 treats the computational aspects of the basic FFT algorithms.

4.6.1 Basic Formulas and Theorems

We shall study *functions with period* 2π. If a function of t has period L, then the substitution $x = 2\pi t/L$ transforms the function to a function of x with period 2π. We assume that the function can have complex values, since the complex exponential function is convenient for manipulations.

The inner product of two complex-valued functions f and g of period 2π is defined in the following way (the bar over g indicates complex conjugation)

$$(f, g) = \int_{-\pi}^{\pi} f(x)\bar{g}(x)dx, \quad \text{(continuous case)}. \tag{4.6.5}$$

(It makes no difference what interval one uses, as long as it has length 2π—the value of the inner product is unchanged.) Often the function f is known only at equidistant arguments $x_\alpha = 2\pi\alpha/N$, $\alpha = 0 : N-1$. In this case we define

$$(f, g) = \sum_{\alpha=0}^{N-1} f(x_\alpha)\bar{g}(x_\alpha), \quad x_\alpha = \frac{2\pi\alpha}{N} \quad \text{(discrete case)}. \tag{4.6.6}$$

As usual the norm of the function f is defined by $\|f\| = (f, f)^{1/2}$. One can make computations with these inner products in the same way as with the inner products defined in Sec. 4.3.1, with certain obvious modifications. Notice especially that $(g, f) = \overline{(f, g)}$. In the continuous case,

Theorem 4.6.1.

The following **orthogonality relations** *hold for the functions*

$$\phi_j(x) = e^{ijx}, \quad j = 0, \pm 1, \pm 2, \ldots.$$

Continuous case:

$$(\phi_j, \phi_k) = \begin{cases} 2\pi, & \text{if } j = k, \\ 0, & \text{if } j \neq k. \end{cases}$$

Discrete case:

$$(\phi_j, \phi_k) = \begin{cases} N, & \text{if } (j-k)/N \text{ is an integer}, \\ 0, & \text{otherwise}. \end{cases}$$

Proof. In the continuous case, if $j \neq k$, it holds that

$$(\phi_j, \phi_k) = \int_{-\pi}^{\pi} e^{ijx} e^{-ikx} dx = \Big|_{-\pi}^{\pi} \frac{e^{i(j-k)x}}{i(j-k)} = \frac{(-1)^{j-k} - (-1)^{j-k}}{i(j-k)} = 0.$$

whereby orthogonality is proved. For $j = k$

$$(\phi_k, \phi_k) = \int_{-\pi}^{\pi} e^{ikx} e^{-ikx} dx = \int_{-\pi}^{\pi} 1 \; dx = 2\pi.$$

In the discrete case, set $h = 2\pi/N$, $x_\alpha = h\alpha$,

$$(\phi_j, \phi_k) = \sum_{\alpha=0}^{N-1} e^{ijx_\alpha} e^{-ikx_\alpha} = \sum_{\alpha=0}^{N-1} e^{i(j-k)h\alpha}.$$

This is a geometric series with ratio $q = e^{i(j-k)h}$. If $(j-k)/N$ is an integer, then $q = 1$ and the sum is N. Otherwise $q \neq 1$, but $q^N = e^{i(j-k)2\pi} = 1$. From the summation formula of a geometric series

$$(\phi_j, \phi_k) = (q^N - 1)/(q - 1) = 0.$$

□

If one knows that the function $f(x)$ has an expansion of the form

$$f = \sum_{j=a}^{b} c_j \phi_j,$$

where $a = -\infty$, $b = \infty$ in the continuous case and $a = 0$, $b = N - 1$ in the discrete case, then it follows formally that

$$(f, \phi_k) = \sum_{j=a}^{b} c_j (\phi_j, \phi_k) = c_k (\phi_k, \phi_k), \quad a \leq k \leq b,$$

since $(\phi_j, \phi_k) = 0$ for $j \neq k$. Thus, changing k to j, we have

$$c_j = \frac{(f, \phi_j)}{(\phi_j, \phi_j)} = \begin{cases} \dfrac{1}{2\pi} \displaystyle\int_{-\pi}^{\pi} f(x) e^{-ijx} dx, & \text{in the continuous case;} \\ \dfrac{1}{N} \displaystyle\sum_{\alpha=0}^{N-1} f(x_\alpha) e^{-ijx_\alpha}, & \text{in the discrete case).} \end{cases} \tag{4.6.7}$$

These coefficients are called **Fourier coefficients**, see the more general case in Theorem 4.3.??. The purely formal treatment given above is, in the discrete case, justified by Theorem 4.3.??. In the continuous case, more advanced methods are required, but we shall not go into this further.

Now consider the *continuous* case. From a generalization of Theorem 4.3.??, we know that

$$\Big\| f - \sum_{j=-n}^{n} k_j \phi_j \Big\|, \quad n < \infty,$$

becomes as small as possible if we choose $k_j = c_j$, $-n \le j \le n$.

In accordance with (4.6.4), set $a_j = c_j + c_{-j}$, $b_j = i(c_j - c_{-j})$. Then with $a_0 = 2c_0$,

$$a_j = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos jx\,dx, \qquad b_j = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin jx\,dx, \quad j \ge 0, \tag{4.6.8}$$

$$\begin{aligned}\sum_{j=-N}^{N} c_j e^{ijx} &= c_0 + \sum_{j=1}^{N}\big(c_j(\cos jx + i\sin jx) + c_{-j}(\cos jx - i\sin jx)\big)\\ &= \frac{1}{2}a_0 + \sum_{j=1}^{N}(a_j\cos jx + b_j\sin jx).\end{aligned}$$

(Notice that the factors preceding the integral are different in the expressions for c_j and for a_j, b_j, respectively.)

Theorem 4.6.2. *Fourier Analysis, Continuous Case.*

Every piecewise continuous function f with period 2π can be associated with a Fourier series in the following two ways:

$$\frac{1}{2}a_0 + \sum_{j=1}^{\infty}(a_j\cos jx + b_j\sin jx),$$

$$\sum_{j=-\infty}^{\infty} c_j e^{ijx}.$$

The coefficients a_j, b_j, and c_j can be computed using (4.6.8) *in the first case and* (4.6.7) *in the second case. If f and its first derivative are everywhere continuous, then the series is everywhere convergent to $f(x)$. If f and f' have a finite number of jump discontinuities in each period, the series gives the mean of the limiting values on the right and on the left of the relevant point. The partial sums of the above expansions give the best possible approximations to $f(x)$ by trigonometric polynomials, in the least squares sense.*

The proof of the convergence results is outside the scope of this book (see, however, the beginning of Sec. 4.7; see also Courant and Hilbert [16].) The rest of the assertions follow from previously made calculations in Theorem 4.6.1 and the comments following; see also the proof of Theorem 4.3.??.

The more regular a function is, the faster its Fourier series converges.

Theorem 4.6.3.
If f and its derivatives up to and including order k are periodic and everywhere continuous, and if $f^{(k+1)}$ is piecewise continuous, then

$$|c_j| \le j^{-(k+1)} \|f^{(k+1)}\|_\infty. \tag{4.6.9}$$

This useful result is relatively easy to prove using (4.6.7) and integrating by parts $k+1$ times.

Theorem 4.6.4.
If f is an even function, i.e. if

$$f(x) = f(-x) \quad \forall x,$$

then $b_j = 0$ for all j; thus the Fourier series becomes a cosine series.
If f is an odd function, i.e. if

$$f(x) = -f(-x) \quad \forall x,$$

then $a_j = 0$ for all j; thus the Fourier series becomes a sine series.
The proof is left as an exercise to the reader (use the formulas in (4.6.10)).

Example 4.6.1. *Fourier Expansion of a Rectangular Wave*
Make a periodic continuation outside the interval $(-\pi, \pi)$ of the function

$$f(x) = \begin{cases} -1, & -\pi < x < 0, \\ 1, & 0 < x < \pi, \end{cases},$$

see Figure 4.6.3.

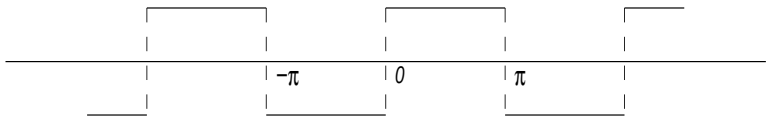


Figure 4.6.1. *Rectangular wave*

The function f is odd, so $a_j = 0$ for all j, and

$$b_j = \frac{2}{\pi} \int_0^\pi \sin jx \, dx = \frac{2}{j\pi}(1 - \cos j\pi).$$

Hence $b_j = 0$ if j is even, and $b_j = 4/(j\pi)$ if j is odd, and

$$f(x) = \frac{4}{\pi}\Big(\sin x + \frac{\sin 3x}{3} + \frac{\sin 5x}{5} + \cdots\Big).$$

Notice that the coefficients c_j decay as j^{-1} in agreement with Theorem 4.6.3. The sum of the series is zero at the points where f has a jump discontinuity; this agrees with the fact that the sum should equal the average of the limiting values to the left and to the right of the discontinuity. The Euler transformation can be used for accelerating the convergence of such series, except in the immediate vicinity of singular points, see Problem 3 of Sec. 4.6.

Theorem 4.5.13 and its corollary, Parseval's identity

$$2\pi \sum_{j=-\infty}^{\infty} |c_j|^2 = \|f\|^2 = \int_{-\pi}^{\pi} |f(x)|^2 \, dx, \tag{4.6.10}$$

are of great importance in many applications of Fourier analysis. The integral in (4.6.10) can be interpreted as the "energy" of the function $f(x)$.

Although the data to be treated in Fourier analysis are often continuous in the time or space domain, for computational purposes this data must usually be represented in terms of a finite discrete sequence. For example, a function $f(t)$ of time, is recorded at evenly spaced intervals Δ in time $f_i = f(i\Delta)$, $i = 0, 1, 2, \ldots$. Such data can be analyzed by discrete Fourier analysis.

Theorem 4.6.5. *Fourier Analysis, Discrete Case*

Every function, defined on the equidistant grid $\{x_0, x_1, \ldots, x_{N-1}\}$, *where* $x_\alpha = 2\pi\alpha/N$, *can be interpolated by the trigonometric polynomial*

$$f(x) = \begin{cases} \displaystyle\sum_{j=-k}^{k+\theta} c_j e^{ijx}, \\ \displaystyle\tfrac{1}{2}a_0 + \sum_{j=1}^{k}(a_j \cos jx + b_j \sin jx) + \frac{1}{2}\theta a_{k+1}\cos(k+1)x, \end{cases} \tag{4.6.11}$$

Here

$$\theta = \begin{cases} 1, & \text{if } N \text{ even}, \\ 0, & \text{if } N \text{ odd}, \end{cases}, \qquad k = \begin{cases} N/2 - 1, & \text{if } N \text{ even}, \\ (N-1)/2, & \text{if } N \text{ odd}, \end{cases} \tag{4.6.12}$$

and

$$c_j = \frac{1}{N}\sum_{\alpha=0}^{N-1} f(x_\alpha) e^{-ijx_\alpha}, \tag{4.6.13}$$

$$a_j = \frac{2}{N}\sum_{\alpha=0}^{N-1} f(x_\alpha)\cos jx_\alpha, \qquad b_j = \frac{2}{N}\sum_{\alpha=0}^{N-1} f(x_\alpha)\sin jx_\alpha. \tag{4.6.14}$$

If the sums in (4.6.11) *are terminated when* $|j| < k + \theta$, *then one obtains the trigonometric polynomial which is the best least squares approximation, among all trigonometric polynomials with the same number of terms, to* f *on the grid.*

Proof. The expression for c_j was formally derived previously, see (4.6.7), and the derivation is justified by Theorem 4.3.??. By Eqs. (4.6.13)–(4.6.14)

$$a_j = c_j + c_{-j}, \qquad b_j = i(c_j - c_{-j}), \qquad c_{k+1} = \frac{1}{2}a_{k+1}.$$

The two expressions for $f(x)$ are equivalent, because

$$\begin{aligned}\sum_{j=-k}^{k+\theta} c_j e^{ijx} &= c_0 + \sum_{j=1}^{k}\big(c_j(\cos jx + i\sin jx) + c_{-j}(\cos jx - i\sin jx)\big) \\ &\quad + \theta c_{k+1}\cos(k+1)x \\ &= c_0 + \sum_{j=1}^{k}(a_j\cos jx + b_j\sin jx) + \frac{1}{2}\theta a_{k+1}\cos(k+1)x.\end{aligned}$$

□

The function $f(x)$ coincides *on the grid* with the function

$$f^*(x) = \sum_{j=0}^{N-1} c_j e^{ijx}, \tag{4.6.15}$$

because $e^{-i(N-j)x_\alpha} = e^{ijx_\alpha}$, $c_{-j} = c_{N-j}$. The functions f and f^* are, however, not identical between the grid points.

Notice that the calculations required to compute the coefficients c_j according to (4.6.13), **Fourier analysis**, are of essentially the same type as the calculations needed to compute $f^*(x)$ at the grid points

$$x_\alpha = 2\pi\alpha/N, \quad \alpha = 0 : N-1,$$

when the expansion in (4.6.15) is known, so-called **Fourier synthesis**. Both calculations can be performed very efficiently using FFT algorithms; see Sec. 4.7.

Functions of several variables are treated analogously. Quite simply, one takes one variable at a time. As an example, consider the discrete case, with two variables. Set

$$x_\alpha = 2\pi\alpha/N, \qquad y_\beta = 2\pi\beta/N,$$

and assume that $f(x_\alpha, y_\beta)$ is known for $\alpha = 0 : N-1$, $\beta = 0 : N-1$. Set

$$c_j(y_\beta) = \frac{1}{N}\sum_{\alpha=0}^{N-1} f(x_\alpha, y_\beta)e^{-ijx_\alpha},$$

$$c_{j,k} = \frac{1}{N}\sum_{\beta=0}^{N-1} c_j(y_\beta)e^{-iky_\beta}.$$

From Theorem 4.6.5, then (with obvious changes in notations),

$$c_j(y_\beta) = \sum_{k=0}^{N-1} c_{j,k} e^{iky_\beta},$$

$$f(x_\alpha, y_\beta) = \sum_{j=0}^{N-1} c_j(y_\beta) e^{ijx_\alpha} = \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} c_{j,k} e^{(ijx_\alpha + iky_\beta)}.$$

The above expansion is of considerable importance, e.g., in crystallography.

4.6.2 Periodic Continuation of a Function

Sometimes Fourier series are used for functions which are defined only on the interval $(-\pi, \pi)$. The series then defines a *periodic continuation of the function* outside the interval (see Figure 4.6.2). Thus the definition is extended so that $f(x) = f(x+2\pi)$ for all x.

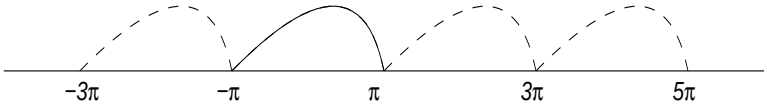


Figure 4.6.2. *Periodic continuation of a function outside* $[-\pi, \pi]$.

With this method, discontinuities in the values of the function or its derivatives can occur at $x = \pi$. This singularity can give rise to a slow rate of convergence, even for a function with good regularity properties in the open interval $(-\pi, \pi)$; cf. Theorem 4.6.3.

There are other ways to make a continuation of a function outside its interval of definition. If the function is defined in $[0, \pi]$, and if $f(0) = f(\pi) = 0$, then one can continue f to the interval $[-\pi, 0]$ by making the definition $f(x) = f(-x)$; thereafter the function is periodically continued outside $[-\pi, \pi]$ by $f(x) = f(x+2\pi)$, see Figure 4.6.3. Since the resulting function is an *odd* function, by Theorems 4.6.4

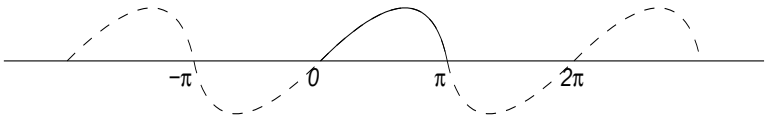


Figure 4.6.3. *Periodic continuation of* f *outside* $[0, \pi]$ *as an odd function.*

and 4.6.5, its Fourier expansion becomes a sine series:

Continuous case:

$$\sum_{j=1}^{\infty} b_j \sin jx, \qquad b_j = \frac{2}{\pi} \int_0^\pi f(x) \sin jx \, dx. \tag{4.6.16}$$

Discrete case: $(x_\alpha = \pi\alpha/N)$

$$\sum_{j=1}^{N-1} b_j \sin jx, \qquad b_j = \frac{2}{N}\sum_{\alpha=1}^{N-1} f(x_\alpha)\sin jx_\alpha. \tag{4.6.17}$$

If $f(0) \neq 0$ or $f(\pi) \neq 0$, one can still use such an expansion, but it will converge slowly.

If the function is defined in $[0, \pi]$, and if $f'(0) = f'(\pi) = 0$, then one can make a continuation of f into an *even* function on $[-\pi, \pi]$. Outside $[-\pi, \pi]$ the function is continued periodically. Its Fourier series then becomes a pure cosine series:

Continuous case:

$$\frac{1}{2}a_0 + \sum_{j=1}^{\infty} a_j \cos jx, \qquad a_j = \frac{2}{\pi}\int_0^\pi f(x)\cos jx\, dx, \tag{4.6.18}$$

Discrete case: $(x_\alpha = \pi\alpha/N$

$$\frac{1}{2}a_0 + \sum_{j=1}^{N-1} a_j \cos jx, \qquad a_j = \frac{2}{N}\sum_{\alpha=0}^{N-1} f(x_\alpha)\cos jx_\alpha. \tag{4.6.19}$$

4.6.3 The Fourier Integral Theorem

In Sec. 4.6.2 we showed how Fourier methods can be used on a nonperiodic function defined on a finite interval. Suppose now that the function $f(x)$ is defined on the entire real axis, and that it satisfies the regularity properties which we required in Theorem 4.6.2. Set

$$\varphi(\xi) = f(x), \quad \xi = 2\pi x/L \in [-\pi, \pi],$$

and continue $\varphi(\xi)$ outside $[-\pi, \pi]$ so that it has period 2π. By Theorem 4.6.2, if

$$c_j = \frac{1}{2\pi}\int_{-\pi}^{\pi} \varphi(\xi)e^{-ij\xi}\, d\xi = \frac{1}{L}\int_{-L/2}^{L/2} f(x)e^{-2\pi ixj/L}\, dx, \tag{4.6.20}$$

then $\varphi(\xi) = \sum_{j=-\infty}^{\infty} c_j e^{ij\xi}$, $\xi \in (-\pi, \pi)$, and hence

$$f(x) = \sum_{j=-\infty}^{\infty} c_j e^{2\pi ixj/L}, \quad x \in (-L/2, L/2).$$

If we set

$$g_L(\omega) = \int_{-L/2}^{L/2} f(x)e^{-2\pi ix\omega}\, dx, \quad \omega = j/L, \tag{4.6.21}$$

then by (4.6.20) we have $c_j = (1/L)g_L(\omega)$, and hence

$$f(x) = \frac{1}{L}\sum_{j=-\infty}^{\infty} g_L(\omega)e^{2\pi ix\omega}, \quad x \in (-L/2, L/2). \tag{4.6.22}$$

Now by passing to the limit $L \to \infty$, one avoids making an artificial periodic continuation outside a finite interval. The sum in (4.6.22) is a "sum of rectangles" similar to the sum which appears in the definition of a definite integral. However, here the argument varies from $-\infty$ to $+\infty$, and the function $g_L(t)$ depends on L. By a somewhat dubious passage to the limit, then, the pair of formulas of (4.6.21) and (4.6.22) becomes the pair

$$g(\omega) = \int_{-\infty}^{\infty} f(x)e^{-2\pi ix\omega}\,dx \iff f(x) = \int_{-\infty}^{\infty} g(\omega)e^{2\pi ix\omega}\,d\omega. \tag{4.6.23}$$

One can, in fact, after a rather complicated analysis, show that the above result is correct; see, e.g., Courant–Hilbert [16]. The proof requires, besides the previously mentioned "local" regularity conditions on f, the "global" assumption that

$$\int_{-\infty}^{\infty} |f(x)|\,dx$$

is convergent. The beautiful, almost symmetric relation of (4.6.23) is called the **Fourier integral theorem**. This theorem, and other versions of it, with varying assumptions under which they are valid, is one of the most important aids in both pure and applied mathematics. The function g is called the **Fourier transform**[41] of f.

Clearly the Fourier transform is a *linear operator*. Two other elementary properties that can easily be verified are:

$$f(ax) \iff \frac{1}{|a|}g(\omega/a), \tag{4.6.24}$$

$$\frac{1}{|b|}f(x/b) \iff g(b\omega). \tag{4.6.25}$$

If the function $f(x)$ has even or odd symmetry and is real or pure imaginary this leads to relations between $g(\omega)$ and $g(-\omega)$ that can be used to increase computational efficiency. Some of these properties are summarized in the table below.

Example 4.6.2.

The function $f(x) = e^{-|x|}$ has Fourier transform

$$\begin{aligned} g(\omega) &= \int_{-\infty}^{\infty} e^{-|x|}e^{-2\pi ix\omega}\,dx = \int_0^{\infty} \left(e^{-(1+2\pi i\omega)x} + e^{-(1-2\pi i\omega)x}\right) dx \\ &= \frac{1}{1+2\pi i\omega} + \frac{1}{1-2\pi i\omega} = \frac{2}{1+4\pi^2\omega^2}. \end{aligned}$$

Here $f(x)$ is real and an even function. In agreement with the table above the Fourier transform is also real and even.

[41] The terminology in the literature varies somewhat as to the placement of the factor 2π—e.g., it can be taken out of the exponent by a simple change of variable.

Function	Fourier transform
$f(x)$ real	$g(-\omega) = \overline{g}(\omega)$
$f(x)$ imaginary	$g(-\omega) = -\overline{g}(\omega)$
$f(x)$ even	$g(-\omega) = g(\omega)$
$f(x)$ odd	$g(-\omega) = -g(\omega)$
$f(x)$ real even	$g(\omega)$ real even
$f(x)$ imaginary odd	$g(\omega)$ real odd

Table 4.6.1. *Useful symmetry properties of Fourier transforms.*

From (4.6.23) it follows that

$$e^{-|x|} = \int_{-\infty}^{\infty} \frac{2}{1+4\pi^2\omega^2} e^{2\pi i x\omega}\, d\omega = \frac{2}{\pi}\int_0^{\infty} \frac{1}{1+x^2}\cos \pi x\, dx, \quad (2\pi\omega = x).$$

It is not so easy to prove this formula directly.

Many applications of the Fourier transform involve the use of convolutions.

Definition 4.6.6.
The **convolution** *of f_1 and f_2 is the function*

$$h(\xi) = conv\,(f_1, f_2) = \int_{-\infty}^{\infty} f_1(x) f_2(\xi - x)\, dx. \tag{4.6.26}$$

It is not difficult to verify that $conv\,(f_1, f_2) = conv\,(f_2, f_1)$. The following theorem states that the convolution of f_1 and f_2 can be computed as the inverse Fourier transform of the product $g_1(\omega)g_2(\omega)$. This fact is of great importance in the application of Fourier analysis, e.g., to differential equations and probability theory.

Theorem 4.6.7.
Let f_1 and f_2 have Fourier transforms g_1 and g_2, respectively. Then the Fourier transform g of the convolution of f_1 and f_2, is the product $g(\omega) = g_1(\omega)g_2(\omega)$.

Proof. By definition the Fourier transform of the convolution is

$$\begin{aligned} g(\omega) &= \int_{-\infty}^{\infty} e^{-2\pi i\xi\omega} \left(\int_{-\infty}^{\infty} f_1(x) f_2(\xi - x)\, dx \right) d\xi \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-2\pi i(x+\xi-x)\omega} f_1(x) f_2(\xi - x)\, dx\, d\xi \\ &= \int_{-\infty}^{\infty} e^{-2\pi i x\omega} f_1(x)\, dx \int_{-\infty}^{\infty} e^{-2\pi i(\xi-x)\omega} f_2(\xi - x)\, d\xi \\ &= \int_{-\infty}^{\infty} e^{-2\pi i x\omega} f_1(\xi)\, dx \int_{-\infty}^{\infty} e^{-2\pi i x\omega} f_2(x)\, dx = g_1(\omega)g_2(\omega) \end{aligned}$$

The legitimacy of changing the order of integration is here taken for granted. □

In many physical applications, the following relation, analogous to Parseval's identity (corollary to Theorem 4.5.13), is of great importance. If g is the Fourier transform of f, then

$$\int_{-\infty}^{\infty} |g(\omega)|\, d\omega = \int_{-\infty}^{\infty} |f(\xi)|\, d\xi. \tag{4.6.27}$$

In signal processing this can be interpreted to mean that the total power in a signal is the same whether computed in the time domain or the frequency domain.

4.6.4 Sampled Data and Aliasing

Consider a function $f(x)$ which is zero outside the interval $[0, L]$. The Fourier transform of $f(x)$ is then given by

$$g(\omega) = \int_0^L f(x) e^{-2\pi i \omega x}\, dx. \tag{4.6.28}$$

We want to approximate $g(\omega)$ using values of $f(x)$ sampled at intervals Δx,

$$f_j = f(j\Delta x), \quad 0 < j < N-1, \quad L = N\Delta x.$$

The integral (4.6.28) can be approximated by

$$g(\omega) \approx \frac{L}{N} \sum_{j=0}^{N-1} f_j e^{-2\pi i \omega j \Delta x}\, dx. \tag{4.6.29}$$

Since only N values of f_j are used as input and we want the computed values to be linearly independent, we cannot approximate $g(\omega)$ at more than N points. The wave of lowest frequency associated with the interval $[0, L]$ is $\omega = 1/L = 1/(N\Delta x)$, since then $[0, L]$ corresponds to one full period of the wave. We therefore choose in the frequency space points $\omega_k = k\Delta\omega$, $i = 0 : N$, such that the following **reciprocity relations** hold:

$$LW = N, \qquad \Delta x \Delta\omega = 1/N \tag{4.6.30}$$

With this choice it holds that

$$W = N\Delta\omega = 1/\Delta x, \qquad L = N\Delta x = 1/\Delta\omega. \tag{4.6.31}$$

Noting that $(j\Delta x)(k\Delta\omega) = jk/N$ we get from the trapezoidal approximation

$$g(\omega_k) \approx \frac{L}{N} \sum_{j=0}^{N-1} f_j e^{-2\pi i k j/N}\, dx = L c_k, \quad k = 0 : N-1,$$

where c_k is the coefficient of the discrete Fourier transform.

The frequency $\omega_c = 1/(2\Delta x) = \Delta\omega/2$ is the so called **Nyquist critical frequency**. Sampling the wave $\sin(2\pi\omega_c x)$ with sampling interval Δx will sample

exactly *two points per cycle*. It is a remarkable fact that if a function $f(x)$, defined on $[-\infty, \infty]$, is *band-width limited* to frequencies smaller or equal to ω_c, then $f(x)$ is completely determined by its sample values $j\Delta x$, $\infty \leq j \leq \infty$; see the Sampling Theorem Sec. 4.8.3.

If the function is not band-width limited the spectral density outside the critical frequency is moved into that range. This is called **aliasing**. The relationship between the Fourier transform $g(\omega)$ and the discrete Fourier transform of a finite sampled representation can be characterized as follows. Assuming that the reciprocity relations (4.6.30) are satisfied, the discrete Fourier transform of $f_j = \tilde{f}(j\Delta x)$, $0 \leq j < N$, will approximate the periodic aliased function

$$\tilde{g}_k = \tilde{g}(k\Delta\omega), \quad 0 \leq j < N. \tag{4.6.32}$$

where

$$\tilde{g}(\omega) = g(\omega) + \sum_{k=1}^{\infty} \left(g(\omega + kW) + g(\omega - kW) \right), \quad \omega \in [0, W] \tag{4.6.33}$$

Since by (4.6.31) $W = 1/\Delta x$, we can *increase* the frequency range $[0, W]$ covered by *decreasing* Δx.

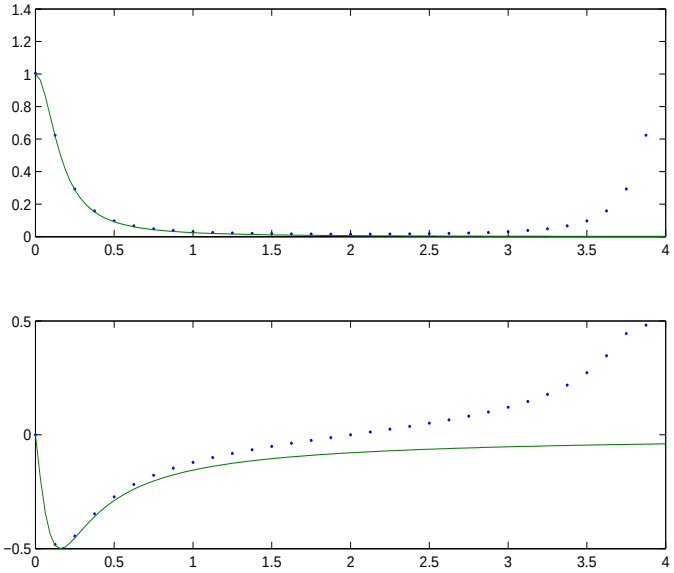


Figure 4.6.4. *The real (top) and imaginary (bottom) parts of the Fourier transform of e^{-x} and the corresponding DFT with $N = 32$, $T = 8$.*

Example 4.6.3.

The function $f(x) = e^{-x}$, $x > 0$, $f(x) = 0$, $x < 0$, has Fourier transform

$$g(\omega) = 1/(1 + 2\pi i\omega) = \frac{1 - i2\pi\omega}{1 + 4\pi^2\omega^2}$$

(cf. Example 4.6.2. Set $f(0) = 1/2$, the average of $f(-0)$ and $f(+0)$, which is the value given by the inverse Fourier transform at a discontinuity.

Set $N = 32$, $T = 8$, and sample the f in the interval $[0, T]$, at equidistant points $j\Delta x$, $j = 0 : N - 1$. Note that T is so large that the aliased function (4.6.32) is nearly equal to f. This sampling rate corresponds to $\Delta x = 8/32 = 1/4$ and $W = 4$.

The effect of aliasing in the frequency domain is evident. The error is significant for frequencies larger than the critical frequency $W/2$. To increase the accuracy W can be increased by decreasing the sampling interval Δx.

Review Questions

1. Derive the orthogonality properties and coefficient formulas which are fundamental to Fourier analysis, for both the continuous and the discrete case.
2. Under what conditions does the Fourier series of the function f converge to f in the continuous case?
3. How does one compute a Fourier expansion of a function of two variables?
4. Explain what a periodic continuation of a function is. What disadvantage for Fourier analysis is incurred if the periodic continuation has a discontinuity in its function value or derivatives at certain points?
5. Formulate the Fourier integral theorem.
6. (a) Explain what a periodic continuation of a function is.
   (b) What disadvantage (for Fourier analysis) is incurred if the periodic continuation has a discontinuity—e.g., in its derivative at certain points?

Problems and Computer Exercises

1. Give a simple characterization of the functions which have a sine expansion containing odd terms only.
2. Let f be an even function, with period 2π, such that

$$f(x) = \pi - x, \quad 0 \le x \le \pi.$$

   (a) Plot the function $y = f(x)$ for $-3\pi \le x \le 3\pi$. Expand f in a Fourier series.
   (b) Use this series to show that $1 + 3^{-2}] + 5^{-2}] + 7^{-2}] + \cdots = \pi^2/8$.
   (c) Compute the sum $1 + 2^{-2}] + 3^{-2}] + 4^{-2}] + 5^{-2}] + \cdots$.

(d) Compute, using (4.6.10), the sum $1 + 3^{-4}] + 5^{-4}] + 7^{-4}] + \cdots$.

(e) Differentiate the Fourier series term by term, and compare with the result in Example 4.6.1.

3. Show that the function $G_1(t) = t - 1/2$, $0 < t < 1$, has the expansion

$$G_1(t) = -\sum_{n=1}^{\infty} \frac{\sin 2n\pi t}{n\pi}.$$

Derive by term-wise integration, the expansion for the functions $G_p(t)$, and show the statement (made in Sec. 10.3.1) that $c_p - G_p(t)$ has the same sign as c_p. Show also that $\sum_{n=1}^{\infty} n^{-p} = \frac{1}{2}|c_p|(2\pi)^p$, p even.

4. (a) Prove that

$$\sum_{k=1}^{N-1} \sin \frac{\pi k}{N} = \cot \frac{\pi}{2N}.$$

Hint: $\sin x$ is the imaginary part of e^{ix}.

(b) Determine a sine polynomial $\sum_{j=1}^{n-1} b_j \sin jx$, which takes on the value 1 at the points $x_\alpha = \pi\alpha/n$, $\alpha = 1 : n - 1$.

Hint: Use (4.6.16) or recall that the sine polynomial is an odd function.

(c) Compare the limiting value for b_j as $n \to \infty$ with the result in Example 9.6.2.

5. (a) Prove the inequality in (4.6.9)!

(b) Show, under the assumptions on f which hold in (4.6.9), that, for $k \geq 1$, f can be approximated by a trigonometric polynomial such that

$$\Big\| f - \sum_{j=-n}^{n} c_j e^{ijx} \Big\|_\infty < \frac{2}{kn^k} \|f^{(k+1)}\|_\infty.$$

In the following problems, we do not require any investigation of whether it is permissible to change the order of summations, integrations, differentiations, etc.; it is sufficient to treat the problems in a purely formal way.

6. The partial differential equation $\dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2}$ is called the heat equation. Show that the function

$$u(x,t) = \frac{4}{\pi} \sum_{k=0}^{\infty} \frac{\sin(2k+1)x}{2k+1}\, e^{-(2k+1)^2 t},$$

satisfies the differential equation for $t > 0$, $0 < x < \pi$, with boundary conditions $u(0,t) = u(\pi,t) = 0$ for $t > 0$, and initial condition $u(x,0) = 1$ for $0 < x < \pi$ (see Example 4.6.1).

7. Show that if $g(t)$ is the Fourier transform of $f(x)$, then

(a) $e^{2\pi\alpha t}]g(t)$ is the Fourier transform of $f(x + \alpha)$.

(b) $(2\pi it)^k g(t)$ is the Fourier transform of $f^{(k)}](x)$, assuming that $f(x)$ and its derivatives up to the kth order tend to zero, as $x \to \infty$.

8. The **correlation** of $f_1(x)$ and $f_2(x)$ is defined by

$$c(\xi) = \int_{-\infty}^{\infty} \phi_1(x+\xi)\phi_2(x)\,dx. \tag{4.6.34}$$

Show that if $f_1(x)$ and $f_2(x)$ have Fourier transforms $g_1(t)$ and $g_2(t)$, respectively, then the Fourier transform of $c(\xi)$ is $h(t) = g_1(t)g_2(-t)$.

Hint: Compare Theorem 4.6.7.

9. Derive Parseval's identities (4.6.27) and (4.6.10)

4.7 The Fast Fourier Transform

4.7.1 The Fast Fourier Algorithm

Consider the problem to compute the discrete Fourier coefficients $\{c_j\}_{j=0}^{N-1}$

$$f(x) = \sum_{j=0}^{N-1} c_j e^{ijx},$$

for a function, whose values f_α are known at the points $x_\alpha = 2\pi\alpha/N$, $\alpha = 0 : N-1$. According to Theorem 4.6.5

$$c_j = \frac{1}{N}\sum_{\alpha=0}^{N-1} f_\alpha e^{-ijx_\alpha}, \quad j = 0 : N-1.$$

Setting $\omega_N = e^{-2\pi i/N}$ (i.e. ω is an Nth root of unity, $\omega^N = 1$), we can rewrite the problem as follows: compute

$$c_j = \frac{1}{N}\sum_{\alpha=0}^{N-1} \omega_N^{j\alpha} f_\alpha, \quad j = 0 : N-1. \tag{4.7.1}$$

It seems from (4.7.1) that to compute the discrete Fourier coefficients would require N^2 complex multiplications and additions. As we shall see, with the **Fast Fourier Transform** (**FFT**) one needs only about $N \log_2 N$ complex multiplications and additions if $N = 2^k$. For example, when $N = 2^{20} = 1\,048\,576$ the FFT algorithm is theoretically a factor of 84 000 faster than the "conventional" $O(N^2)$ algorithm. On a 266 MHz Pentium laptop, a real FFT of this size takes about 1.2 seconds using MATLAB 6, whereas 28 hours would be required by the conventional algorithm! The FFT not only uses fewer operations to evaluate the DFT, it also is more accurate. Whereas using the conventional method the roundoff error is proportional to N, for the FFT algorithm it is proportional to $\log_2 N$.

In many areas of application (digital signal and image processing, time-series analysis, to name a few) the FFT has caused a complete change of attitude toward what can be done using discrete Fourier methods. Without the FFT many modern devices like cell phones, digital cameras, CAT scans and DVDs would not be

possible. Some future applications considered in astronomy are expected to require FFTs of several gigapoints

Similar ideas were used already by Gauss and several other mathematicians, e.g., Danielson and Lanczos [18].[42].

In the following we will use the common convention *not* to scale the sum in (4.7.1) by $1/N$.

Definition 4.7.1.

The **Discrete Fourier Transform (DFT)** *of the vector* $f \in \mathbf{C}^N$ *is*

$$y = F_N f. \tag{4.7.2}$$

where $F_n \in \mathbf{R}^{N\times N}$ *is the DFT matrix with elements*

$$(F_N)_{j\alpha} = \omega_N^{j\alpha}, \quad j, \alpha = 0 : N-1, \tag{4.7.3}$$

where $\omega_N = e^{-2\pi i/N}$.

From the definition it follows that the Fourier matrix F_N is a complex Vandermonde matrix. Since $\omega_N^{j\alpha} = \omega_N^{\alpha j}$, F_N is symmetric. By Theorem 4.6.5

$$\frac{1}{N} F_N^* F_N = I,$$

where F_N^* is the complex conjugate transpose of F_N. Hence the inverse transform can be written

$$f = \frac{1}{N} F_N^* y.$$

Example 4.7.1.

For $n = 2^2 = 4$, the DFT matrix is

$$F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4 & \omega_4^2 & \omega_4^3 \\ 1 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ 1 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}. \tag{4.7.4}$$

where $\omega_4 = e^{-2\pi i/4}$, and $\omega_4^4 = 1$.

We now describe the central idea of the FFT algorithm. Let $N = 2^k$ and set

$$\alpha = \begin{cases} 2\alpha_1, & \text{if } \alpha \text{ is even,} \\ 2\alpha_1 + 1 & \text{if } \alpha \text{ is odd,} \end{cases}, \quad 0 \le \alpha_1 \le m-1.$$

[42]The modern usage of FFT started in 1965 with the publication of the [15] by James W. Cooley of IBM Research and John W. Tukey, Princeton University. Tukey came up with the basic algorithm at a meeting of President Kennedy's Science Advisory Committee. One problem discussed at this meeting was that the ratification of a US–Sovjet Union nuclear test ban depended on a fast method to detect nuclear tests by analyzing seismological time-series data.

where $m = N/2 = 2^{k-1}$. Split the DFT sum in an even and an odd part

$$y_j = \sum_{\alpha_1=0}^{m-1} (\omega_N^2)^{j\alpha_1} f_{2\alpha_1} + \omega_N^j \sum_{\alpha_1=0}^{m-1} (\omega_N^2)^{j\alpha_1} f_{2\alpha_1+1}, \quad j = 0 : N-1,$$

Let β be the quotient and j_1 the remainder when j is divided by m, i.e. $j = \beta m + j_1$. Then, since $\omega_N^N = 1$,

$$(\omega_N^2)^{j\alpha_1} = (\omega_N^2)^{\beta m \alpha_1} (\omega_N^2)^{j_1\alpha_1} = (\omega_N^N)^{\beta\alpha_1} (\omega_N^2)^{j_1\alpha_1} = \omega_m^{j_1\alpha_1}.$$

Thus if, for $j_1 = 0 : m-1$, we set

$$\phi_{j_1} = \sum_{\alpha_1=0}^{m-1} f_{2\alpha_1} \omega_m^{j_1\alpha_1}, \qquad \psi_{j_1} = \sum_{\alpha_1=0}^{m-1} f_{2\alpha_1+1} \omega_m^{j_1\alpha_1}. \tag{4.7.5}$$

then, $y_j = \phi_{j_1} + \omega_N^j \psi_{j_1}$, The two sums on the right are elements of the DFTs of length $N/2$ applied to the parts of f with odd and even subscripts. *The entire DFT of length N is obtained by combining these two DFTs!* Since $\omega_N^m = -1$ we have

$$y_{j_1} = \phi_{j_1} + \omega_N^{j_1} \psi_{j_1}, \tag{4.7.6}$$

$$y_{j_1+N/2} = \phi_{j_1} - \omega_N^{j_1} \psi_{j_1}, \quad j_1 = 0 : N/2 - 1. \tag{4.7.7}$$

These expressions, noted already by Danielson and Lanczos [18], are often called **butterfly relations** because of the data flow pattern. Note that these can be performed in place, i.e. no extra vector storage is needed.

The computation of ϕ_{j_1} and ψ_{j_1} means that one does *two* Fourier transforms with $m = N/2$ terms instead of one with N terms. If $N/2$ is even the same idea can be applied to these two Fourier transforms. One then gets *four* Fourier transforms, each of with has $N/4$ terms; If $N = 2^k$ this reduction can be continued recursively until we get N DFTs with 1 term. But $F_1 = I$, the identity.

The number of complex operations required to compute $\{y_j\}$ from the butterfly relations when $\{\phi_{j_1}\}$ and $\{\psi_{j_1}\}$ have been computed is 2^k, assuming that the powers of ω are precomputed and stored. Thus, if we denote by p_k the total number of operations needed to compute the DFT when $N = 2^k$, we have

$$p_k \le 2p_{k-1} + 2^k, \quad k \ge 1.$$

Since $p_0 = 0$, it follows by induction that $p_k \le k \cdot 2^k = N \cdot \log_2 N$. *Hence, when N is a power of two, the fast Fourier transform solves the problem with at most $N \cdot \log_2 N$ operations.* The FFT is an example of the general technique of divide-and-conquer algorithms (see Sec. 1.3.2). For a recursive implementation of the FFT algorithm, see Problem 11.

Example 4.7.2.

Let $N = 2^4 = 16$. Then the 16-point DFT (0:1:15) can be split into two 8-points DFTs (0:2:14) and (1:2:15), which each can be split in two 4-point DFTs.

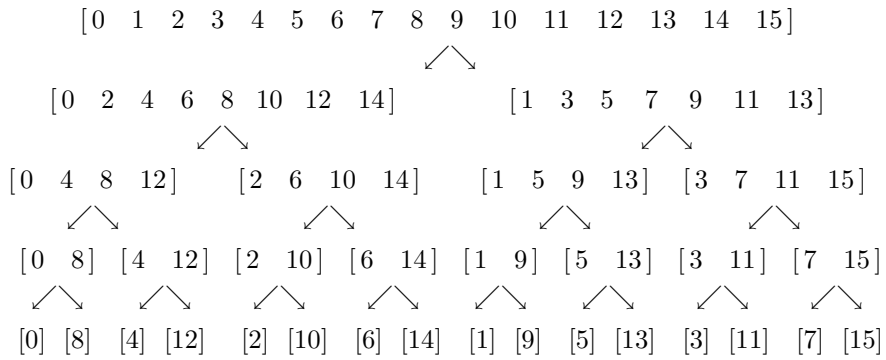


Figure 4.7.1. *The structure of an* $2^4 = 16$*-point FFT.*

Repeating these splittings we finally get 16 one-point DFTs, which are the identity $F_1 = 1$; see Figure 4.6.1.

In most implementations the explicit recursion is avoided. Instead the FFT algorithm is implemented in two stages:

- a reordering stage in which the data vector f is permuted;
- a second stage in which first $N/2$ FFT transforms of length 2 are computed on adjacent elements, next $N/4$ transforms of length 4, etc, until the final result is obtained by merging two FFTs of length $N/2$.

We now consider each stage in turn.

Each step of the recursion involves an even-odd permutations. In the first step the points with last digit equal to 0 are ordered first and those with last digit equal to 1 last. In the next step the two resulting subsequences of length $N/2$ are reordered according to the second binary digit, etc. It is not difficult to see that the combined effect of the reorderings in stage 1 is a **bit-reversal permutation** of the data points. For $i = 0 : N - 1$, let the index i have the binary expansion

$$i = b_0 + b_1 \cdot 2 + \cdots + b_{t-1} \cdot 2^{t-1}$$

and set

$$r(i) = b_{t-1} + \cdots + b_1 \cdot 2^{t-2} + b_0 \cdot 2^{t-1}.$$

That is, $r(i)$ is the index obtained by reversing the order of the binary digits. If $i < r(i)$ then exchange f_i and $f_{r(i)}$. This reordering is illustrated for $N = 16$ in Figure 4.7.1.

We denote the permutation matrix corresponding to bit-reversal ordering by P_N. Note that if an index is reversed twice we end up with the original index. This means that $P_N^{-1} = P_N^T = P_N$, that is P_N is symmetric. The permutation can be carried out "in place" by a sequence of pairwise interchanges or transpositions of the data points. For example, for $N = 16$ the pairs (1,8), (2,4), (3,12), (5,10), (7,14)

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

$\Longrightarrow$

Decimal	Binary
0	0000
8	1000
4	0100
12	1100
2	0010
10	1010
6	0110
14	1110
1	0001
9	1001
5	0101
13	1101
3	0011
11	1011
7	0111
15	1111

Figure 4.7.2. *Bit-reversal ordering. The original order left and the bit-reversal order right.*

and (11,13) are interchanged. The bit-reversal permutation can take a substantial fraction of the total time to do the FFT. Which implementation is best depends strongly on the computer architecture.

We now consider the second stage of the FFT. The key observation to develop a matrix-oriented description of this stage is to note that the Fourier matrices F_N after an odd-even permutation of the columns can be expressed as a 2×2 block matrix, where each block is either $F_{N/2}$ or a diagonal scaling of $F_{N/2}$.

Theorem 4.7.2. Van Loan [56, Theorem 1.2.1]

Let Π_N^T be the permutation matrix, which applied to a vector groups the even-indexed components first and the odd-indexed last.[43] *If $N = 2m$ then*

$$F_N \Pi_N = \begin{pmatrix} F_m & \Omega_m F_m \\ F_m & -\Omega_m F_m \end{pmatrix} = \begin{pmatrix} I_m & \Omega_m \\ I_m & -\Omega_m \end{pmatrix} \begin{pmatrix} F_m & 0 \\ 0 & F_m \end{pmatrix},$$

$$\Omega_m = \mathrm{d}iag\,(1, \omega_N, \ldots, \omega_N^{m-1}), \quad \omega_N = e^{-2\pi i/N}. \tag{4.7.8}$$

Proof. The proof essentially follows from the derivation of the butterfly relations (4.7.6)–(4.7.7). □

[43] Note that $\Pi_N^T = \Pi_N^{-1}$ is the so called **perfect shuffle permutation**. In this the permuted vector $\Pi_N^T f$ is obtained by splitting f in half and then "shuffling" the top and bottom halves.

Example 4.7.3.

We illustrate Theorem 4.7.2 for $N = 2^2 = 4$. The DFT matrix F_4 is given in Example 4.7.3. After a permutation of the columns F_4 can be written as a 2×2 block-matrix

$$F_4\Pi_4^T = \left(\begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 1 & -1 & -i & i \\ \hline 1 & 1 & -1 & -1 \\ 1 & -1 & i & -i \end{array}\right) = \begin{pmatrix} F_2 & \Omega_2 F_2 \\ F_2 & -\Omega_2 F_2 \end{pmatrix},$$

where

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad \Omega_2 = \mathrm{d}iag\,(1, -i).$$

When $N = 2^k$ the FFT algorithm can be interpreted as a sparse factorization of the DFT matrix

$$F_N = A_k \cdots A_2 P_N, \tag{4.7.9}$$

where P_N is the bit-reversal permutation matrix and

$$A_q = \mathrm{d}iag\,(\underbrace{B_L, \ldots, B_L}_{r}), \quad L = 2^q, \quad r = n/L. \tag{4.7.10}$$

Here $B_k \in \mathbf{C}^{L\times L}$ is the radix-2 butterfly matrix defined by

$$B_L = \begin{pmatrix} I_{L/2} & \Omega_{L/2} \\ I_{L/2} & -\Omega_{L/2} \end{pmatrix}, \tag{4.7.11}$$

$$\Omega_{L/2} = \mathrm{d}iag\,(1, \omega_L, \ldots, \omega_L^{L/2-1}), \quad \omega_L = e^{-2\pi i/L}. \tag{4.7.12}$$

The FFT algorithm described above is usually referred to as the Cooley–Tukey FFT algorithm. Using the fact that both P_N and the DFT matrix F_n is symmetric, we obtain by transposing (4.7.9) the factorization

$$F_N = F_N^T = P_N A_1^T A_2^T \cdots A_k^T. \tag{4.7.13}$$

This gives rise to a "dual" FFT algorithm, referred to as the Gentleman–Sande algorithm [27]. In this the bit-reversal permutation comes *after* the other computations. In many important applications such as convolution and the solution of discrete Poisson equation, this permits the design of in-place FFT solutions that avoid bit-reversal altogether.

In the operation count for the FFT above we assumed that the weights ω_L^j, $j = 1 : L-1$, $\omega_L = e^{-2\pi i/L}$ are precomputed. To do this one could use at

$$\omega_L^j = \cos(j\theta) - i\sin(j\theta), \quad \theta = 2\pi/L.$$

for $L = 2^q$, $q = 2 : k$. This is accurate, but expensive, since it involves $L-1$ trigonometric functions calls. An alternative is to compute $\omega = \cos(\theta) - i\sin(\theta)$ and use repeated multiplication,

$$\omega^j = \omega\omega^{j-1}, \quad j = 2 : L-1.$$

This replaces one sine/cosine call with a single complex multiplication, but has the drawback that accumulation of roundoff errors will give an error in ω_L^j of order ju.

4.7.2 FFTs and Discrete Convolutions

Many applications of the FFT involve the use of a discrete version of the convolution,

Definition 4.7.3.
Given two sequences f_i and g_i, $i = 0 : N-1$. Then the convolution of f and g is the sequence defined by

$$h_k = conv\,(f,g) = \sum_{i=0}^{N-1} f_i g_{k-i}, \quad i = 0 : N-1, \tag{4.7.14}$$

where the sequences are extended to have period N, by setting $f_i = f_{i+jN}$, $g_i = g_{i+jN}$, for all integers i, j.

The discrete convolution can be used to approximate the convolution defined for continuous functions in Definition 4.6.6 in a similar way as the Fourier transform was approximated Using sampled values in Sec. 4.6.4.

We can write the sum in (4.7.14) as a matrix-vector multiplication $h = Gf$, or writing out components

$$\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ \vdots \\ h_{N-1} \end{pmatrix} = \begin{pmatrix} g_0 & g_{N-1} & g_{N-2} & \cdots & g_1 \\ g_1 & g_0 & g_{N-1} & \cdots & g_2 \\ g_2 & g_1 & g_0 & \cdots & g_3 \\ \vdots & \vdots & \vdots & \cdots & \\ g_{N-1} & g_{N-2} & g_{N-3} & \cdots & g_0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix}.$$

Note that each column in G is a cyclic down-shifted version of the previous column. Such a matrix is called a **circulant matrix**. We have

$$G = [\, g \quad R_N g \quad R_N^2 g \quad \cdots \quad R_N^{N-1} g \,],$$

and R_N is a circulant permutation matrix. For example,

$$R_4 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

(see Problem 11).

Theorem 4.7.4.
Let f_i and g_i, $i = 0 : N-1$ be two sequences with DFTs equal to $F_N f$ and $F_N g$. Then the DFT of the **convolution** *of f and g, is $F_N f .* F_N g$, where $.*$ denotes elementwise product.*

Proof. The proof depends on the fact that the circulant matrix G is diagonalized by the DFT matrix F_N, i.e.

$$G = F_N^{-1} \mathrm{d}iag\,(F_N g) F_N;$$

see Problem 15. (Here d$iag\,(x)$, where x is a vector, denotes a diagonal matrix with diagonal elements equal to x.) It follows that

$$h = Gf = F_N^{-1}\mathrm{d}iag\,(F_N g)F_N f = F_N^{-1}((F_N g).*(F_N f)). \tag{4.7.15}$$

It follows from (4.7.15) that $conv\,(f,g) = conv\,(g,f)$. □

This shows that using the FFT algorithm the discrete convolution can be computed in order $N\log_2 N$ operations as follows. First the two FFTs of f and g are computed and multiplied (pointwise) together. Then the inverse DFT of this product is computed. This is one of the most useful properties of the FFT. Computing convolutions is of great importance in signal processing.

Using the Gentleman–Sande algorithm $F_N = P_N A^T$ for the forward DFT and the Cooley–Tukey algorithm for the inverse DFT, $F^{-1} = (1/N)\bar{F}_N = (1/N)\bar{A}P_N/N$, we get from (4.7.15)

$$f = \frac{1}{N}\bar{A}P_N((P_N A^T h).*(P_N A^T g)) = \frac{1}{N}\bar{A}((A^T h).*(A^T g)). \tag{4.7.16}$$

This shows that h can be computed without the bit-reversal permutation P_N, which typically can save 10–30 percent of the overall computation time.

4.7.3 Real Data and Fast Trigonometric Transforms

Frequently the FFT of a real data vector is required. The complex FFT algorithm can still be used, but is inefficient both in terms of storage and operations. Better alternatives can be found by using symmetries in the DFT, which correspond to the symmetries noted in the Fourier transform in Table 4.6.1.

We first show that the conjugate transpose of the DFT matrix F_N can be obtained by reversing the order of the last $N-1$ rows.

Lemma 4.7.5. Van Loan [56, Theorem 4.3.1]

Let T_N be the $N\times N$ permutation matrix which reverses the last $N-1$ elements. Then $\overline{F}_N = T_N F_N = F_N T_N$.

Proof. To verify that $\overline{F}_N = T_N F_N$, observe that

$$[T_N F_N]_{j\alpha} = \omega_N^{(N-j)\alpha} = \omega_N^{-j\alpha} = \overline{\omega}_N^{j\alpha} = [\overline{F}_N]_{j\alpha}, \quad 1 \le j \le N-1.$$

Since F_N and T_N are both symmetric, we also have $\overline{F}_N = (T_N F_N)^T = F_N T_N$. □

We say that a vector $x \in \mathbf{C}^N$ is **conjugate even** if $\bar{x} = T_N y$, and **conjugate odd** if $\bar{x} = T_N y$. Suppose now that f is real and $u = F_N f$. Then it follows that

$$\overline{u} = \overline{F}_N f = T_N F_N f = T_N u,$$

i.e. u is conjugate even. If a vector u of even length $N = 2m$ is conjugate even, this implies that

$$u_j = \bar{u}_{N-j}, \quad j = 1:m.$$

In particular u_j is real for $j = 0, m$.

For purely imaginary data g and $v = F_N g$, we have

$$\overline{v} = \overline{F}_N \overline{g} = -\overline{F}_N g = -T_N F_N g = -T_N v,$$

i.e. v is conjugate odd. Some other useful symmetry properties are given in the table below, where E_N denotes the $N \times N$ permutation matrix which reverses all elements in an N-vector.

Data f	Definition	DFT $F_N f$
real		conjugate even
imaginary		conjugate odd
real even	$f = T_n f$	real
real odd	$f = -T_n f$	imaginary
conjugate even	$\overline{f} = T_n f$	real
conjugate odd	$\overline{f} = -T_n f$	imaginary

Table 4.7.1. *Useful symmetry properties of DFTs.*

We now outline how symmetries can be used to compute the DFTs $u = F_N f$ and $v = F_N g$ of *two* real functions f and g simultaneously. First form the complex function $f + ig$ and compute its DFT

$$w = F_N(f + ig) = u + iv$$

by any complex FFT algorithm. Multiplying by T_N we have

$$T_N w = T_N F_N(f + ig) = T_N(u + iv) = \bar{u} + i\bar{v},$$

where we have used that u and v are conjugate even. Adding and subtracting these two equations we obtain

$$\begin{aligned} w + T_N w &= (u + \bar{u}) + i(v + \bar{v}), \\ w - T_N w &= (u - \bar{u}) + i(v - \bar{v}). \end{aligned}$$

We can now retrieve the two DFTs from

$$u = F_N f = \frac{1}{2}\big[\mathrm{Re}(w + T_N w) + i\,\mathrm{Im}(w - T_N w)\big], \tag{4.7.17}$$

$$v = F_N g = \frac{1}{2}\big[\mathrm{Im}(w + T_N w) - i\,\mathrm{Re}(w - T_N w)\big]. \tag{4.7.18}$$

Note that because of the conjugate even property of u and v there is no need to save the entire transforms.

The above scheme is convenient when, as for convolutions, two real transforms are involved. It can also be used to efficiently compute the DFT of a single real

function of length $N = 2^k$. First express this DFT as a combination of the two real FFTs of length $N/2$ corresponding to even and odd numbered data points (see as in (4.7.5)). Then apply the procedure above to simultaneously compute these two real FFTs.

Two more real transforms, the **discrete sine transform (DST)** and **discrete cosine transform (DCT)**, are of interest. These are defined as follows:

- Given real f_j, $j = 1 : m-1$ compute

$$y_k = \sum_{j=1}^{m-1} \sin(kj\pi/m)\, f_j \quad \text{(DST)}. \tag{4.7.19}$$

- Given real f_j, $j = 0 : m$ compute

$$y_k = \frac{1}{2}(f_0 + (-1)^k f_m) + \sum_{j=1}^{m-1} \sin(kj\pi/m)\, f_j \quad \text{(DCT)}. \tag{4.7.20}$$

These can be computed by applying the FFT algorithm (for real data) to an auxiliary vector formed by extending the given data f either into an odd or even sequence.

For the DST the f_j, $j = 1 : m-1$ is extended to an *odd* sequence of length $N = 2m$ by setting

$$f_0 = f_m = 0, \quad f_{2m-j} \equiv -f_j, \quad j = 1 : m-1.$$

For example, the data $\{f_1, f_2, f_3\}$, $(m = 2^2)$ is extended to

$$\tilde{f} = \{f_0, f_1, f_2, f_3, f_0, -f_3, -f_2, -f_1\}.$$

The extended vector satisfies $\tilde{f} = -T_N \tilde{f}$, and thus by Table 4.6.2 the DFT of $\tilde{f}$ will imaginary.

For the DCT the data f_j, $j = 0 : m$ is extended to an *even* sequence of length $N = 2m$ by setting

$$f_0 = f_m = 0, \quad f_{2m-j} \equiv f_j, \quad j = 1 : m-1.$$

For example, the data $\{f_0, f_1, f_2, f_3, f_4\}$, $(m = 2^2)$ is extended to

$$\tilde{f} = \{f_0, f_1, f_2, f_3, f_4, f_3, f_2, f_1\}.$$

so that $\tilde{f} = T_N \tilde{f}$. By Table 4.6.2 the DFT of $\tilde{f}$ will then be real.

Theorem 4.7.6. Van Loan [56, Sec. . 4.4]

Let f_j, $j = 1 : m-1$ form a real data vector f and extend it to a vector $\tilde{f}$ with $\tilde{f}_0 = \tilde{f}_m = 0$, so that $\tilde{f} = -T_N \tilde{f}$. Then $y(1 : m-1)$ is the DST of f, where

$$y = \frac{i}{2} F_{2m} \tilde{f}.$$

Let f_j, $j = 0 : m$ form a real data vector f and extend it to an vector $\tilde{f}$ so that $\tilde{f} = T_N \tilde{f}$. Then $y(0 : m)$ is the DST of f, where

$$y = \frac{1}{2} F_{2m} \tilde{f}.$$

There is an inefficiency factor of two in the above procedure. This can be eliminated by using a different auxiliary vector. For details we refer to [39, p. 420–421] and [56, Sec. 4.4.].

4.7.4 The General Case FFT

It can be argued ([39, p. 409]) that one should always choose $N = 2^k$ when using the FFT. If necessary the data can be padded with zeros to achieve this. To introduce an odd factor s, let $N = sr$, where r is a power of two. Then one can combine the power of two algorithm for the r-point subseries with a special algorithm for the s-point subseries. If s is a small number then one could generate the DFT matrix F_s and use matrix-vector multiplication; see Problem 10. However, the general case when if N is not a power of two, is at least of theoretical interest.

Suppose that $N = r_1 r_2 \cdots r_p$. We will describe an FFT algorithm which requires $N(r_1 + r_2 + \cdots + r_p)$ operations. Set

$$N_\nu = \prod_{i=\nu+1}^{p} r_i, \quad \nu = 0 : p-1, \quad N_p = 1.$$

Thus

$$N = r_1 r_2 \cdots r_\nu N_\nu, \quad N_0 = N.$$

The algorithm is based on two representations of integers, which are generalizations of the position principle (see Sec. 2.2.1).

I. Every integer j, $0 \le j \le N-1$ has a unique representation of the form

$$j = \alpha_1 N_1 + \alpha_2 N_2 + \cdots + \alpha_{p-1} N_{p-1} + \alpha_p, \quad 0 \le \alpha_i \le r_i - 1. \tag{4.7.21}$$

II. For every integer β, $0 \le \beta \le N-1$, β/N has a unique representation of the form

$$\frac{\beta}{N} = \frac{k_1}{N_0} + \frac{k_2}{N_1} + \cdots + \frac{k_p}{N_{p-1}}, \quad 0 \le k_i \le r_i - 1. \tag{4.7.22}$$

Set

$$j_\nu = \sum_{i=\nu+1}^{p} \alpha_i N_i, \qquad \frac{\alpha_\nu}{N_\nu} = \sum_{i=\nu}^{p-1} \frac{k_{i+1}}{N_i}, \quad (j_\nu < N_\nu). \tag{4.7.23}$$

As an exercise, the reader can verify that the coefficients in the above representations can be recursively determined from the following algorithms: [44]

$$j_0 = j, \qquad j_{i-1}/N_i = \alpha_i + j_i/N_i, \quad i = 1 : p;$$

[44] These algorithms can, in the special case that $r_i = B$ for all i, be used for converting integers or fractions to the number system whose base is B; see Algorithm 2.2.1.

$$\beta_0 = \beta, \qquad \beta_{i-1}/r_i = \beta_i + k_i/r_i, \quad i = 1:p.$$

From (4.7.21)–(4.7.23), it follows that, since N_i is divisible by N_ν for $i \le \nu$,

$$\frac{j\beta}{N} = \text{integer} + \sum_{\nu=0}^{p-1} \frac{k_{\nu+1}}{N_\nu}\Big(\sum_{i=\nu+1}^{p} \alpha_i N_i\Big) = \sum_{\nu=0}^{p-1} \frac{k_{\nu+1} j_\nu}{N_\nu} + \text{integer}.$$

From this, it follows that

$$\omega^{j\beta} = e^{2\pi i j\beta/N} = \prod_{\nu=0}^{p-1} e^{k_{\nu+1} j_\nu 2\pi i/N_\nu} = \prod_{\nu=0}^{p-1} \omega_\nu^{j_\nu k_{\nu+1}}, \tag{4.7.24}$$

where $\omega_\nu = e^{2\pi i/N_\nu}$, $\omega_0 = \omega$.

We now give an illustration of how the factorization in (4.7.24) can be utilized in fast Fourier transform for the case $p = 3$. Set, in accordance with (4.7.22),

$$f_\beta = c^{(0)}(k_1, k_2, k_3).$$

We have then

$$c_j = \sum_{\beta=0}^{N-1} f_\beta \omega^{j\beta} = \sum_{k_1=0}^{r_1-1} \sum_{k_2=0}^{r_2-1} \sum_{k_3=0}^{r_3-1} c^{(0)}(k_1, k_2, k_3) \omega_2^{j_2 k_3} \omega_1^{j_1 k_2} \omega^{j k_1}.$$

One can thus compute successively (see (4.7.23))

$$c^{(1)}(k_1, k_2, \alpha_3) = \sum_{k_3=0}^{r_3-1} c^{(0)}(k_1, k_2, k_3) \omega_2^{j_2 k_3} \quad (j_2 \text{ depends only on } \alpha_3),$$

$$c^{(2)}(k_1, \alpha_2, \alpha_3) = \sum_{k_2=0}^{r_2-1} c^{(1)}(k_1, k_2, \alpha_3) \omega_1^{j_1 k_2} \quad (j_1 \text{ depends only on } \alpha_2, \alpha_3),$$

$$c_j = c^{(3)}(\alpha_1, \alpha_2, \alpha_3) = \sum_{k_1=0}^{r_1-1} c^{(2)}(k_1, \alpha_2, \alpha_3) \omega^{j k_1} \quad (j \text{ depends on } \alpha_1, \alpha_2, \alpha_3).$$

The quantities $c^{(i)}$ are computed for all $r_1 r_2 r_3 = N$ combinations of the values of the arguments. Thus the total number of operations for the entire Fourier analysis becomes at most $N(r_3 + r_2 + r_1)$. The generalization to arbitrary p is obvious.

Review Questions

1. Suppose we want to compute the DFT for $N = 2^{10}$. Roughly how much faster is the FFT algorithm compared to the straightforward $O(N)$ algorithm?
2. Show that the matrix $U = \frac{1}{\sqrt{N}} F_N$ is unitary, i.e. $U^* U = I$, where $U^* = (\overline{U})^T$.

3. Show that the DFT matrix F_4 can be written as a 2×2 block matrix where each block is related to F_2. Give a generalization of this for F_N, $N = 2^m$, that holds for arbitrary m..

4. Work out on your own the bit-reversal permutation of the vector $[0 : N-1]$ for the case $N = 2^4 = 16$. How many exchanges need to be performed?

Problems and Computer Exercises

1. The following MATLAB script uses an algorithm due to Cooley et al. to permute the vector $x(1 : 2^m)$, in bit-reversal order:

```
n = 2^m;
nv2 = n/2; nm1 = n - 1;
j = 1;
for i = 1:nm1
   if i < j
      t = x(j); x(j) = x(i); x(i) = t;
   end
   k = nv2;
   while k < j
      j = j - k; k = k/2;
   end
   j = j + k;
end
```

Plot the time taking by this algorithm on your computer for $m = 5 : 10$. Does the execution time depend linearly on $N = 2^m$?

2. The following MATLAB program (C. Moler and S. Eddins [37]) demonstrates how the FFT idea can be implemented in a simple but efficient recursive MATLAB program. The program uses the fast recursion as long as n is a power of two. When it reaches an odd length it sets up the Fourier matrix and uses matrix vector multiplication.

```
function y = fftx(x);
% FFT computes the Fast Fourier Transform of x(1:n)
x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);
if rem(n,2) == 0
% Recursive divide and conquer
   k = (0:n/2-1)
   w = omega.^k;
   u = fftx(x(1:2:n-1));
   v = w.*fftx(x(2:2:n));
   y = [u+v; u-v];
```

```
else
% Generate the Fourier matrix
   j = 0:n-1;
   k = j';
   F = omega.^(k*j);
   y = F*x;
end
```

Apply this program to compute DFT of the function treated in Example 4.6.1 sampled at the points $2\pi\alpha/N$, $\alpha = 0 : N-1$. Choose, for instance, $N = 32, 64, 128$.

3. Write an efficient Matlab program for computing the DFT of a real data vector of length $N = 2^m$. As outlined in Sec. 4.7.3, first split the data in odd and even data points. Compute the corresponding DFTs using one call of the function fftx in Problem 10 with complex data of length $N/2$.

4. Verify the last four symmetry properties of DFTs in Table 4.6.2.

5. Let C be the square matrix

$$C_n = \begin{pmatrix} 0_{n-1}^T & 1 \\ I_{n-1} & 0_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & & & & 0 \\ & 1 & & & \vdots \\ & & \ddots & & 0 \\ & & & 1 & 0 \end{pmatrix} \in \mathbf{R}^{n\times n}.$$

Show that the effect of C_nA and AC_n, respectively, is a circular shift downwards of the rows and a shift to the left of the columns in $A \in \mathbf{R}^{n\times n}$. What about C^T?

6. A circulant matrix $A \in \mathbf{R}^{n\times n}$ generated by $(a_1, a_2, \ldots, a_{n-1}, a_n)$ has the form

$$A = \begin{pmatrix} a_0 & a_{n-1} & \cdots & a_2 & a_1 \\ a_1 & a_0 & \cdots & a_3 & a_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \cdots & a_0 & a_{n-1} \\ a_{n-1} & a_{n-2} & \cdots & a_1 & a_0 \end{pmatrix}.$$

(a) Show that $A = a_0I + a_1C + \ldots + a_{n-1}C^{n-1}$, where C is the circulant matrix in Problem 6.

(b) Show that the eigenvalues and eigenvectors of A are given by

$$\lambda_j = a_0 + a_1\omega_j + \ldots + a_{n-1}\omega_j^{n-1}, \quad x_j = \frac{1}{\sqrt{n}}(1, \omega_j, \ldots, \omega_j^{n-1})^T,$$

where $\omega_j = e^{2\pi j/n}$, $j = 1 : n$ are the n roots of unity $\omega^n = 1$.

(c) Show that the result in (b) implies that $C = F\Lambda F^H$, where

$$\Lambda = \mathrm{d}iag\,(\lambda_1, \ldots, \lambda_n), \qquad F = (x_1, \ldots, x_n)$$

is the matrix of the discrete Fourier transform, and the eigenvalues are given by the Fourier transform of its first column

$$F(a_0, a_{n-1}, \ldots, a_2, a_1)^T = (\lambda_1, \ldots, \lambda_n)^T.$$

4.8 Complex Analysis in Interpolation

In this section we make a more detailed theoretical and experimental study of interpolation of an analytic function $f(z)$ on a real interval. including an analysis of the Runge phenomenon (see Sec. 4.1.4). We then study interpolation at an infinite equidistant point set from the point of view of Complex Analysis. This interpolation problem, which was studied by Whittaker and others at the beginning of the century, became revived at the middle of the century under the name of the **Shannon sampling theorem**, with important applications to Communication Theory.

We shall encounter multi-valued functions: the logarithm and the square root. For each of these we choose that branch, which is positive for large positive values of the argument. They will appear in such contexts that we can then keep them non-ambiguous by forbidding z to pass the interval $[-1, 1]$. (We can, however, allow z to approach that interval.)

4.8.1 Interpolation of Analytic Functions

We first consider the general problem of polynomial interpolation of an analytic function, at an arbitrary sequence of points in $\mathbf{C}$. Multiple points are allowed. Set[45]

$$\Phi(z) = (z - u_1)(z - u_2) \cdots (z - u_n), \quad z, u_j \in \mathbf{C}.$$

Let $\mathcal{D}$ be a simply connected open domain in $\mathbf{C}$ that contains the point u and the nodes $u_1, u_2, \ldots, u_n$. We consider the interpolation problem to find the polynomial $p^* \in \mathcal{P}_n$ that is determined by the conditions $p^*(u_j) = f(u_j)$, $j = 1 : n$, or the appropriate Hermite interpolation problem in the case of multiple nodes. We know that p^* depends linearly on f, that is there exists a linear mapping L_n from some appropriate function space so that $p^* = L_n f$.

Assume that f is an analytic function in the closure of $\mathcal{D}$, perhaps except for a finite number of poles p. A pole must not be a node. Recall the elementary identity

$$\frac{1}{z-u} = \sum_{j=1}^{n} \frac{\Phi_{j-1}(u)}{\Phi_j(z)} + \frac{\Phi_n(u)}{\Phi_n(z)(z-u)}, \tag{4.8.1}$$

which is valid also for multiple nodes. Introduce the linear operator K_n,

$$(K_n f)(u) = \frac{1}{2\pi i} \int_{\partial\mathcal{D}} \frac{\Phi_n(u) f(z)}{\Phi_n(z)(z-u)}, \tag{4.8.2}$$

[45] We use the notation u, u_i instead of x, x_i here, since x is traditionally associated with the real part of a complex variable z.

multiply the above identity by $f(z)/(2\pi i)$, and integrate along the boundary of $\mathcal{D}$:

$$\frac{1}{2\pi i}\int_{\partial\mathcal{D}} \frac{f(z)}{z-u} = \sum_{j=1}^{n} \Phi_{j-1}(u)\frac{1}{2\pi i}\int_{\partial\mathcal{D}} \frac{f(z)}{\Phi_j(z)}\,dz + (K_n f)(u). \tag{4.8.3}$$

The following theorem is valid, when the interpolation points x_j are in the complex plane, although we shall here mainly apply it to the case, when they are located in the interval $[-1, 1]$.

Theorem 4.8.1.

Assume that $f(z)$ is analytic in a domain $\mathcal{D}$ that contains the points $x_1, x_2, \dots x_n$, as well as the point $u \in \mathbf{C}$. Let $L_n f$ be the solution of the interpolation problem $(L_n f)(x_j) = f(x_j)$, $j = 1 : n$. Then the interpolation error can be expressed as a complex integral, $f(u) - (L_n f)(u) = I_n(u)$, where

$$I_n(u) = \frac{1}{2\pi i}\int_{\partial\mathcal{D}} \frac{\Phi(u)f(z)}{\Phi(z)(z-u)}\,dz.$$

Proof. By the residue theorem,

$$I_n(u) = \sum_{j=1}^{n} \frac{\Phi(u)f(x_j)}{(u-x_j)\Phi'(x_j)} + f(u),$$

where the sum, with reversed sign, is Lagrange's form of the interpolation polynomial. □

(Note the relation between the Lagrange interpolation formula and the expansion of $f(z)/\Phi(z)$ into partial fractions, when, e.g., $f(z)$ is a polynomial.)

We now proceed to *Chebyshev interpolation*, i.e. interpolation at the the zeros of the Chebyshev polynomials. We shall show that it is almost as efficient as the truncation of a Chebyshev expansion. In this case, $\Phi(z) = 2^{1-n}T_n(z)$. Let $\mathcal{D} = E_R$, $x \in [-1, 1]$, $z \in \partial E_R$, where E_R is the ellipse

$$E_R = \{z : |z-1| + |z+1| \le R + R^{-1}\},$$

introduced in Sec. 3.5.1 (see (3.2.24). Consider the integral in Theorem 4.8.1 and assume that $|f(z)| \le M$ for $z \in \partial E_R$. It can be shown (Problem 2) that $|T_n(x)| \le 1$ and

$$|T_n(z)| \ge \tfrac{1}{2}(R^n - R^{-n}), \quad |z-x| \ge a-1, \quad \int_{\partial E_R} |dz| \le 2\pi a,$$

where a is the major semi-axis of E_R, i.e. $a = \frac{1}{2}(R + R^{-1})$. Then, by a straightforward calculation,

$$|f(x) - (L_n f)(x)| \le \frac{2MR^{-n}a}{(1-R^{-2n})(a-1)}. \tag{4.8.4}$$

This is somewhat less sharp than the result obtained by the Chebyshev expansion, in particular when $R \approx 1$. The details are left for Problem 2.

Note that $f(z)$ is allowed to have a singularity arbitrarily close to the interval $[-1, 1]$, and the convergence of Chebyshev interpolation will still be exponential. Of course, the exponential rate will be very poor, when $R \approx 1$.

4.8.2 Analysis of a Generalized Runge Phenomenon

It is well known that the Taylor series of an analytic function converges at an exponential rate inside its circle of convergence, while it diverges at an exponential rate outside. We shall see that a similar result holds for certain interpolation processes. In general, the domains of convergence are not disks but bounded by level curves of a logarithmic potential, related to the asymptotic distribution of the interpolation points.

For the sake of simplicity, we now confine the discussion to the case, when the points of interpolation are located in the standard interval $[-1, 1]$, but we are still interested in the evaluation of the polynomials in the complex domain. Part of the discussion can, however, be generalized to a case, when the interpolation points are on an arc in the complex plane.

Let $q : [a, b] \mapsto [-1, 1]$ be an increasing and continuously differentiable function. Set $t_{n,j} = a + (b - a)j/n$, $j = 0 : n$, and let the interpolation points be $x_{n,j}$, $j = 1 : n$, where $q(t_{n,j-1}) < x_{n,j} \leq q(t_{n,j})$, i.e. one interpolation point in each of n subintervals of $[-1, 1]$. In the definition of Φ, we now write $x_{n,j}$, Φ_n instead of x_j, Φ. Note that, as $n \to \infty$,

$$\frac{1}{n} \ln \Phi_n(z) = \frac{1}{n} \sum_{j=1}^{n} \ln (z - x_{n,j}) \;\to\; \psi(z) := \frac{1}{b-a} \int_a^b \ln (z - q(t))\, dt, \quad z \notin [-1, 1]. \tag{4.8.5}$$

Put $x = q(t)$, and introduce a density function $w(x), x \in]-1, 1[$ that is the derivative of the inverse function of q, i.e. $w(x) = 1/\big(q'(t(x))(b - a)\big) > 0$. Then

$$\psi(z) = \int_{-1}^{1} \ln(z - x) w(x)\, dx, \quad w(x) > 0, \quad \int_{-1}^{1} w(x)\, dx = 1, \tag{4.8.6}$$

and $\psi(z)$ is analytic in the whole plane outside the interval $[-1, 1]$. Its real part $P(z)$ is the **logarithmic potential** of a weight distribution,

$$P(z) = \Re\psi(z) = \int_{-1}^{1} \ln |z - x| w(x)\, dx, \quad w(x) > 0. \tag{4.8.7}$$

The function $\frac{1}{n} \ln |\Phi_n(z)|$ is itself the logarithmic potential of a discrete distribution of equal weights $\frac{1}{n}$, at the interpolation points $x_{j,n}$. This function is less pleasant to deal with than $P(z)$, since it becomes $-\infty$ at the interpolation points while, according to classical results of potential theory, $P(z)$ *is continuous everywhere, also on the interval* $[-1, 1]$. If we set $z = x + iy$, $\partial P(z)/\partial x$ is also continuous for $z \in]-1, 1[$, while $\partial P(z)/\partial y$ has a jump there. We write it thus,

$$\psi'(x - 0i) - \psi'(x + 0i) = 2\pi i w(x). \tag{4.8.8}$$

Figure 4.8.1. *Figure to be made.*

By (4.8.6), $\psi(z) = \ln z + O(z^{-1})$, $|z| \to \infty$, or even $O(z^{-2})$, if the weight distribution is symmetric around the origin.

We make the definition

$$\mathcal{D}(v) = \{z \in \mathbf{C} : P(z) < P(v)\}.$$

and set $P^* = \max_{x\in[-1,1]} P(x)$. It can be shown that $\mathcal{D}(v)$ is a simply connected domain if $P(v) > P^*$. The level curve $\partial\mathcal{D}(v) = \{z : P(z) = P(v)\}$ then encloses $[-1, 1]$. A level curve $\{z : P(z) = a\}$ is strictly inside the level curve $\{z : P(z) = a'\}$ if $a \le P^* < a'$. (The proof of these statements essentially utilizes the minimum principle for harmonic functions and the fact that $P(z)$ is a regular harmonic function outside $[-1, 1]$ that grows to ∞ with $|z|$.)

We now consider two examples.

Example 4.8.1. *Equidistant interpolation*

In this case we may take $q(t) = t$, $t \in [-1, 1]$, hence $w(x) = 1/2$. For the **equidistant case** we have if $z \notin [-1, 1]$,

$$P(z) = \tfrac{1}{2}\Re \int_{-1}^{1} \ln(z - x)\, dx = \tfrac{1}{2}\Re\big((1 - z)\ln(z - 1) + (1 + z)\ln(z + 1)\big) - 1.$$

The upper half of the level curves may look something like Figure 4.8.1.:

On the imaginary axis,

$$P(iy) = \tfrac{1}{2}\ln(1 + y^2) + y(\tfrac{1}{2}\pi - \arctan y) - 1.$$

When $z \to x \in [-1, 1]$, from any direction, $P(z)$ tends to

$$P(x) = \tfrac{1}{2}\big((1 - x)\ln(1 - x) + (1 + x)\ln(1 + x)\big) - 1. \tag{4.8.9}$$

$P'(x)$ is continuous in the interior, but becomes infinite at $x = \pm 1$. The imaginary part of $\psi(z)$ has, however, different limits, when the interval is approached from above and below: $\Im(\psi(x \pm 0i)) = \pm\pi(1 - x)$.

The level curve of $P(z)$ that passes through the points ± 1, intersects the imaginary axis at the points $\pm iy$, determined by the equation $P(iy) = P(1) = \ln 2 - 1$, with the root $y = 0.5255$. Theorem 4.8.2 (below) will tell us that $L_n f(x) \to f(x)$, $\forall x \in]-1, 1[$, if $f(z)$ is analytic inside and on this contour.

In the classical example of Runge, $f(z) = 1/(1 + 25z^2)$ has poles inside this contour at $z = \pm 0.2i$. Proposition 4.8.3 will tell us that the level curve of $P(z)$ that passes through these poles will separate between the points, where the interpolation process converges and diverges. Its intersections with the real axis is determined by the equation $P(x) = P(0.2i) = -1.41142$. The roots are $x = \pm 0.72668$.

Example 4.8.2. *Chebyshev interpolation*

In this example we have

$$q(t) = \cos(\pi(1-t)), \quad t \in [0,1], \qquad w(x) = \frac{1}{\pi}(1-x^2)^{-\frac{1}{2}}.$$

Moreover (see Sec. 3.5.1) substitute s for w,

$$\Phi_n(z) = 2^{1-n} T_n(z) = 2^{-n}(s^n + s^{-n}),$$

where $z = \frac{1}{2}(s + s^{-1})$, $s = z + \sqrt{z^2 - 1}$. Note that $|s| \geq 1$, according to our convention about the choice of branch for the square root. Hence,

$$P(z) = \lim \frac{1}{n} \ln |\Phi_n(z)| - \ln 2 = \ln \frac{|s|}{2} = \ln |z + \sqrt{z^2 - 1}| - \ln 2.$$

Therefore, the family of confocal ellipses ∂E_R are, in this example, the level curves of $P(z)$. In fact, by (1.3') and the formula for $P(z)$, the interior of E_R equals $\mathcal{D}(\ln R - \ln 2)$. The family includes, as a limit case ($R = 1$), the interval $[-1, 1]$, in which $P(z) = -\ln 2$.

Our problem is related to a more conventional application of potential theory, namely the problem of finding the electrical charge distribution of a long insulated charged metallic plate in the strip

$$\{(x, y) \in \mathbf{R}^2 : \ -1 < x < 1, -L < y < L\}, \quad L \gg 1.$$

Such a plate will be equipotential. The charge density at the point (x, y) is then proportional to

$$w(x) = \frac{1}{\pi}(1-x^2)^{-1/2};$$

a fascinating relationship between electricity and approximation.

Note that if $z \notin [-1, 1]$, we can, by the definition of $P(z)$ as a Riemann sum (see (4.1)) find a sequence $\{\epsilon_n\}$ that decreases monotonically to zero, such that

$$\frac{1}{n} |\ln \Phi_n(z) - \psi(z)| < \epsilon_n, \quad z \notin [-1, 1]. \tag{4.8.10}$$

It is conceivable that the same sequence can be used for all z on a curve that does not touch the interval $[-1,1]$. (The proof is omitted.)

We can only claim a *one-sided inequality*, if we allow that $u \in [-1,1]$.

$$\frac{1}{n}(\Re \ln \Phi_n(u) - \psi(u)) < \epsilon_n, \quad u \in \mathbf{C}. \tag{4.8.11}$$

(Recall that $\Re \ln \Phi_n(u) = -\infty$ at the interpolation points.) We can use the same sequence for z and u. We can also say that $|\Phi_n(u)|$ *behaves like* $\exp\big((P(u) \pm \delta)n\big)$ *outside the immediate vicinity of the interpolation points.*

Theorem 4.8.2.

Assume that $[-1,1]$ *is strictly inside a simply connected domain* $\mathcal{D} \supseteq \mathcal{D}(v)$. *If* $f(\zeta)$ *is analytic in the closure of* $\mathcal{D}$, *then the interpolation error* $(L_n f)(u) - f(u)$ *converges like an exponential to* 0 *for any* $u \in \mathcal{D}(v)$.

Proof. By Theorem 4.8.1, $f(u) - (L_n f)(u) = I_n(u)$, where

$$I_n(u) = \frac{1}{2\pi i} \int_{\partial \mathcal{D}} \frac{\Phi_n(u) f(z)}{\Phi_n(z)(z-u)}\, dz. \tag{4.8.12}$$

Note that $P(z) \geq P(v)$, because $\mathcal{D} \supseteq \mathcal{D}(v)$. Then, by (4.8.10) and (4.8.11),

$$|\Phi_n(u)/\Phi_n(z)| < \exp n\big(P(u) - P(v) + 2\epsilon_n\big)$$

Let $|f(z)| \leq M$. For any $u \in \mathcal{D}(v)$, we can choose $\delta > 0$, such that $P(u) < P(v) - 3\delta$, $|z-u| > \delta$. Next, choose n large enough so that $\epsilon_n < \delta$. Then

$$|f(u) - (L_n f)(u)| < \frac{1}{2\pi} M \exp n(-3\delta + 2\delta) \int_{\partial \mathcal{D}(v)} \frac{|dz|}{\delta} \leq \frac{K \exp(-n\delta)}{\delta}.$$

where $K = K(v)$ does not depend on n, δ and u, hence the convergence is exponential. □

Remark 4.8.1. *If* u *is away from the boundary* $\partial\mathcal{D}(v)$, *more realistic estimates of the interpolation error and the speed of convergence is for* $n \gg 1$ *given by*

$$K_1(v)|\Phi_n(u)|e^{(-P(v)+\delta)n} \leq K_1(v) e^{(P(u)-P(v)+\delta)n},$$

where $K_1(v)$ *is another constant. The latter estimate is realistic outside the immediate vicinity of the interpolation points.*

It seems, as if one should choose $|v|$ as large as possible, in order to increase $P(v)$. A bound for $P(v)$ is usually set by the singularities of $f(z)$. If $f(z)$ is an entire function, the growth of the maximum modulus of $|f(z)|$, $|z| \in \mathcal{D}(v)$, hidden in $K_1(v)$, sets a bound for $P(v)$ that usually increases with n.

We shall now derive a complement and a kind of converse to Theorem 4.8.2, for functions $f(z)$ that have simple poles in $\mathcal{D}(v)$.

Proposition 4.8.3.

Assume that $[-1,1]$ *is strictly inside a domain* $\mathcal{D} \supset \mathcal{D}(v)$*, and that* $f(\zeta)$ *is analytic in the closure of* $\mathcal{D}$*, except for a finite number of simple poles* p *in the interior, all with the same value of* $P(p)$*.*

Outside the interval $[-1,1]$*, the curve* $\partial\mathcal{D}(p)$ *then separates the points, where the sequence* $\{(L_n f)(u)\}$ *converges, from the points, where it diverges. The behavior of* $|(L_n f)(u) - f(u)|$*, when* $u \in \mathcal{D}(v)$*,* $n \gg 1$ *is roughly described by the formula,*

$$|(f - L_n f)(u)| \approx K|\Phi_n(u)|e^{(-P(p)\pm\delta)n} / \max_p(1/|p-u|). \tag{4.8.13}$$

This can be further simplified, if u *is not in immediate vicinity of the interpolation points, see* (4.8.14)*.*

Proof. (Sketch:) At the application of the residue theorem to the integral $I_n(u)$, see (4.8.12), we must this time also consider the poles of $f(z)$. We obtain

$$\frac{I_n(u)}{\Phi_n(u)} = \frac{(f - L_n f)(u)}{\Phi_n(u)} + \sum_p \frac{\mathrm{res_f}(p)}{\Phi_n(p)(p-u)},$$

where $\mathrm{res_f}(p)$ is the residue of f at the pole $p \in \mathcal{D}(v)$. Roughly speaking, for $n \gg 1$,

$$I_n(u) = O\big(e^{-P(v)+\delta)n}\big), \qquad \sum_p = O\big(e^{(-P(p)\pm\delta)n} / \max_p(1/|p-u|), \quad P(v) > P(p).$$

It follows that $|I_n| \ll \sum$, unless there is a cancellation of terms in $\sum$. For fixed n, $\sum = 0$ is equivalent to an algebraic equation of degree less than the number of poles, and the roots will depend on n. We conclude that the case of cancellation can be ignored, and hence we obtain (4.8.13), and the following simplified version, valid if u is not in the immediate vicinity of the interpolation points. $|\Phi_n(u)|$ behaves like $\exp(P(u) \pm \delta)n$.

$$|(f - L_n f)(u)| \approx K e^{(P(u)-P(p)\pm\delta)n} / \max_p(1/|p-u|). \tag{4.8.14}$$

The separation statement follows from this. □

There are several interpolation processes with interpolation points in $[-1,1]$ that converge for all $u \in [-1,1]$, when the condition of analyticity is replaced by a more modest smoothness assumption, e.g., $f \in C^p$. This is the case, when the sequence of interpolation points are the zeros of the orthogonal polynomials which belong to a density function that is continuous and strictly positive in $]-1,1[$. We shall prove the following result.

Proposition 4.8.4.

Consider an interpolation process where the interpolation points has a (perhaps unknown) asymptotic density function $w(x)$*,* $x \in [-1,1]$*. Assume that*

$$(L_n f - f)(x) \to 0, \quad \forall x \in [-1,1], \quad \forall f \in C^k[-1,1],$$

as $n \to \infty$, for some $k \geq 1$. Then the logarithmic potential $P(x)$ must be constant in $[-1, 1]$, and the density function must be the same as for Chebyshev interpolation, i.e. $w(x) = \frac{1}{\pi}(1 - x^2)^{-1/2}$.

Proof. Let $f(z)$ be analytic in some neighborhood of $[-1, 1]$, e.g. any function with a pole at a point p (arbitrarily) close to this interval. A fortiori, for such a function our interpolation process must converge at all points u in some neighborhood of the interval $[-1, 1]$.

Suppose that $P(x)$ is not constant, and let x_1, x_2 be points, such that $P(x_1) < P(x_2)$. We can then choose the pole p so that $P(x_1) + \delta < P(p) < P(x_2) - \delta$. By Proposition 4.8.3, the process would then diverge at some points u arbitrarily close to x_2. This contradiction shows that $P(x)$ must be constant in $[-1, 1]$, $P(x) = a$, (say).

This gives a Dirichlet problem for the harmonic function $P(z)$, $z \notin [-1, 1]$, which has a unique solution, and one can verify that the harmonic function $P(z) = a + \Re \ln(z + \sqrt{z^2 - 1})$ satisfies the boundary condition. We must also determine a. This is done by means of the behaviour as $z \to \infty$. We find that

$$\begin{aligned} P(z) &= a + \Re \ln(z + z(1 - z^{-2})^{1/2}) \\ &= a + \Re \ln(2z - O(z^{-1}) = a + \Re \ln z + \ln 2 - O(z^{-2}). \end{aligned}$$

This is to be matched with the result of the discussion of the general logarithmic potential in the beginning of Sec. 4.8.2. In our case, where we have a symmetric distribution, and $\int_1^1 w(x)\,dx = 1$, we obtain $P(z) = \Re\psi(z) = \Re \ln z + O(z^{-2})$. The matching yields $a = -\ln 2$.

Finally, by (4.8.6), we obtain after some calculation, $w(x) = (1-x^2)^{-1/2}$. The details are left for Problem 3. □

Compare the above discussion with the derivations and results concerning the asymptotic distribution of the zeros of orthogonal polynomials, given in the standard monograph G. Szegö [54].

4.8.3 The Sampling Theorem

The ideas of this paper can be applied to other interpolation problems than polynomial interpolation. We shall apply them to a derivation of the celebrated sampling theorem which is an interpolation formula that expresses a function that is **band-limited** to the frequency interval $[-W,\ W]$, i.e. a function that has a Fourier representation of the following form (see also Strang [53, p. 325].

$$f(z) = \frac{1}{2\pi} \int_{-W}^{W} \hat{f}(k) e^{ikz}\,dk, \quad |\hat{f}(k)| \leq M, \tag{4.8.15}$$

in terms of its values at all integer points. The **Shannon Sampling Theorem** reads,

$$f(z) = \sum_{j=-\infty}^{\infty} f\left(\frac{j\pi}{W}\right) \frac{\sin(Wz - j\pi)}{(Wz - j\pi)}. \tag{4.8.16}$$

This is, like Lagrange's interpolation formula, a so-called *cardinal interpolation formula*. As Wz/π tends to an integer m, all terms except one on the right hand side become zero; for $j = m$ the term becomes $f(m\pi/W)$.

We shall sketch a derivation of this for $W = \pi$. We first note that (4.8.15) shows that $f(z)$ is analytic for all z. Then we consider the same Cauchy integral as many times before,

$$I_n(u) = \frac{1}{2\pi i}\int_{\partial\mathcal{D}_n} \frac{\Phi(u)f(z)}{\Phi(z)(z-u)}\,dz, \quad u \in \mathcal{D}_n.$$

Here $\Phi(z) = \sin \pi z$, which vanishes at all integer points, and $\mathcal{D}_n$ is the *open* rectangle with vertices at $\pm(n + 1/2) \pm bi$. By the residue theorem, we obtain after a short calculation,

$$I_n(u) = f(u) + \sum_{j=-n}^{n} \frac{\Phi(u)f(j)}{\Phi'(j)(j-u)} = f(u) - \sum_{j=-n}^{n} \frac{f(j)\sin\pi(j-u)}{\pi(j-u)}.$$

Set $z = x + iy$. Note that

$$|f(z)| \le \frac{1}{2\pi}\int_{-\pi}^{\pi} Me^{-ky}dk \le \frac{M(e^{|\pi y|} - e^{-|\pi y|})}{|2\pi y|}, \quad |\Phi(z)| \ge e^{|\pi y|}.$$

These inequalities, applied for $y = b$, allow us to let $b \to \infty$; ($2b$ is the height of the symmetric rectangular contour). Then it can be shown that $I_n(u) \to 0$ as $n \to \infty$, which establishes the sampling theorem for $W = \pi$. The general result is then obtained by "regula de tri", but it is sometimes hard to get it right Strang [53] gives an entirely different derivation, based on Fourier analysis.

Problems and Computer Exercises

1. We use the notations and assumptions of Theorem 4.8.1. (a) Using the representation of the interpolation operator as an integral operator, show that

$$(L_n f)(x) = \frac{1}{2\pi i}\int_{\partial\mathcal{D}} K(x,z)\frac{f(z)}{\Phi(z)}\,dz, \quad K(x,z) = \frac{\Phi(x) - \Phi(z)}{(x-z)},$$

also if $x \notin \mathcal{D}$. Note that $K(x, z)$ is a polynomial, symmetric in the two variables x, z.

(b) *A formula for the divided difference.* Show that

$$[x_1, x_2, \ldots, x_n]f = \frac{1}{2\pi i}\int_{\partial\mathcal{D}} \frac{f(z)}{\Phi(z)}\,dz.$$

Hint: Look at the leading term of the polynomial $(L_n f)(x)$.

2. Check the omitted details of the derivations in Sec. 4.8.3.

3. Check the validity of (4.8.6) on the Chebyshev and the equidistant cases. Also show that $\int_{-1}^{1} w(x)\,dx = 1$, and check the statements about the behaviour of $P(z)$ for $|z| \gg 1$.

4. (a) Work out the details of the proof of the Sampling Theorem.

(b) The formulation of the Sampling Theorem with a general W in Strang [53] does not agree with ours in (4.8.16). Who is right?

5. (a) Write a program for solving equations of the form $\psi(z) = c$, where c runs through a rectangular grid in a complex plane, not necessarily equidistant. You may assume that ψ is defined in such a way, that that its derivative is rather easily computed. When applicable, compute also the intersections of two families of level curves, i.e. with constant $\Re c$ and constant $\Im c$, with the real axis.

(b) Apply your program(s) to the plotting of these level curves in the two cases of the text, (related to, respectively, equidistant and Chebyshev interpolation). Due to the symmetry it is sufficient to draw the curves in a quarter-plane. Think of the "aspect" of the plotting so that the conformality of the mappings becomes visible.

If the scanning of the grid (of c) leads z to cross the forbidden interval $[-1, 1]$, or to some other exceptional situation, the program should return a nice message, and continue the scanning without interrupt, with more fruitful values of c, so that nothing is lost. By the way, find out, how the system you work with, handles the logarithm and square root in the complex domain. It may not be entirely according to our conventions, but it almost certainly produces some value that your own program can modify appropriately.

(c) If $\Re c \gg 1$, the level curve for the real part is, close to a circle (why?). Use equidistant values of $\Im c \in [0, \frac{1}{2}\pi]$. The values of $\Re c$ are to be chosen so that the drawings become intellectually interesting and/or visually pleasing. You are then likely to find that the density of the level curves for the imaginary part, when they approach the interval $[0, 1]$ is different for the Chebyshev case and the equidistant case. Explain theoretically how this is related to the density function w in the text.

(d) The level curves of the imaginary part intersect the interval $[0, 1]$, at different angles in the Chebyshev and the equidistant cases. Give a theoretical analysis of this.

6. (a) (After Meray (1884) and Cheney [11, p. 65]. Let $L_n f$ be the polynomial of degree $< n$, which interpolates to the function $f(z) = 1/z$ at the n'th roots of unity. Show that $(L_n f)(z) = z^{n-1}$, and that

$$\lim_{n\to\infty} \max_{|u|=1} |(L_n f - f)(u)| > 0.$$

Hint: Solve this directly, without the use of the previous theory.

(b) Modify the theory of Sec. 4.8.1 to the case in (a) with equidistant interpolation points on the unit circle, and make an application to $f(z) = 1/(z-a)$, $a > 0$, $a \neq 1$. Here, $\Phi_n(z) = z^n - 1$. What is $\psi(z)$, $P(z)$? The density function? (The integral for $\psi(z)$ is a little tricky, but you may find it in a

table. There are, however, simpler alternatives to the integral, see the end of Sec. 4.8.1. Check your result by thinking like Faraday.) Find out for which values of a, u, ($|u| \neq 1$, $|u| \neq a$), $(L_n f - f)(u) \to 0$, and estimate the speed of convergence (divergence).

(c) What can be said about the cases excluded above, i.e. $|u| = 1$, $|u| = a$? Also look at the case, when $|a| = 1$, ($a \neq 1$).

(d) Is the equidistant interpolation on the unit circle identical to the Cauchy FFT method (with $a = 0$, $R = 1$) for the approximate computation of the coefficients in a power series? See, in particular (3.1.10).

7. (a) We saw in 6 (b) that the equidistant interpolation on the unit circle gives no good polynomial approximation when the pole is inside the unit circle. The coefficients computed by the Cauchy FFT are however useful with a different interpretation, namely as coefficients in an interpolation polynomial $p(z^{-1})$ for $f(z) = 1/(z-a)$, or as approximate coefficients in a Laurent series $1/(z-a) = \sum_{j=1}^{\infty} c_j z^{-j}$, that converges for $|z > a|$. Note that the Cauchy integral, (3.1.8), is valid also for the coefficients of a Laurent expansion. Now consider

$$f(z) = \frac{-5}{(3-z)(1-2z)} = \frac{1}{3-z} - \frac{2}{1-2z}.$$

This has three Laurent expansions, i.e. an ordinary Taylor series for $|z| < \frac{1}{2}$, an expansion into negative powers for $|z| > 3$, and a mixed expansion for the annulus $\frac{1}{2} < |z| < 3$. It is conceivable that the the first two expansions can be found by FFT, with different interpretations of the results, but what about the annulus case? It is easily seen from the above partial fraction form of $f(z)$ what the Laurent expansion should be. When the FFT is applied to $f(z)$, it does therefore, in principle, find a coefficient by adding a coefficient of a *negative* power of z from the first term of the partial fraction decomposition, to the coefficient of a *positive* power of z from the second term. *Can this really work?*

Explain, why things go so well with a careful treatment, in spite that we almost tried to convince you above that it would not work. Also try to formulate what "careful treatment" means in this case.

Hint: Generalize to the case of a Laurent expansion the relation between the FFT output and the series coefficients given (for a Taylor series) in (3.1.11). Also read in Strang [53, Chapter 4] or somewhere else about "aliasing".

Notes and Further Reading

The problem of choosing a good orderings of points in Newton and Lagrange interpolations is discussed in [57]. Newton interpolation using the Leja ordering of points has been analyzed by Reichel [40]. The barycentric form of Lagrange's interpolation formula was advocated in lecture notes by Rutishauser [43] already in the 1960's. Berrut and Trefethen [3] argue convincingly that this should be the standard method of polynomial interpolation, and in historical notes discuss why

it is not better known. The scheme for computing the inverse of a Vandermonde matrix is due to Higham [31, Sec. 22.1].

The $O(n^2)$ algorithm for solving primal Vandermonde systems decribed in Sec. 4.3.4 is due to Björck and Pereyra [4]. It has been generalized to yield fast algorithm for Vandermonde-like matrices defined by $V = (v_{ij}) = ((p_i(x_j))$, where p_i is a polynomial of degree n that satisfies a thre term recurrence relation; see Higham [31, Sec. 22.2]. Also so called Cauchy linear systems can be solved with a Björck–Pereyra-type algorithm; see Boros, Kailath and Olshevsky [8].

The computational advantage of the Stieltjes approach for discrete least squares fitting was pointed out by Forsythe [24, 1956]. Shampine [48, 1975] established the advantage in using the alternative formula involving the residual r_k.

Working for the French car companies Renault and Citroën, Bézier and de Casteljau, independently in 1962 developed the Bézier curve as a tool in Computer Aided Design (CAD) for fitting curves and surfaces. A more geometric view of spline functions is taken in Farin [23]. Several packages are available for computing with splines, e.g., the spline toolbox in MATLAB and FITPACK Dierckx [20]–[21].

The FFT algorithm algorithm has been discovered independently by several people. Indeed the idea was published in a paper by Gauss and the doubling algorithm is contained in a textbook by Runge and König [42]. The modern usage of FFT started in 1965 with the publication of the papers [15, 14] by James W. Cooley of IBM Research and John W. Tukey, Princeton University. The re-discovery of the FFT algorithm is surveyed by James W. Cooley in [13]. Applications are surveyed in [10] and [9]. The matrix-oriented framework for the FFT used in this book is developed in [56]). A roundoff error analysis is given in [1].) Algorithms for the bit-reversal permutation are reviewed in [34].

Ideas related to those in Sec. 4.8.2 were applied in the thesis of Lothar Reichel at KTH. He studied the Helmholtz equation in 2D, with a regionally constant complex coefficient, with potential applications (excuse our pun!), e.g., to the microwave heating of cheeseburgers. Since one can find nice bases of particular solutions of the Helmholtz equation in different regions, i.e. bread, meat and cheese, one may try **boundary collocation**, to express the appropriate continuity conditions at the interfaces between meat and cheese etc. in a finite number of points.

Bibliography

[1] Mario Arioli, Hans Z. Munthe-Kaas, and L. Valdettaro. Componentwise error analysis for FFTs with applications to fast Helmholz solvers. *Numer. Algorithms*, 12:65–88, 1996.

[2] R. W. Barnard, Germund Dahlquist, K. Pearce, Lothar Reichel, and K. C. Richards. Gram polynomials and the Kummer function,. *J. Approx. Theory*, 94:128–143, 1998.

[3] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange interpolation. *SIAM Review*, 46:3:501–517, 2004.

[4] Åke Björck and Victor Pereyra. Solution of Vandermonde system of equations. *Math. of Comp.*, 24:893–903, 1970.

[5] P. Bloomfield. *Fourier Analysis and Time Series*. John Wiley, New York, 1976.

[6] Carl de Boor. On calculating with B-splines. *J. Approx. Theory*, 6:50–62, 1972.

[7] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, Berlin, revised edition, 1991.

[8] Tibor Boros, Thomas Kailath, and Vadim Olshevsky. A fast parallel Björck–Pereyra-type algorithm for solving Cauchy linear systems. *Linear Algebra Appl.*, 302–303:265–293, 1999.

[9] W. L. Briggs and Van Emden Henson. *The DFT. An Owners Manual for the Discrete Fourier Transform*. SIAM, Philadelphia, PA, 1995.

[10] E. O. Brigham. *The Fast Fourier Transform and Its Application*. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[11] E. W. Cheney. *Introduction to Approximation Theory*. McGraw-Hill, New York, NY, 1966.

[12] E. W. Cheney and W. Light. *A Course in Approximation Theory*. Brooks/Cole, Pacific Grove, CA, 2000.

[13] James W. Cooley. The re-discovery of the fast Fourier transform algorithm. *Mikrochimica Acta,*, 3:33–45, 1987.

[14] James W. Cooley, Peter A. W Lewis, and Peter D. Welsh. The fast Fourier transform and its application. *IEEE Trans. Education,*, E–12:27–34, 1969.

[15] James W. Cooley and John W. Tukey. An algorithm for machine calculation of complex Fourier series. *Math. Comp.,*, 19:297–301, 1965.

[16] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume I. Interscience, New York, 1953.

[17] Maurice G. Cox. The numerical evaluation of B-splines. *J. Inst. Math. Appl.*, 10:134–149, 1972.

[18] G. Danielson and Cornelius Lanczos. Some improvements in practical Fourier analysis and their applications to x-ray scattering from liquids. *J. Franklin Inst.*, 233:365–380, 435–452, 1942.

[19] Philip J. Davis. *Interpolation and Approximation*. Dover, New York, NY, 1975.

[20] P. Dierckx. FITPACK user guide part i: Curve fitting routines. TW Report 89, Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, 1983.

[21] P. Dierckx. FITPACK user guide part i: Surface fitting routines. TW Report 122, Department of Computer Science, Katholieke Universiteit, Leuven, Belgium, 1983.

[22] P. Dierckx. *Curve and Surface Fitting with Splines*. Clarendon Press, New York, 1993.

[23] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, New York, 1988.

[24] George E. Forsythe. Generation and use of orthogonal polynomials for data-fitting with a digital computer,. *J. Soc. Indust. Appl. Math.*, 5:74–88, 1957.

[25] Walter Gander. Change of basis in polynomial interpolation. *Numer. Linear Algebra Appl.*, page submitted, 2004.

[26] Walter Gautschi. *Numerical Analysis, an Introduction*. Birkhäuser, Boston, MA, 1997.

[27] W. M. Gentleman and G. Sande. Fast Fourier transforms—for fun and profit. In *Proceedings AFIPS 1966 Fall Joint Computer Conference*, pages 503–578. Spartan Books, Washington, D.C., 1966.

[28] Ernst Hairer and Gerhard Wanner. *Analysis by Its History*. Springer Verlag, Berlin, third corrected printing edition, 2000.

[29] Peter Henrici. *Essentials of Numerical Analysis*. John Wiley, New York, 1982.

[30] Nicholas J. Higham. Error asnalysis of the Björck–Pereyra algorithm for solving Vandermonde systems. *Numer. Math.*, 50:613–632, 1987.

[31] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, second edition, 2002.

[32] Nicholas J. Higham. The numerical stability of barycentric Lagrange interpolation. *IMA J. Numer. Anal.*, 24:547–556, 2004.

[33] F. B. Hildebrand. *Introduction to Numerical Analysis*. McGraw-Hill, New York, 1974.

[34] Alan H. Karp. Bit reversal on uniprocessors. *SIAM Review*, 38:1:1–26, 1996.

[35] Fred T. Krogh. A variable step variable order multistep method for the numerical solution of ordinary differential equations. In A. J. Morell, editor, *Proceedings of the IFIP Congress 1968*, pages 194–199. North-Holland, Amsterdam, 1969.

[36] J. G. Mason and D. C. Handscomb. *Chebyshev Polynomials*. Chapman & Hall/CRC, London, 2003.

[37] Cleve Moler and Steve Eddins. Fast finite Fourier transforms. *MATLAB News and Notes*, pages 14–15, Winter, 2001.

[38] M. J. D. Powell. *Approximation Theory and Methods*. Cambridge University Press, Cambridge, UK, 1981.

[39] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in Fortran; The Art of Scientific Computing*. Cambridge University Press, Cambridge, GB, second edition, 1992.

[40] Lothar Reichel. Newton interpolation at Leja points. *BIT*, 30:332–346, 1990.

[41] Friedrich Riesz and Béla Sz.-Nagy. *Vorlesungen Über Funktionalanalysis*. VEB Deutscher Verlag der Wissenschaften, Berlin, 1956.

[42] Carle Runge and H. König. *Vorlesungen über Numerisches Rechnen. Band XI*. Verlag Julius Springer, Berlin, 1924.

[43] Heinz Rutishauser. *Vorlesungen über numerische Mathematik, Vol. I*. Birkhäuser, Basel–Stuttgart, 1976. English translation *Lectures on Numerical Mathematics*, by W. Gautschi, Birkhäuser, Boston, 1990.

[44] I. J. Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. *Quart. Appl. Math.*, 4:45–99 and 112–141, 1946.

[45] I. J. Schoenberg and A. Whitney. On Pólya frequency functions III: The positivity of translation determinants with an application to the interpolation problem by spline curves. *Trans. Amer. Math. Soc.*, 74:246–259, 1953.

[46] H. R. Schwarz. *Numerische Methematik*. Teubner, Stuttgart, fourth edition, 1997. English translation of 2nd ed.: Numerical Analysis: A Comprehensive Introduction, John Wiley, New York.

[47] Hubert Schwetlick and Torsten Schütze. Least squares approximation by splines with free knots. *BIT*, 35:361–384, 1995.

[48] Lawrence F. Shampine. Discrete least squares polynomial fits. *Comm. ACM*, 18:179–180, 1975.

[49] R. C. Singleton. On computing the fast Fourier transform. *Comm. ACM*, 10:647–654, 1967.

[50] R. C. Singleton. Algorithm 338: Algol procedure for the fast Fourier transform with arbitrary factors. *Comm. ACM*, 11:773–779, 1968.

[51] J. F. Steffensen. *Interpolation.* Chelsea, New York, second edition, 1950.

[52] Joseph Stoer and Roland Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, New York, third edition, 2002.

[53] Gilbert Strang. *Introduction to Applied Mathematics.* Wellesley-Cambridge Press, Wellesley, MA, 1986.

[54] Gabor Szegö. *Orthogonal Polynomials*, volume 23 of *Colloq. Publ.* Amer. Math. Soc., Providence, RI, fourth edition, 1975.

[55] T. N. Thiele. *Interpolationsrechnung.* B. G. Teubner, Leipzig, 1909.

[56] Charles F. Van Loan. *Computational Framework for the Fast Fourier Transform.* SIAM, Philadelphia, 1992.

[57] Wilhelm Werner. Polynomial interpolation: Lagrange versus Newton. *Math. Comp.*, 43:205–217, 1984.

[58] M. Zelen. Linear estimation and related topics. In John Todd, editor, *Survey of Numerical Analysis*, pages 558–584. McGraw-Hill, New York, 1962.

Index

adjoint operator, 90
Aitken interpolation, 24–25
algorithm
  divided difference table, 15
  Newton coefficients, 15
  Vandermonde system, 43
    dual, 42
aliasing, 120
approximation
  in maximum norm, 81
arrowhead system, 61

B-spline, 64–74
  basis, 68
  definition, 66
  evaluation, 71
  exterior knots, 65
  hat function, 65
  multiple knots, 68
  properties, 67
  recurrence relation, 69
Bézier
  curve, 49–53
  polygon, 50
Banach space, 80
Bernstein polynomials, 47–49
  derivatives, 51
Bessel's inequality, 93
bilinear interpolation, 44
Björck–Pereyra algorithm, 43
butterfly relations, 125

cardinal basis, 6
Cauchy sequence, 78
Cauchy–Schwarz inequality, 88
Chebyshev
  interpolation, 8, 10, 17, 22, 96, 138
  points, 4, 17
  support coefficients, 22
circulant matrix, 129
complete space, 78
complex analysis, 137–145
computer aided design, 49
continued fraction, 28
control points, 49
convex
  hull, 50
  set, 50
convolution, 118
  discrete, 129
correlation, 123
cubic spline
  'not a knot' condition, 59
  complete interpolant, 58
  interpolation error, 61–64
  natural interpolant, 59
  periodic boundary conditions, 59, 60
  tridiagonal system, 57

de Casteljau's algorithm, 52
discrete
  cosine transform (DCT), 132
  sine transform (DST), 132
distance, 78
divided difference, 11
  inverse, 29
  reciprocal, 29
  scaled, 19
  table, 13

Euclidean norm, 80

weighted, 80
Euler's formulas, 108

Fast Fourier Transform, 123–134
FFT, *see* Fast Fourier Transform
Cooley–Tukey, 128
Gentleman–Sande, 128
Fourier, 108
analysis
continuous case, 111
discrete case, 113
coefficients, 91, 110
matrix, 124
series, 108
function
aliased, 120
analytic, 137–145

Gauss–Markov theorem, 102
Gram polynomials, 101

Hermite interpolation, 33–38
Hilbert space, 88

inner product space, 88–91
interpolation
Birkhoff, 37
broken line, 55
condition number, 25
error in linear, 32
Hermite, 33–38
inverse, 38–39
iterative linear, 24–25
lacunary, 37
of analytic functions, 137–139
osculatory, 33–38
piecewise cubic, 56
rational, 27–30
remainder term, 14
with derivatives, 33–38
interpolation formula
barycentric form, 21
Hermite's, 33
Lagrange's, 6, 19–24
Newton's, 12
inverse divided difference, 29
inverse interpolation, 38–39

knot, 54

Lagrange
interpolation, 6
polynomial
generalized, 34
Lagrange's
interpolation formula, 19–24
polynomials, 20
least squares, 7
approximation, 6–7
data fitting, 100
statistical aspects, 102–104
Lebesgue constant, 26
Leibniz' formula, 69
Leja
ordering, 18
points, 19
linear approximation, 79
linear space, 79
linear system
overdetermined, 7
logarithmic potential, 139

matrix
circulant, 136
shift, 136
totally nonnegative, 72
maximum norm, 80
metric space, 78
multidimensional interpolation, 44–46
multiplicity
of interpolation point, 33

Neville's algorithm, 24–25
Newton polynomials, 5
Newton's interpolation formula, 12
norm, 79
L_p, 80
l_p, 80
of operator, 82–83
norm and distance formula, 84–87
normal equations, 7, 91
numerical differentiation, 40

Nyquist critical frequency, 119

operator
  norm, 82–83
  positive definite, 90
  self-adjoint, 90
orthogonal
  coefficients, 91
  expansion, 92
  function, 89
  polynomials, 94–102
    construction, 96
  system, 88–91
orthonormal system, 89
osculating polynomial, 33
osculatory interpolation, 33–38

parametric spline, 61
Parseval's identity, 93, 113
Peano kernel, 68
permutation
  bit-reversal, 126
  perfect shuffle, 127
polynomial interpolation, 137–145
power basis, 2
  shifted, 4
  truncated, 64
projection, 90
Pythagoras' theorem, 89

rational interpolation, 27–30
  Neville-type, 30
reciprocity relations, 119
remainder term
  interpolation, 14
Runge's phenomenon, 7–9

sampling theorem, 137–145
Schoenberg–Whitney condition, 72
Scylla and Charybdis, 40
self-adjoint operator, 90
Shannon's sampling theorem, 144
smoothing, 7
spectral analysis, 108
spline
  best approximation property, 59
  function, 57–74
    definition, 56
  interpolation, 53–72
    closed curves, 61
  least squares, 72–74
  parametric, 61
  truncated power basis, 64
Stieltjes procedure, 100
support coefficients, 20

titanium data, 73
totally positive matrix, 44
triangle family
  of polynomials, 4–5
triangle inequality, 89
trigonometric polynomials, 108

undetermined coefficients
  method of, 3
uniform convergence, 81
unitary operator, 106

Vandermonde
  systems, 41–44
Vandermonde matrix, 2
  complex, 124
  confluent, 34
  inverse, 23
vector
  conjugate even, 130
  conjugate odd, 130
vector space, 79

Weierstrass' theorem, 87
weighted mean, 99

Contents

5 Numerical Integration **1**

5.1 Interpolatory Quadrature Rules 1
5.1.1 Introduction 1
5.1.2 Some Classical Formulas 3
5.1.3 Higher Order Newton–Cotes' Formulas 7
5.1.4 Weighted Quadrature Rules 11
Review Questions 14
Problems and Computer Exercises 15
5.2 Quadrature Rules with Free Nodes 18
5.2.1 Gauss–Christoffel Quadrature 22
5.2.2 Applications of Gauss Quadrature 24
5.2.3 Matrix Formulas Related to Gauss Quadrature . . . 30
5.2.4 Symmetric Weight Functions 36
Review Questions 37
Problems and Computer Exercises 37
5.3 Extrapolation Methods 40
5.3.1 Euler–Maclaurin Formula 40
5.3.2 Romberg's Method 41
5.3.3 The Epsilon Algorithm 48
5.3.4 Infinite Intervals 49
5.3.5 Adaptive Quadrature 51
Review Questions 54
Problems and Computer Exercises 54
5.4 Multiple Integrals 56
5.4.1 Product Rules 56
5.4.2 Successive One-Dimensional Quadrature 58
5.4.3 Product Rules 59
5.4.4 Irregular Triangular Grids 60
5.4.5 Monte Carlo Methods 65
Review Questions 66
Problems 67

Bibliography **69**

Index **72**

Chapter 5
Numerical Integration

5.1 Interpolatory Quadrature Rules

5.1.1 Introduction

As is well known, even many relatively simple integrals cannot be expressed in finite terms of elementary functions, and must be evaluated by numerical methods. The problem to calculate the definite integral of a given function over a finite interval is often called **numerical quadrature**, since it relates to the ancient problem of the quadrature of the circle, i.e., constructing a square with equal area to that of a circle.

In this chapter we study the problem of *how to find the parameters in a formula* for the approximate calculation integrals

$$I(f) = \int_a^b f(x)\,dx.$$

Note that $I(f)$ is a linear functional and hence the problem is a special case of approximating a linear functional considered in Sec. 3.3.4. The quadrature rules considered will be of the form

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i), \tag{5.1.1}$$

where the **nodes** $x_1 < x_2 < \cdots < x_n$ are distinct and **weights** $w_1, w_2, \ldots, w_n$. Often (but not always) all nodes lie in $[a, b]$.

The weights w_i are usually determined so that the formula (5.1.1) is exact for polynomials of as high degree as possible.

Definition 5.1.1. *A quadrature rule* (5.1.1) *has* **order of accuracy** *(or degree of exactness) equal to d if it is exact for all polynomials of degree $\leq d$, i.e. for all $p \in \mathcal{P}_{d+1}$.*

The coefficients w_i depend only on the distribution of the points $\{x_i\}_{i=1}^n$. Note that the relation

$$\int_a^b dx = \sum_{i=1}^n w_i = (b-a) \tag{5.1.2}$$

follows from the requirement that the formula is exact for $f(x) \equiv 1$. Suppose that the function values $f(x_i)$ is evaluated with an error e_i, such that $|e_i| \leq \epsilon$, for all $i = 1:n$. Then, if $w_i \geq 0$, the related error in the quadrature formula satisfies

$$\Big|\sum_{i=1}^n w_i e_i\Big| \leq \epsilon \sum_{i=1}^n |w_i| \leq \epsilon(b-a). \tag{5.1.3}$$

However, this upper bound does not hold if some weights in the quadrature rules are negative.

In an **interpolatory** quadrature formula the integral is approximated by $\int_a^b w(x)p(x)\,dx$, where $p(x)$ is the unique polynomial of degree $n-1$ interpolating $f(x)$ at the distinct points $x_1, x_2, \ldots, x_n$. By Lagrange's interpolation formula (Theorem 4.2.6)

$$p(x) = \sum_{i=1}^n f(x_i)\ell_i(x), \qquad \ell_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^n \frac{(x-x_j)}{(x_i-x_j)},$$

where $\ell_i(x)$ are the elementary Lagrange polynomials associated with the nodes $x_1, x_2, \ldots, x_n$. It follows that the weights are given by

$$w_i = \int_a^b \ell_i(x)\,dx. \tag{5.1.4}$$

In practice, the coefficients are often more easily computed using the method of undetermined coefficients rather than by integrating $\ell_i(x)$.

An expression for the truncation error is obtained by integrating the remainder (see Theorems 4.2.3 and 4.2.4)

$$\begin{aligned} R_n(f) &= \int_a^b [x_1, \ldots, x_n, x] f \prod_{i=1}^n (x - x_i)\,dx \\ &= \frac{1}{n!}\int_a^b f^{(n)}(\xi_x) \prod_{i=1}^n (x-x_i)\,dx, \quad \xi_x \in [a,b]. \end{aligned} \tag{5.1.5}$$

where the second expression holds if $f^{(n)}$ is continuous in $[a,b]$.

Theorem 5.1.2. *For any given set of nodes $x_1, x_2, \ldots, x_n$ an interpolatory quadrature formula with weights* (5.1.4) *has order of exactness equal to at least $d = n-1$. Conversely, if the formula has degree of exactness $n-1$, then the formula is interpolatory.*

Proof. For any $f \in \mathcal{P}_n$ we have $p(x) = f$, and hence (5.1.4) has degree of exactness at least equal to $n-1$. On the other hand, if the degree of exactness of (5.1.4) is $n-1$, then putting $f = \ell_i(x)$ shows that the weights w_i satisfy (5.1.4), i.e. the formula is interpolatory. □

5.1.2 Some Classical Formulas

Interpolatory quadrature formulas, where the nodes are constrained to be equally spaced, are called **Newton–Cotes'**[1] formulas. These are especially suited for integrating a tabulated function, a task that was more common before the computer age. The midpoint, trapezoidal and Simpson's formula, to be described here, are all special cases of Newton–Cotes' formulas.

The **trapezoidal rule** (cf. Figure 1.2.5) is based on linear interpolation of $f(x)$ at $x_1 = a$ and $x_2 = b$, that is $f(x)$ is approximated by

$$p(x) = f(a) + (x-a)[a,b]f = f(a) + (x-a)\frac{f(b)-f(a)}{b-a}.$$

The integral of $p(x)$ equals the area of a trapezoid with base $(b-a)$ times the average height $\frac{1}{2}(f(a)+f(b))$. Hence

$$\int_a^b f(x)\,dx \approx \frac{(b-a)}{2}(f(a)+f(b)).$$

To increase the accuracy we subdivide the interval $[a,b]$ and assume that $f_i = f(x_i)$ is known on a grid of equidistant points

$$x_0 = a, \quad x_i = x_0 + ih, \quad x_n = b. \tag{5.1.6}$$

where $h = (b-a)/n$ is the **step length**. The trapezoidal approximation for the ith subinterval is

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = T(h) + R_i, \quad T(h) = \frac{h}{2}(f_i + f_{i+1}), \tag{5.1.7}$$

which is the **composite trapezoidal rule**

Assume now that $f''(x)$ is continuous in $[a,b]$. Using the exact remainder in Newton's interpolation formula (see Theorem 4.2.3) we have

$$R_i = \int_{x_i}^{x_{i+1}} (f(x) - p_2(x))\,dx = \int_{x_i}^{x_{i+1}} (x-x_i)(x-x_{i+1})\,[x_i, x_{i+1}, x]f\,dx. \tag{5.1.8}$$

Since $[x_i, x_{i+1}, x]f$ is a continuous function of x and $(x-x_i)(x-x_{i+1})$ has constant (negative) sign for $x \in [x_i, x_{i+1}]$, the mean-value theorem of integral calculus gives

$$R_i = [x_i, x_{i+1}, \xi_i]f \int_{x_i}^{x_{i+1}} (x-x_i)(x-x_{i+1})\,dx, \quad \xi_i \in [x_i, x_{i+1}].$$

[1] Roger Cotes (1682–1716) was a highly appreciated young colleague of Isaac Newton. He was entrusted with the preparation of of the second edition of Newton's *Principia*. He worked and published the coefficients for Newton's formulas for numerical integration for $n \le 11$.

Setting $x = x_i + ht$, and using the Theorem 4.2.4, we get

$$R_i = -\frac{1}{2}f''(\zeta_i)\int_0^1 h^2 t(t-1)h\,dt = -\frac{1}{12}h^3 f''(\zeta_i), \quad \zeta_i \in [x_i, x_{i+1}]. \tag{5.1.9}$$

For another proof of this result using the Peano kernel, see Example 3.2.7.

Summing the contributions for each subinterval $[x_i, x_{i+1}]$, $i = 0 : n.$ gives

$$\int_a^b f(x)\,dx = T(h) + E_T, \quad T(h) = \frac{h}{2}(f_0 + f_n) + h\sum_{i=2}^{n-1} f_i, \tag{5.1.10}$$

where the **global** truncation error is

$$E_T = -\frac{h^3}{12}\sum_{i=0}^{n-1} f''(\zeta_i) = -\frac{1}{12}(b-a)h^2 f''(\xi), \quad \xi \in [a, b]. \tag{5.1.11}$$

(The last equality follows since f'' was assumed to be continuous on the interval $[a, b]$.) This shows that by choosing h small enough we can make the truncation error arbitrary small. In other words we have **asymptotic convergence** when $h \to 0$.

In the **midpoint rule** $f(x)$ is approximated on $[x_i, x_{i+1}]$ by its value $f_{i+1/2} = f((x_i + x_{i+1})/2$ at the midpoint of the interval. This leads to the approximation

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = M(h) + R_i, \quad M(h) = hf_{i+1/2} \tag{5.1.12}$$

The midpoint rule approximation can be interpreted as the area of the trapezium defined by the tangent of f at the midpoint $x_{i-\frac{1}{2}}$.

The remainder term in Taylor's formula gives

$$f(x) - f_{i-\frac{1}{2}} = (x - x_{i-\frac{1}{2}})f'_{i-\frac{1}{2}} + \tfrac{1}{2}(x - x_{i-\frac{1}{2}})^2 f''(\zeta_x), \quad \zeta_x \in [x_{i-1}, x_i].$$

By symmetry the integral over $[x_{i-1}, x_i]$ of the linear term vanishes. We can use the mean value theorem, to show that

$$R_i = \int_{x_i}^{x_{i+1}} \tfrac{1}{2}f''(\zeta_x)(x - x_{i-\frac{1}{2}})^2\,dx = \tfrac{1}{2}f''(\zeta_i)\int_{-\frac{1}{2}}^{\frac{1}{2}} h^3 t^2\,dt = \frac{h^3}{24}f''(\zeta_i).$$

Although it uses just *one function value* the midpoint rule, like the trapezoidal rule, is exact when $f(x)$ is a linear function. Summing the contributions for each subinterval we obtain the **composite midpoint rule**

$$\int_a^b f(x)\,dx = R(h) + E_M, \quad R(h) = h\sum_{i=0}^{n-1} f_{i+1/2}, \tag{5.1.13}$$

(Compare the above approximation with the Riemann sum in the *definition* of a definite integral.) For the global error we have

$$E_M = \frac{(b-a)h^2}{24}f''(\zeta), \quad \zeta \in [a, b]. \tag{5.1.14}$$

The trapezoidal rule is called a **closed rule** because values of f at both endpoints are used. It is not uncommon that f has an integrable singularity at an endpoint. In that case an **open rule**, like the midpoint rule, can still be applied.

If $f''(x)$ has constant sign in each subinterval then the error in the midpoint rule is approximately half as large as that for the trapezoidal rule and has the opposite sign. However, the trapezoidal rule is more economical to use when a sequence of approximations for $h, h/2, h/4, \ldots$ is to be computed, since about half of the values needed for $h/2$ were already computed and used for h, etc. indeed, it is easy to verify the following useful relation between the trapezoidal and midpoint rules:

$$T(h/2) = \frac{1}{2}(T(h) + M(h)). \tag{5.1.15}$$

If the magnitude of the error in the function values does not exceed $\frac{1}{2}U$, then for the trapezoidal and midpoint rules the magnitude of the propagated error in the approximation is bounded by $(b-a)\frac{1}{2}U$, independent of h. Note that this holds *for any quadrature formula* (5.1.1), *provided that all weights* w_i *are positive.*

If the rounding error is negligible and h sufficiently small, then it follows from (5.1.11) that the error in $T(h/2)$ is about 1/4-th of that in $T(h)$. Hence the magnitude of the error in $T(h/2)$ can be estimated by $\frac{1}{3}|T(h/2) - T(h)|$, or more conservatively by $|T(h/2) - T(h)|$. (A more systematic use of Richardson extrapolation is made in Romberg's method; see Sec. 5.3.2.)

Example 5.1.1.

Compute approximately $\displaystyle\int_0^{0.8} \frac{\sin x}{x}\, dx$. As an exercise the reader should check some of the midpoint and trapezoidal sums given below, which are correct to ten decimals. (Use (5.1.15).)

h	$M(h)$	$T(h)$
0.8	0.77883 66846	0.75867 80454
0.4	0.77376 69772	0.76875 73650
0.2	0.77251 27162	0.77126 21711
0.1		0.77188 74437

The correct value, to six decimals, is 0.772096. Verify that in this example the error is approximately proportional to h^2 for both $M(h)$ and $T(h)$. We estimate the error in $T(0.1)$ to be $\frac{1}{3}6.26 \cdot 10^{-4} \leq 2.1 \cdot 10^{-4}$.

From the error analysis above we note that the error in the midpoint rule is roughly half the size of the error in the trapezoidal rule and of opposite sign. Hence it seems that the linear combination

$$S(h) = \frac{1}{3}(T(h) + 2M(h)). \tag{5.1.16}$$

should be a better approximation. This is indeed the case and (5.1.16) is equiva-

lent to **Simpson's rule**[2], one of the most famous classical formulas for numerical integration.

Another way to derive Simpson's rule is to approximate $f(x)$ by a piecewise polynomial of third degree. It is convenient to shift the origin to the midpoint of the interval and consider the integral over the interval $[x_i-h, x_i+h]$. From Taylor's formula we have

$$f(x) = f_i + (x-x_i)f_i' + \frac{(x-x_i)^2}{2}f_i'' + \frac{(x-x_i)^3}{3!}f_i''' + O(h^4),$$

where the remainder is zero for all polynomials of degree 3 or less. Integrating term by term, the integrals of the second and fourth term vanishes giving

$$\int_{x_i-h}^{x_i+h} f(x)\,dx = 2hf_i + 0 + \frac{1}{3}h^3 f_i'' + 0 + O(h^5).$$

Using $h^2 f_i'' = (f_{i-1} - 2f_i + f_{i+1}) + O(h^4)$ (see (4.7.5)) we have that

$$\begin{aligned}\int_{x_i-h}^{x_i+h} f(x)\,dx &= 2hf_i + \frac{1}{3}h(f_{i-1} - 2f_i + f_{i+1}) + O(h^5) \\ &= \frac{1}{3}h(f_{i-1} + 4f_i + f_{i+1}) + O(h^5),\end{aligned} \tag{5.1.17}$$

where the remainder term is zero for all third-degree polynomials. We now determine the error term for $f(x) = (x-x_i)^4$, which is

$$R_T = \frac{1}{3}h(h^4 + 0 + h^4) - \int_{x_i-h}^{x_i+h} x^4\,dx = (2/3 - 2/5)h^5 = \frac{4}{15}h^5.$$

It follows that an *asymptotic* error estimate is

$$R_T = h^5\frac{4}{15}\frac{f^{(4)}(x_i)}{4!} + O(h^6) = \frac{h^5}{90}f^{(4)}(x_i) + O(h^6).$$

A strict error estimate for Simpson's rule is more difficult to obtain. As for the midpoint formula the midpoint x_i can be considered as a double point of interpolation; see Problem 3. The general error formula (5.1.5) then gives

$$R(f) = \frac{1}{4!}\int_{x_{i-1}}^{x_{i+1}} f^{(4)}(\xi_x)(x-x_{i-1})(x-x_i)^2(x+x_{i+1})\,dx.$$

where $(x-x_{i-1})(x-x_i)^2(x+x_{i+1})$ has constant sign on $[x_{i-1}, x_{i+1}]$. If $2h$ is the length of the interval of integration Using the mean value theorem gives the error

$$-\frac{1}{90}f^{(4)}(\xi)h^5, \quad |\xi| < h. \tag{5.1.18}$$

[2]The English mathematician Thomas Simpson (1710–1761) is best known for his work on interpolation and quadrature. He also worked on probability theory.

The remainder can also be obtained from Peano's error representation It can be shown (see Stoer [32, p. 152 ff]) that for Simpson's rule

$$Rf = \int_{\mathbf{R}} f^{(4)}(u)K(u)\,du,$$

where the kernel equals

$$K(u) = -\frac{1}{72}(h-u)^3(3u+h)^2, \quad 0 \le u \le h,$$

and $K(u) = K(|u|)$ for $u < 0$, $K(u) = 0$ for $|u| > h$. This again gives (5.1.18).

In the **composite Simpson's formula** one divides the interval $[a, b]$ into an *even* number $n = 2m$ steps of length h, and use the formula (5.1.17) on each of m double steps, giving

$$\int_a^b f(x)\,dx = \frac{h}{3}(f_0 + 4U + 2E + f_n) + R_T, \tag{5.1.19}$$

where

$$U = f_1 + f_3 + \cdots + f_{n-1}, \qquad E = f_2 + f_4 + \cdots + f_{n-2}.$$

The remainder is

$$R_T = \sum_{i=0}^{m-1} \frac{h^5}{90} f^{(4)}(\xi_i) = \frac{(b-a)}{180} h^4 f^{(4)}(\xi), \qquad \xi \in [a, b]. \tag{5.1.20}$$

This shows that wee have gained *two orders of accuracy* compared to the trapezoidal rule, without using more function evaluations. This is why Simpson's rule is such a popular general-purpose quadrature rule.

5.1.3 Higher Order Newton–Cotes' Formulas

The classical Newton–Cotes' quadrature rules, are interpolatory rules obtained for $w(x) = 1$ and equidistant points in $[0, 1]$. There are two classes: **closed formulas**, where the end points of the interval belong to the nodes; **open formulas**, where all nodes lie strictly in the interior of the interval. The closed Newton–Cotes' formulas are usually written

$$\int_0^{nh} f(x)\,dx = h\sum_{j=0}^{n} w_j f(jh) + R_n(f) \quad w_j = w_{n-j}, \tag{5.1.21}$$

where, in principle, the weights w_i can be determined from (5.1.4). By (5.1.2) they satisfy

$$\sum_{j=0}^{n} h w_j = nh. \tag{5.1.22}$$

(Note that we here sum over $n+1$ points in contrast to our previous notation.) The closed Newton–Cotes' rule for $n = 1$ and $n = 2$ are equivalent to the trapezoidal rule and Simpson's rule, respectively.

In general it can be shown that the closed Newton–Cotes' formula integrate all polynomials of degree d exactly, where $d = n$ for n odd and $d = n + 1$ for n even. The extra accuracy for n even is, as in Simpson's rule, due to symmetry. For $n \leq 7$ the coefficients w_i are positive, but for $n = 8$ and $n \geq 10$ negative coefficients appear. Such formulas may still be useful, but since $\sum_{j=0}^{n} h|w_j| > nh$, they are less robust with respect to errors in the function values f_i.

Similarly the open Newton–Cotes' formulas are usually written as

$$\int_0^{nh} f(x)\,dx = h\sum_{i=1}^{n-1} w_i f(ih) + R_{n-1,n}(h), \quad w_{-j} = w_{n-j}.$$

The simplest open Newton–Cotes' formula for $n = 2$ is the midpoint rule with step size $2h$. The open formulas have order $d = n - 1$ for n even and $n - 2$ for n odd. For the open formulas negative coefficients occur already for $n = 4$ and $n = 6$.

The Peano kernels for both the open and the closed formulas can be shown to have constant sign (see Steffensen [31]). Thus the local truncation error can be written as

$$R_n(h) = c_{n,d}h^{d+1}f^{(d)}(\zeta), \quad \zeta \in [0, nh], \tag{5.1.23}$$

It is easily shown that the Peano kernels for the corresponding composite formulas also have constant sign.

The Newton–Cotes' closed formulas for $n \leq 6$ and open formulas for $n \leq 5$, with error terms, are given in Tables 5.1.1 and 5.1.2, respectively. Note that the sign of the error coefficients in the open rules are opposite the sign in the closed rules. Higher order Newton–Cotes' formulas are given in Abramowitz and Stegun [1, pp. 886–887],

Table 5.1.1. *The coefficients $w_i = Ac_i$ in the n-points closed Newton–Cotes' formulas.*

n	d	A	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_n
1	1	1/2	1	1						−1/12
2	3	1/3	1	4	1					−1/90
3	3	3/8	1	3	3	1				−3/80
4	5	2/45	7	32	12	32	7			−8/945
5	5	5/288	19	75	50	50	75	19		−275/12 096
6	7	1/140	41	236	27	272	27	236	41	−9/1400

We now show how the classical Newton–Cotes formulas for $w(x) = 1$ can be derived using the operator methods developed in Sec. 3.3. Let m, n be given integers and let h be a positive step size. In order to utilize the symmetry of the problem easier, we move the origin to the midpoint of the interval of integration. If we set

$$x_j = jh, \qquad f_j = f(jh), \quad j = -n/2 : 1 : n/2,$$

Table 5.1.2. *The coefficients $w_i = Ac_i$ in the n-points open Newton–Cotes' formulas.*

n	d	A	c_1	c_2	c_3	c_4	c_5		c_n.
2	1	2	1						1/24
3	1	3/2	1	1					1/4
4	3	4/3	2	−1	2				14/45
5	3	5/24	11	1	1	11			95/144
6	5	3/10	11	−14	26	−14	11		41/140
7	5	7/1440	611	−453	562	562	−453	611	5257/8640

the Newton–Cotes formula now reads

$$\int_{-mh/2}^{mh/2} f(x)\,dx = h \sum_{j=-n/2}^{n/2} w_j f_j + R_{m,n}(h), \quad w_{-j} = w_j. \tag{5.1.24}$$

Note that j, $n/2$ and $m/2$ are not necessarily integers. For a Newton–Cotes formula $n/2 - j$ and $m/2 - j$ are evidently integers. Hence $(m - n)/2$ is an integer too, but there may be other formulas, perhaps almost as good, where this is not the case. The coefficients $w_j = w_{j;m,n}$ are to be determined so that the remainder $R_{m,n}$ vanishes if $f \in \mathcal{P}_q$, with q as large as possible for given m, n. The left hand side of (5.1.24), divided by h, reads in operator form,

$$(e^{hDm/2} - e^{-hDm/2})(hD)^{-1} f(x_0),$$

which is an even function of hD. By (3.3.38), hD is an odd function of δ. It follows that the left hand side is an even function of δ, hence we can, for every m, write

$$(e^{hDm/2} - e^{-hDm/2})(hD)^{-1} \quad \mapsto A_m(\delta^2) = a_{1m} + a_{2m}\delta^2 + \ldots + a_{k+1,m}\delta^{2k} \ldots \tag{5.1.25}$$

We truncate after (say) δ^{2k}; the first neglected term is then $a_{k+2,m}\delta^{2k+2}$. We saw in Sec. 3.3.4 how to bring a truncated δ^2-expansion to $B(E)$-form

$$b_1 + b_2(E + E^{-1}) + b_3(E^2 + E^{-2}) + \ldots + b_k(E^k + E^{-k}).$$

by matrix multiplication with a matrix M of the form given in (3.3.45). By comparison with (5.1.24), we conclude that $n/2 = k$, that the indices j are integers, and that $w_j = b_{j+1}$ (if $j \geq 0$). *If m is even, this becomes a Newton–Cotes formula.* If m is odd, it may still be a useful formula, but it does not belong to the Newton–Cotes family, because $(m - n)/2 = m/2 - k$ is no integer.

If $n = m$ a formula is of the closed type. Its remainder term is the first neglected term of the operator series, truncated after δ^{2k}, $2k = n = m$ (and multiplied by h). Hence the remainder of (5.1.24) can be estimated by $a_{2+m/2}\delta^{m+2} f_0$.

or (better)

$$R_{m,m} \sim (a_{m/2+2}/m)H(hD)^{m+2}f_0.$$

where we call $H = mh$ the "bigstep".

If the integral is computed over $[a, b]$ by means of a sequence of "bigsteps", each of length H, an estimate of the *global* error has the same form, except that H is replaced by $b-a$, and f_0 is replaced by $\max_{x\in[a,b]} |f(x)|$. The exponent of hD in an error estimate that contains H or $b-a$, is known as the *global order of accuracy* of the method.

If $n < m$, a formula of the open type is obtained. Among the open formulas we shall only consider the case that $n = m-2$, which are the open Newton–Cotes formula. The operator expansion is truncated after δ^{m-2}, and we obtain

$$R_{m-2,m} \sim (a_{m/2+1}/m)H(hD)^{m}f_0.$$

Formulas with $n > m$ are rarely mentioned in the literature (except for $m = 1$). We do not understand why; it is rather common that an integrand has a smooth continuation outside the interval of integration.

Example 5.1.2.

The coefficients a_{im} in the expansion (5.1.25) can be computed by means of the Cauchy+FFT method. In this way extensive algebraic calculations are avoided[3]. It can be shown that the exact coefficients are rational numbers, though it is sometimes hard to estimate in advance the order of magnitude of the denominators. The algorithm must be used with judgment.

The coefficients are first obtained in floating point representation. The transformation to rational form is obtained by a continued fraction algorithm, described in Example 3.4.1.

For the case $m = 8$ the result reads,

$$A_8(\delta^2) = 8 + \frac{64}{3}\delta^2 + \frac{688}{45}\delta^4 + \frac{736}{189}\delta^6 + \frac{3956}{14175}\delta^8 - \frac{2368}{467775}\delta^{10} + \dots \qquad (5.1.26)$$

The closed integration formula becomes

$$\int_{-x_4}^{x_4} f(x)dx = \frac{4h}{14175}\Big(-4540f_0 + 10496(f_1 + f_{-1}) - 928(f_2 + f_{-2}) + 5888(f_3 + f_{-3)}) + 989(f_4 + f_{-4})\Big) + R, \qquad (5.1.27)$$

$$R \sim \frac{296}{467775}Hh^{10}f^{(10)}(x_0). \qquad (5.1.28)$$

It goes without saying that this is not how Newton and Cotes found their methods. Our method may seem complicated, but the Matlab programs for this are rather short, and to a large extent useful for other purposes. The computation of about 150 Cotes-coefficients and 25 remainders ($m = 2 : 14$), took less than two seconds on a PC. This includes the calculation of several alternatives for rational

[3]These could, however, be carried out using a system like Maple.

approximations to the floating-point results. For a small number of the 150 coefficients the judicious choice among the alternatives took, however, much more than 2 (human) seconds; this detail is both science and art.

It was mentioned that, *if m is odd*, (5.1.25) does not provide formulas of the Newton–Cotes family, since $(m-n)/2$ is no integer, nor are the indices j in (5.1.24) integers. So, the operator associated with the right hand side of (5.1.24) is of the form

$$c_1(E^{1/2}+E^{-1/2})+c_2(E^{3/2}+E^{-3/2})+c_3(E^{5/2}+E^{-5/2})+\dots.$$

If it is divided algebraically by $\mu=\frac{1}{2}(E^{1/2}+E^{-1/2})$, however, it becomes of the $B(E)$-form (say)

$$b'_1+b'_2(E+E^{-1})+b'_3(E^2+E^{-2})+\ldots+b_k(E^k+E^{-k}).$$

If m is odd we therefore expand

$$(e^{hDm/2}-e^{-hDm/2})(hD)^{-1}/\mu, \qquad \mu=\sqrt{1+\delta^2/4},$$

into a δ^2-series, with coefficients a'_j. Again this can be done numerically by the Cauchy+FFT method. For each m two truncated δ^2-series, one for the closed an one for the open case, are then transformed into $B(E)$-expressions numerically by means of the matrix M, as described above. The expressions are then multiplied *algebraically* by $\mu=\frac{1}{2}(E^{1/2}+E^{-1/2})$. We then have the coefficients of a Newton–Cotes formula with m odd.

The asymptotic error is

$$a'_{m/2+1}H(hD)^{m+1} \quad \text{and} \quad a'_{m/2-1}H(hD)^{m-1}$$

for the closed type, and open type, respectively ($2k=m-1$). The global orders of accuracy for Newton–Cotes methods with odd m are thus the same as for the methods, where m is one less.

5.1.4 Weighted Quadrature Rules

Newton–Cotes' quadrature rules consist of approximating the integrand by a polynomial and then integrate the polynomial exactly. Thus the accuracy depends on how well the function $f(x)$ can (locally) be approximated by a polynomial. A sufficient condition that the method converges as $h\to 0$ is that the integrand be continuous, but to get rapid convergence more is required.

If the integrand becomes infinite at a point, some modification is *necessary*. Even if some low-order derivative of the function is infinite at some point in or near the interval of integration, one *should* make such a modification. It is not uncommon that, when using a constant step-size, a single step taken close to a point where, for example, the derivative of the integrand is infinite, gives a larger error than all other steps combined.

It is often advantageous to consider quadrature rules of the form

$$\int_a^b f(x)w(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i). \tag{5.1.29}$$

Here $w(x) \geq 0$ is a given **weight function** (or density function) chosen so that $f(x)$ can be well approximated by a polynomial. To assure that the integral (5.1.29) is well defined when $f(x)$ is a polynomial, we assume in the following that the integrals

$$\mu_k = \int_a^b x^k w(x)\,dx, \quad k = 1, 2, \ldots, , \tag{5.1.30}$$

are defined for all $k \geq 0$, and $\mu_0 > 0$. The limits (a, b) of integration are here allowed to be infinite. Since the formula should be exact for $f(x) = 1$ it holds that

$$\mu_0 = \int_a^b 1 \cdot w(x)\,dx = \sum_{i=1}^{n} w_i. \tag{5.1.31}$$

The quantity μ_k is called the kth (ordinary) **moment** with respect to the weight function $w(x)$. For an interpolatory quadrature formula the weights are given by

$$w_i = \int_a^b \ell_i(x)w(x)\,dx. \tag{5.1.32}$$

Example 5.1.3.

Newton–Cotes formulas with weight functions other than $w(x) = 1$ are useful, e.g., when the integrand has a singularity. Such formulas can be derived by the method of undetermined coefficients. Consider the formula

$$\frac{1}{\sqrt{2h}} \int_0^{2h} x^{-1/2} f(x)\,dx \approx C_0 f(0) + C_1 f(h) + C_2 f(2h),$$

which is to be exact for any second-degree polynomial $f(x)$. Equating the left and right hand sides for $f(x) = 1, x, x^2$ we obtain

$$C_0 + C_1 + C_2 = 2, \qquad \frac{1}{2}C_1 + C_2 = \frac{2}{3}, \qquad \frac{1}{4}C_1 + C_2\frac{2}{5}.$$

This linear system is easily solved, giving $C_0 = 12/15$, $C_1 = 16/15$, $C_2 = 2/15$.

There are also other possibilities to treat integrals, where the integrand has a **singularity** or is "almost singular".

Example 5.1.4.

In the integral

$$I = \int_0^1 \frac{1}{\sqrt{x}} e^x\,dx$$

the integrand is infinite at the origin. By the substitution $x = t^2$ we get $I = 2\int_0^1 e^{t^2}\,dt$, which can be treated without difficulty.

Another possibility is to use integration by parts.

$$I = \int_0^1 x^{-1/2}e^x\,dx = 2x^{1/2}e^x\big|_0^1 - 2\int_0^1 x^{1/2}e^x\,dx$$
$$= 2e - 2\frac{2}{3}x^{3/2}e^x\big|_0^1 + \frac{4}{3}\int_0^1 x^{3/2}e^x\,dx = \frac{2}{3}e + \frac{4}{3}\int_0^1 x^{3/2}e^x\,dx.$$

The last integral has a mild singularity at the origin. If one wants high accuracy, then it is advisable to integrate by parts a few more times before the numerical treatment.

It is often profitable to investigate whether or not one can transform or modify the given problem in some way to make it more suitable for numerical integration. Below we give give some selected examples.

Example 5.1.5. (Simple Comparison Problem)

In $I = \int_{0.1}^1 x^{-3}e^x\,dx$ the integrand is infinite near the left end point. If we write

$$I = \int_{0.1}^1 x^{-3}\Big(1 + x + \frac{x^2}{2}\Big)\,dx + \int_{0.1}^1 x^{-3}\Big(e^x - 1 - x - \frac{x^2}{2}\Big)\,dx$$

the first integral can be computed analytically. The second integrand can be treated numerically. The integer and its derivatives are of moderate size. Note, however, the cancellation in the evaluation of the integrand.

For integrals over an infinite interval one can try some substitution which maps the interval $(0,\infty)$ to $(0,1)$, e.g., $t = e^{-x}$ of $t = 1/(1+x)$. However, in such cases one must be careful not to introduce an unpleasant singularity into the integrand instead.

Example 5.1.6.

Consider the integral $I = \int_0^\infty (1+x^2)^{-4/3}\,dx$. If one wants five decimal digits in the result then $\int_R^\infty$ is not negligible until $R \approx 10^3$. But one can expand the integrand in powers of x^{-1} and integrate term-wise,

$$\int_R^\infty (1+x^2)^{-4/3}\,dx = \int_R^\infty x^{-8/3}(1+x^{-2})^{-4/3}\,dx$$
$$= \int_R^\infty \Big(x^{-8/3} - \frac{4}{3}x^{-14/3} + \frac{14}{9}x^{-20/3} - \cdots\Big)$$
$$= R^{-5/3}\Big(\frac{3}{5} - \frac{4}{11}R^{-2} + \frac{14}{51}R^{-4} - \cdots\Big).$$

If this expansion is used, then one needs only apply numerical integration to the interval $[0, 8]$.

Example 5.1.7.

With the substitution $t = 1/(1+x)$ the integral in the previous example becomes

$$I = \int_0^1 (t^2 + (1-t)^2)^{-4/3} t^{2/3}\, dt.$$

The integrand now has an infinite derivative at the origin. This can be eliminated by making the substitution $t = u^3$, to get

$$I = \int_0^1 (u^6 + (1-u^3)^2)^{-4/3} 3u^4\, du,$$

which can be computed with, for example, a Newton–Cotes' method.

If the integrand is **oscillating**, then with ordinary integration methods one must choose a step size which is small with respect to the wave length; this is often an irritating limitation in many applications. The techniques previously mentioned (simple comparison problem, special integration formula, etc.) are sometimes effective in such situations. In addition, the following method can be used on integrals of the form

$$I = \int_0^\infty f(x) \sin(g(x))\, dx,$$

where $g(x)$ is an increasing function, and both $f(x)$ and $g(x)$ can be approximated by a polynomial. Set

$$I = \sum_{n=0}^{\infty} (-1)^n u_n, \qquad u_n = \int_{x_n}^{x_{n+1}} f(x) |\sin(g(x))|\, dx,$$

where $x_0, x_1, x_2, \ldots$ are the successive zeros of $\sin(g(x))$. The convergence of this alternating series can then be improved with the help of repeated averaging, see Sec. 3.2.1.

Review Questions

1. Why is a weight function $w(x) > 0$ included in many quadrature rules?
2. What is meant by the order of accuracy of a quadrature formula? Name three classical quadrature methods and give their order of accuracy.
3. What is meant by a composite quadrature rule? What is the difference between local and global error?
4. Give an account of the theoretical background of the classical Newton–Cotes rules.
5. Describe some possibilities for treating integrals, where the integrand has a **singularity** or is "almost singular".

Problems and Computer Exercises

1. (a) Derive the closed Newton–Cotes rule for $m = 3$,

$$I = \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3) + R_T, \qquad h = (b-a)/3,$$

also known as Simpson's (3/8)-rule.

(b) Derive the open Newton–Cotes rule for $m = 4$,

$$I = \frac{4h}{3}(2f_1 - f_2 + 2f_3) + R_T, \qquad h = (b-a)/4.$$

(c) Find asymptotic error estimates for the formulas in (a) and (b) by applying them to suitable polynomials.

2. (a) Show that Simpson's formula is the unique quadrature formula of the form

$$\int_{-h}^{h} f(x)\,dx \approx h(a_{-1}f(-h) + a_0 f(0) + a_1 f(h))$$

that is exact whenever $f \in \mathcal{P}_4$. Try to find several derivations of Simpson's formula, with or without the use of difference operators.

(b) Find the Peano kernel $K_2(u)$, such that $Rf = \int_{\mathbf{R}} f''(u)K_2(u)\,du$, and find the best constants c, p, such that

$$|Rf| \le ch^p \max|f''(u)|, \quad \forall f \in C^2[-h,h].$$

If you are going to deal with functions that are not in C^3, would you still prefer Simpson's formula to the trapezoidal rule?

3. The quadrature formula

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\,dx \approx h\big(af(x_{i-1}) + bf(x_i) + cf(x_{i+1})\big) + h^2 df'(x_i),$$

can be interpreted as a Hermite interpolatory formula with a *double point* at x_i. Show that $d = 0$ and that this formula is identical to Simpson's rule. Then show that the error can be written as

$$R(f) = \frac{1}{4!}\int_{x_{i-1}}^{x_{i+1}} f^{(4)}(\xi_x)(x - x_{i-1})(x - x_i)^2(x - x_{i+1})\,dx,$$

where $f^{(4)}(\xi_x)$ is a continuous function of x. Deduce the error formula for Simpson's rule. Setting $x = x_i + ht$, we get

$$R(f) = \frac{h^4}{24}f^{(4)}(\xi_i)\int_{-1}^{1}(t+1)t^2(t-1)h\,dt = \frac{h^5}{90}f^{(4)}(\xi_i).$$

4. A second kind of Newton–Cotes" open quadrature rule uses the midpoints of the equidistant grid $x_i = ih$, $i = 1 : n$, i.e.

$$\int_{x_0}^{x_n} f(x)\,dx = \sum_{i=1}^{n} w_i f_{i-1/2}, \quad x_{i-1/2} = \tfrac{1}{2}(x_{i-1} + x_i).$$

(a) For $n = 1$ we get the midpoint rule. Determine the weights in this formula for $n = 3$ and $n = 5$. (Use symmetry!)

(b) What is the order of accuracy of these two rules?

5. Derive Simpson's formula with end corrections, i.e. a formula of the form

$$\int_{-h}^{h} f(x)\,dx \approx h\big(af(-h) + bf(0) + af(h)\big) + h^2 c(f'(-h) - f'(h)),$$

that is exact for polynomials of degree five. What is the corresponding composite formula for the interval $[a, b]$ with $b - a = 2nh$?

6. Compute the integral

$$\frac{1}{2\pi}\int_0^{2\pi} e^{\frac{1}{\sqrt{2}}\sin x}\,dx$$

by the trapezoidal rule, using $h = \pi/2$ and $h = \pi/4$.

7. Compute the integral $\displaystyle\int_0^{\infty} (1 + x^2)^{-4/3}\,dx$ with five correct decimals. Expand the integrand in powers of x^{-1} and integrate term-wise over the interval $[R, \infty]$, for a suitable value of R. Then use a Newton–Cotes' rule on the remaining interval $[0, R]$.

8. Write a program for the derivation of a formula for integrals of the form $I = \int_0^1 x^{-1/2} f(x)\,dx$ that is exact for $f \in \mathcal{P}_n$ and uses the values $f(x_i)$, $i = 1 : n$, by means of the power basis.

(a) Compute the coefficients b_i for $n = 6 : 8$ with equidistant points, $x_i = (i - 1)/(n - 1)$, $i = 1 : n$. Apply the formulas to the integrals

$$\int_0^1 x^{-1/2} e^{-x}\,dx; \qquad \int_0^1 \frac{dx}{\sin\sqrt{x}}; \qquad \int_0^1 (1 - t^3)^{-1/2}\,dt.$$

In the first of the integrals compare with the result obtained by series expansion in Problem 3.1.1. In the last integral a substitution is needed for bringing it to the right form.

(b) Do the same for the case, where the step size $x_{i+1} - x_i$ grows proportionally to i; $x_1 = 0$; $x_n = 1$. Is the accuracy significantly different compared to (a), for the same number of points?

(c) Make some very small random perturbations of the x_i, $i = 1 : n$ in (a), (say) of the order of 10^{-13}. Of which order of magnitude are the changes in the coefficients b_i , and the changes in the results for the first of the integrals?

9. Propose a suitable plan (using a computer) for computing the following integrals, for $s = 0.5, 0.6, 0.7, \ldots, 3.0$:

(a) $\int_0^\infty (x^3 + sx)^{-1/2}\,dx$; (b) $\int_0^\infty (x^2+1)^{-1/2}e^{-sx}\,dx$, error $< 10^{-6}$;

(c) $\int_\pi^\infty (s+x)^{-1/3}\sin x\,dx$.

10. It is not true that any degree of accuracy can be obtained by using a Newton–Cotes' formula of sufficiently high order. To show this, Compute approximations to the integral

$$\int_{-4}^{4} \frac{dx}{1+x^2} = 2\tan^{-1} 4 \approx 2.6516353\ldots.$$

using the closed Newton–Cotes' formula with $n = 2, 4, 6, 8$. Which formula gives the smallest error?

11. For expressing integrals appearing in the solution of certain integral equations the following modification of the midpoint rule is often used:

$$\int_{x_0}^{x_n} K(x_j, x)y(x)\,dx = \sum_{i=0}^{n-1} m_{ij}y_{i+1/2},$$

where $y_{i+1/2} = y(\frac{1}{2}(x_i + x_{i+1}))$ and m_{ij} is the moment integral

$$m_{ij} = \int_{x_i}^{x_{i+1}} K(x_j, x)\,dx.$$

Derive an error estimate for this formula.

12. (a) Suppose that you have found a truncated δ^2-expansion, (say) $A(\delta^2) \equiv a_1 + a_2\delta^2 + \ldots + a_{k+1}\delta^{2k}$. Then an equivalent symmetric expression of the form $B(E) \equiv b_1 + b_2(E + E^{-1}) + \ldots + b_{k+1}(E^k + E^{-k})$ can be obtained as $b = M_{k+1}a$, where a, b are column vectors for the coefficients, and M_{k+1} is the $(k+1) \times (k+1)$ submatrix of the matrix M given in (3.2.45).
Use this for deriving (5.1.27) from (5.1.26). How do you obtain the remainder term? If you obtain the coefficients as decimal fractions, multiply them by 14175/4 in order to check that they agree with (5.1.27).

(b) Use Cauchy+FFT for deriving (5.1.26), and the open formula and the remainder for the same interval.

(c) Set $z_n = \nabla^{-1}y_n - \Delta^{-1}y_0$. We have, in the literature, seen the interpretation that $z_n = \sum_{j=0}^{n} y_j$ if $n \geq 0$. It seems to require some extra conditions to be true. Investigate if the conditions $z_{-1} = y_{-1} = 0$ are necessary and sufficient. Can you suggest better conditions? (The equations $\Delta\Delta^{-1} = \nabla\nabla^{-1} = 1$ mentioned earlier are assumed to be true.)

13. (a) Write a program for the derivation of quadrature formulas and error estimates according to Example 5.1.2 for $m = n-1, n, n+1$. Test the formulas and the error estimates for some m, n on some simple (though not too simple)

examples. Some of these formulas are listed in Handbook of Mathematical Functions [1, Sec. 25.4]. In particular, check the closed Newton–Cotes' 9-point formula ($n = 8$). .

(b) Sketch a program for the case that $h = 1/(2n + 1)$, with the computation of f at $2m$ symmetrical points.

(c) Abramowitz–Stegun [1, Sec. 25.4] gives several Newton–Cotes formulas of closed and open types, with remainders. Try to reproduce and extend their tables with techniques related to Example 5.2.2.

5.2 Quadrature Rules with Free Nodes

Previously we have assumed that all nodes x_i of the quadrature formula are given. A natural questions is whether we can do better by a judicious choice of the free nodes. This question is answered in the following theorem, which shows that by a careful choice of grid points the order of accuracy of the quadrature rule can substantially improved.

Theorem 5.2.1 (Gautschi [16] Theorem 3.2.1).

Let k be an integer such that $0 \le k \le n$. Consider the quadrature rule (5.1.1) *and let*

$$s(x) = (x - x_1)(x - x_2) \cdots (x - x_n) \tag{5.2.1}$$

be the corresponding **node polynomial**. *Then the quadrature rule has degree of exactness equal to $d = n + k - 1$, if and only if the following two conditions are satisfied:*

(a) *The quadrature rule* (5.1.1) *is interpolatory, i.e. the coefficients C_i are given by* (5.1.4).

(b) *The node polynomial satisfies*

$$\int_a^b p(x)s(x)w(x)\,dx = 0, \tag{5.2.2}$$

for all polynomials $p \in \mathcal{P}_k$.

Proof. We first prove the *necessity* of the conditions (a) and (b). Since the degree of exactness is $d = n+k-1 \ge n-1$, the condition (a) follows immediately. Further, for any $p \in \mathcal{P}_k$ the product $p(x)s(x)$ is in $\mathcal{P}_{n+k}$. Hence

$$\int_a^b p(x)s(x)w(x)\,dx = \sum_{j=1}^{n} w_k f(x_k)s(x_k) = 0.$$

since $s(x_k) = 0$, $k = 1 : n$, so that (b) holds.

To prove the *sufficiency*, let $p(x)$ be any polynomial of degree $n + k - 1$. Let $q(x)$ and $r(x)$ be the quotient and remainder, respectively, in the division

$$f(x) = q(x)s_n(x) + r(x).$$

Then $q(x)$ and r are polynomials of degree $k-1$ and $n-1$, respectively, and it holds that

$$\int_a^b p(x)w(x)\,dx = \int_a^b q(x)s_n(x)w(x)\,dx + \int_a^b r(x)w(x)\,dx.$$

Here the first integral is zero because of the orthogonality property of $s(x)$. For the second we have

$$\sum_{i=1}^n w_i p(x_i) = \sum_{i=1}^n w_i q(x_i)s_n(x_i) + \sum_{i=1}^n w_i r(x_i) = \sum_{i=1}^n w_i r(x_i),$$

since $s_n(x_i) = 0$, $i = 1:n$. But

$$\int_a^b r(x)w(x)\,dx = \sum_{i=1}^n w_i r(x_i),$$

since the weights were chosen such that the formula was interpolatory and therefore exact for all polynomials of degree $n-1$. □

In the previous section we derived Newton–Cotes' quadrature rules using Lagrange interpolation or operator series. We now outline another general technique, the method of undetermined coefficients, for determining approximate quadrature formulas of maximum order.

Let L be a linear functional and consider approximation formulas of the form

$$Lf \approx \tilde{L}f = \sum_{i=1}^p a_i f(x_i) + \sum_{j=1}^q b_j f(z_j), \tag{5.2.3}$$

where the x_i are p *given nodes*, while the z_j are q *free nodes*. The latter are to be determined together with the weight factors a_i, b_j. The altogether $p+2q$ parameters in the formula are to be determined, if possible, so that the formula becomes exact for all polynomials of degree less than $p+2q$.

We introduce the two node polynomials

$$r(x) = (x-x_1)\cdots(x-x_p), \qquad s(x) = (x-z_1)\cdots(x-z_q), \tag{5.2.4}$$

of degree p and q, respectively.

Let $\phi_1, \phi_2, \ldots, \phi_N$ be a basis of the space of polynomials of degree less than N. We assume that the quantities $L\phi_k$, $k = 1:p+2q$ are known. Then we obtain the *non-linear* system,

$$\sum_{i=1}^p \phi_k(x_i)a_i + \sum_{j=1}^q \phi_k(z_j)b_j = L\phi_k(x), \quad k = 1,2,\ldots,p+2q, \tag{5.2.5}$$

This is a non-linear system in z_j, but of a very special type. Note that the free nodes z_j appear in a symmetric fashion; the system (5.2.5) is invariant with respect

to permutations of the free nodes together with their weights. We therefore first ask for their **elementary symmetric functions**, i.e. for the coefficients g_j of the node polynomial

$$s(x) = \phi_{q+1}(x) - \sum_{j=1}^{q} s_j \phi_j(x) \tag{5.2.6}$$

that has the free nodes $z_1, z_2, \dots z_q$ as zeros. We change the basis to the set

$$\phi_1(x), \dots \phi_q(x), s(x)\phi_1(x), \dots, s(x)\phi_{p+q}(x).$$

In the system (5.2.5), the equations for $k = 1 : q$ will not be changed, but the equations for $k = 1 + q : p + 2q$ become,

$$\sum_{i=1}^{p} \phi_{k'}(x_i)s(x_i)a_i + \sum_{j=1}^{q} \phi_{k'}(z_j)s(z_j)b_j = L(s\phi_{k'}), \quad 1 \le k' \le p + q. \tag{5.2.7}$$

Here the second sum disappears since $s(z_j) = 0$, for all j. (This is the nice feature of this treatment!) Further by (5.2.6)

$$L(s\phi_{k'}) = L(\phi_{k'}\phi_{q+1}) - \sum_{j=1}^{q} L(\phi_{k'}\phi_j)s_j, \quad 1 \le k' \le p + q. \tag{5.2.8}$$

We thus obtain the following *linear* system for the computation of the $q + p$ quantities, s_j, and $A_i = s(x_i)a_i$:

$$\sum_{j=1}^{q} L(\phi_{k'}\phi_j)s_j + \sum_{i=1}^{p} \phi_{k'}(x_i)A_i = L(\phi_{k'}\phi_{q+1}), \quad k' = 1 : p + q. \tag{5.2.9}$$

The weights of the fixed nodes are $a_i = A_i/s(x_i)$. The free nodes z_j are then determined by finding the q roots of the polynomial

$$s(x) = \phi_{q+1}(x) - \sum_{j=1}^{q} s_j \phi_j(x) = 0.$$

(Methods for computing roots of a polynomial are given in Sec. 6.5.) Finally, with a_i and z_j known, the weights b_j are obtained by the solution of the first q equations of the system (5.2.5). which are linear in b_j.

Let $p = 0$, $[a, b] = [0, b]$, (b may be infinite) and consider the monomial basis. The reader is advised to verify that, when $p > 0$ the matrix becomes a kind of combination of a Hankel matrix and a Vandermonde matrix. In this case the condition number of of the linear system (5.2.5) increases exponentially with $p+2q$ and the free nodes and corresponding weights may become rather inaccurate. It is usually found, however, that unless the condition number is so big that the solution breaks down completely, the computed solution will satisfy equation (5.2.5) with a small residual. That is what really matters for the application of formula (5.2.3).

Example 5.2.1.

Consider the linear functional $L(f) = \int_0^1 f(x)\,dx$. Set $p = 0$, $q = 3$ and choose the monomial basis $\phi_i(x) = x^{i-1}$. Introducing the node polynomial

$$s(x) = (x - z_1)(x - z_2)(x - z_3) = x^3 - s_3x^2 - s_2x - s_1,$$

the linear system (5.2.8) becomes

$$\begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 1/4 \\ 1/5 \\ 1/6 \end{pmatrix}.$$

The exact solution is $s_1 = 1/20$, $s_2 = -3/5$, and $s_3 = 3/2$. The free nodes thus are the zeros of $s(x) = x^3 - 3x^2/2 + 3x/5 - 1/20$, which are $z_2 = 1/2$ and $z_{1,3} = 1/2 \pm \sqrt{3/20}$. The weights b_1, b_2, b_3 are then found by solving (5.2.5) for $k = 1 : 3$.

For the purpose of error estimation, we can add the two equations

$$\sum_{i=1}^{p} \phi_k(x_i)\, a_i + \sum_{j=1}^{q} \phi_k(z_j)\, b_j + (k-1)!c_{k-1} = L\phi_k, \qquad (5.2.10)$$
$$k = p + 2q + 1, p + 2q + 2,$$

The remainder term of the method is of the form $c_N(Lf_N - \tilde{L}f_N)$, where f_N is any monic polynomial of degree N. Normally $N = p + 2q$, but this is inadequate if $c_{p+2q} = 0$. This exceptional case actually happens, if a certain kind of symmetry is present. The formula is then more accurate than expected, and we take $N = p + 2q + 1$ instead. This is why c_k is to be computed for two values of k.

For the determination of the error constant we compute, according to the comments to (5.2.10), the difference between the right hand side and the left hand side of (5.2.9), and divide by $(k')!$, for $k' = p + q + 1$, $p + q + 2$.

From a pure mathematical point of view all bases are equivalent, but equation (5.2.5) may be better conditioned with some bases than with others, and this turns out to be an important issue when $p + 2q$ is large. The simplest choice of basis is

$$\phi_k(x) = x^{k-1}, \quad x \in (0, b),$$

(b may be infinite). For this choice the condition number of (5.2.5), will increase exponentially with $p + 2q$.

In the case $[a, b] = [-b, b]$, where the weight function $w(x)$ and the given nodes x_i are symmetrical with respect to the origin it holds that $L(\phi_k(x)) = 0$, when k is even. Then the weights a_i and b_i, and the free nodes z_j will also be symmetrically located. If $p = 2p'$ is even, the number of parameters will be reduced to $p' + q$ by the transformation

$$x = \sqrt{\xi}, \quad \xi \in [0, b^2].$$

Note that $w(x)$ will be replaced by $w(\sqrt{\xi})/\sqrt{\xi}$. If p is odd, one node is at the origin, and one can proceed in an analogous way. This should also reduce the condition number approximately to its square root, and it is possible to derive in a numerically stable way formulas with about twice as high order of accuracy as in the unsymmetric case.

5.2.1 Gauss–Christoffel Quadrature

By Theorem 5.2.1 if the n nodes in a quadrature formula are chosen so that the node polynomial $s(x) = (x - x_1)(x - x_2) \cdots (x - x_n)$ satisfies

$$\int_a^b p(x)s(x)w(x)\,dx = 0, \quad \forall\, p(x) \in \mathcal{P}_n, \tag{5.2.11}$$

then the corresponding interpolatory quadrature rule has the maximum possible order of accuracy $2n - 1$. These formulas are called **Gauss' quadrature** formulas associated with the weight function w. The construction of such quadrature rules is closely related to the theory of orthogonal polynomials. For the weight function $w(x) \equiv 1$ they were derived in 1814 by Gauss [11]. Formulas for more general weight functions were given by Christoffel[4] [5] in 1858, which is why these are referred to as **Gauss–Christoffel quadrature** formulas.

We denote by

$$(f, g) = \int_a^b f(x)g(x)w(x)\,dx, \tag{5.2.12}$$

the inner product with respect to the weight function $w(x) \geq 0$ and the interval $[a, b]$. The corresponding norm is $(f, f) = \|f\|_2^2$. This inner product has the important property that

$$(xf, g) = (f, xg). \tag{5.2.13}$$

We recall the assumption that $w(x) \geq 0$ is a weight function on $[a, b]$ such that moments

$$\mu_k = (x^k, 1) = \int_a^b x^k w(x)\,dx.$$

are defined for all $k \geq 0$, and $\mu_0 > 0$.

The zeros of these polynomials then determine the nodes in the corresponding Gaussian formula. The weights are then determined by integrating the elementary Lagrange polynomials (5.1.4)

$$w_i = \int_a^b \ell_i(x)w(x)\,dx, \quad \ell_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}.$$

In Sec. 5.2.3 we will discuss a more stable algorithm that determines the nodes and weights directly from the coefficients in the recurrence relation (**??**).

The condition (5.2.11) for the node polynomial can now be interpreted to mean that $s(x)$ is orthogonal to all polynomials in $\mathcal{P}_n$. We shall now prove some important results from the general theory of orthogonal polynomials.

[4] Elvin Bruno Christoffel (1829–1900) worked mostly in Strasbourg. He is best known for his work in geometry and tensor analysis, which Einsten later used in his theory of relativity.

Theorem 5.2.2.

The roots x_i, $i = 1 : n$, of the orthogonal polynomial polynomial φ_{n+1} of degree n, associated with the weight function $w(x) \geq 0$ on $[a, b]$, are real, distinct and contained in the open interval (a, b).

Proof. Let $a < x_1 < x_2 \cdots < x_m$, be the roots of φ_{n+1} of odd multiplicity, which lie in (a, b). At these roots φ_n changes sign and therefore the polynomial $q(x)\varphi_{n+1}$, where

$$q(x) = (x - x_1)(x - x_2) \cdots (x - x_m),$$

has constant sign in $[a, b]$. Hence,

$$\int_a^b \varphi_{n+1} q(x) w(x)\, dx > 0.$$

But this is possible only if the degree of $q(x)$ is equal to n. Thus $m = n$ and the theorem follows. □

Corollary 5.2.3.

If $x_1, x_2, \ldots, x_n$ are chosen as the n distinct zeros of the orthogonal polynomial φ_{n+1} of degree n in the family of orthogonal polynomials associated with $w(x)$, then the formula

$$\int_a^b f(x) w(x)\, dx \approx w_1 f_1 + w_2 f_2 + \ldots + w_n f_n, \tag{5.2.14}$$

$$w_i = \int_a^b \ell_i(x) w(x)\, dx, \tag{5.2.15}$$

is exact for polynomials of degree $2n - 1$.

Apart from having optimal degree of exactness equal to $2n - 1$, Gaussian quadrature rules have several important properties, which we now outline.

Theorem 5.2.4.

All weights in a Gaussian quadrature rule are real, distinct and positive.

Proof. Let

$$\ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}, \quad i = 1 : n,$$

be the Lagrange polynomials. Then the quadrature formula (5.2.14) is exact for $p(x) = (\ell_i(x))^2$, which is of degree $2(n-1)$. Further $\ell_i(x_j) = 0$, $j \neq i$, and therefore

$$\int_a^b (\ell_i(x))^2 w(x)\, dx = w_i (\ell_i(x_i))^2 = w_i.$$

Since $w(x) > 0$ it follows that $w_i > 0$. □

Gaussian quadrature formulas can also be derived by Hermite interpolation on the nodes x_k, *each counted as a double node*, and requiring that coefficients of the derivative terms should be zero. This interpretation gives a convenient expression for the error term in Gaussian quadrature.

Theorem 5.2.5.
The remainder term in Gauss' quadrature is given by the formula

$$\frac{f^{(2n)}(\xi)}{(2n)!}\int_a^b \Big[\prod_{i=1}^n (x-x_i)\Big]^2 w(x)\,dx = c_n f^{(2n)}(\xi), \quad a<\xi<b. \tag{5.2.16}$$

The constant c_n can be determined by applying the formula to some polynomial of degree $2n$.

Proof. Denote by $q(x)$ the polynomial of degree $2n-1$ which solves the Hermite interpolation problem (see Sec. 4.3.1)

$$q(x_i) = f(x_i), \qquad q'(x_i) = f'(x_i), \qquad i = 1:n.$$

The Gauss quadrature formula is exact for $q(x)$, and hence

$$\int_a^b q(x)w(x)\,dx = \sum_{i=1}^n w_i q(x_i) = \sum_{i=1}^n w_i f(x_i).$$

Thus

$$\sum_{i=1}^n w_i f(x_i) - \int_a^b f(x)w(x)\,dx = \int_a^b (q(x)-f(x))w(x)\,dx.$$

Using the remainder term (4.3.4) in Hermite interpolation gives

$$f(x)-q(x) = \frac{f^{(2n)}(\xi)}{(2n)!}(\varphi_n(x))^2, \quad \varphi_n(x) = \prod_{i=1}^n (x-x_i).$$

and the theorem now follows. □

5.2.2 Applications of Gauss Quadrature

For the uniform weight distribution $w(x) = 1$ on $[-1,1]$ the relevant orthogonal polynomials are the **Legendre polynomials** $P_n(x)$. As a historical aside, Gauss derived his quadrature formula by considering the continued fraction

$$\tfrac{1}{2}\ln\left(\frac{z+1}{z-1}\right) = \tfrac{1}{2}\int_{-1}^1 \frac{dx}{z-x} = \frac{1}{z-}\,\frac{1/3}{z-}\cdots \tag{5.2.17}$$

$$= \frac{1}{z} + \frac{1}{3z^3} + \frac{1}{5z^5} + \cdots \tag{5.2.18}$$

The nth convergent of this continued fraction is a rational function with a numerator of degree $n-1$ in z and denominator of degree n, which is the $(n-1,n)$ Padé approximation to the function. Decomposing this fraction in partial fractions the residues and the poles can be taken as nodes of a quadrature formula. The denominators are precisely the **Legendre polynomials**. Using the accuracy properties of the Padé approximants Gauss showed that the formula will have order $2n-1$. For more on this interesting connections between Padé approximants and orthogonal polynomials see Brezinski [3].

Since the weight distribution is symmetric about the origin the Legendre polynomials have the symmetry property

$$P_n(-x) = (-1)^n P_n(x).$$

They satisfy the three-term recurrence formula $P_0(x) = 1$, $P_1(x) = x$,

$$P_{n+1}(x) = \frac{2n+1}{n+1} x P_n(x) - \frac{n}{n+1} P_{n-1}(x), \quad n \geq 1. \tag{5.2.19}$$

giving

$$P_2(x) = \tfrac{1}{2}(3x^2-1), \qquad P_3(x) = \tfrac{1}{2}(5x^3-3x), \ldots \tag{5.2.20}$$

The Legendre polynomials have leading coefficient

$$A_n = \frac{1}{2^n n!} 2n(2n-1)(2n-2)\ldots(n+1).$$

and $\|P_n\| = 2/(2n+1)$.

The Legendre polynomials can also be defined by

$$P_0(x) = 1, \qquad P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}\big((x^2-1)^n\big), \qquad n = 1, 2, \ldots \tag{9.3.21}$$

Since $(x^2-1)^n$ is a polynomial of degree $2n$, $P_n(x)$ is a polynomial of degree n. The extreme values are

$$|P_n(x)| \leq 1, \quad x \in [-1, 1].$$

There seems to be no easy proof for this result; see Henrici [1964, p. 219].

Example 5.2.2.

Derive a two-point Gauss quadrature rule for $\int_{-1}^{1} f(x)\,dx$. Here $w(x) = 1$, and the relevant orthogonal polynomials are the Legendre polynomials $P_{m+1}(x)$. For $m = 1$ we have $P_2(x) = \frac{1}{2}(3x^2-1)$, and hence $x_0 = -3^{-1/2}$, $x_1 = 3^{-1/2}$. The weights can be determined by application of the formula to $f(x) = 1$ and $f(x) = x$, respectively, i.e.,

$$w_0 + w_1 = 2, \qquad -3^{-1/2} w_0 + 3^{-1/2} w_1 = 0,$$

with solution $w_0 = w_1 = 1$. Hence the formula

$$\int_{-1}^{1} f(x)\,dx \approx f(-3^{-1/2}) + f(3^{-1/2})$$

Table 5.2.1. *Abscissas and weight factors for Gauss–Legendre quadrature from Abramowitz–Stegun* [1, Table 25.4].

x_i	w_i
$n = 4$	
±0.33998 10435 84856	0.65214 51548 62546
±0.86113 63115 94053	0.34785 48451 37454
$n = 5$	
0.00000 00000 00000	0.56888 88888 88889
±0.53846 93101 05683	0.47862 86704 99366
±0.90617 98459 38664	0.23692 68850 56189
$n = 6$	
±0.23861 91860 83197	0.46791 39345 72691
±0.66120 93864 66265	0.36076 15730 48139
±0.93246 95142 03152	0.17132 44923 79170

Table 5.2.2. *Summary of Gaussian quadrature rules*

interval $[a, b]$	weight function $w(x)$	abscissas zeros of	polynomials
$[-1, 1]$	1	$P_n(x)$	Legendre
$[-1, 1]$	$(1-x^2)^{-1/2}$	$T_n(x)$	Chebyshev 1st kind
$[-1, 1]$	$(1-x)^\alpha(1+x)^\beta$	$J_n(x;\alpha,\beta)$	Jacobi
$[-1, 1]$	$(1-x^2)^{1/2}$	$U_n(x)$	Chebyshev, 2nd kind
$[0, \infty]$	e^{-x}	$L_n(x)$	Laguerre
$[-\infty, \infty]$	e^{-x^2}	$H_n(x)$	Hermite

is exact for polynomials of third degree.

In order to use the Gaussian quadrature rule the abscissas and weight factors must be known numerically. Note that for $w(x) = 1$ and the interval $[-1, 1]$ the abscissas are symmetric with respect to the origin. The two-point formulas was given above; for the three-point formula see Problem 1 below. In Table 5.2.1 we give abscissas and weights for some higher order Gauss–Legendre formulas using $n = m + 1$ points.

The **Jacobi polynomials** $J_n(x;\alpha,\beta)$ arise from the weight function

$$w(x) = (1-x)^\alpha(1+x)^\beta, \quad x \in [-1, 1], \quad \alpha, \beta > -1,$$

They are special cases of Gauss hypergeometric function $F(a, b, c : x)$

$$F(-n, \alpha + 1 + \beta + n, \alpha + 1; x).$$

(see (3.1.12)). The Jacobi polynomials are usually defined so that the coefficient A_n of x^n in $J_n(x; \alpha, \beta)$ is given by

$$A_n = \frac{1}{2^n n!} \frac{\Gamma(2n + \alpha + \beta + 1}{\Gamma(n + \alpha + \beta + 1}.$$

We obtain the Gauss–Legendre quadrature formula as the special case when $\alpha = \beta = 0$. Further the case $\alpha = \beta = -1/2$, which corresponds to $w(x) = 1/\sqrt{1 - x^2}$, give the Gauss–Chebyshev quadrature formula. These and some other important Gaussian quadrature rules are summarized in Table 5.2.2.

The above rules are given for the standard interval $[-1, 1]$. The corresponding formula for an integral over the interval $[a, b]$ is obtained by the change of variable $t = \frac{1}{2}((b - a)x + (a + b))$, which maps the interval $[a, b]$ onto $[-1, 1]$, so that

$$\int_a^b f(t)dt = \frac{b - a}{2} \int_{-1}^1 g(x)\, dx, \qquad g(x) = f\left(\frac{1}{2}\big((b - a)x + (a + b)\big)\right).$$

If $f(t)$ is a polynomial then $g(x)$ will be a polynomial of the same degree, since the transformation is linear. Hence the order of accuracy of the formula is not affected.

Two other important cases of Gauss quadrature rules are the following: For the weight function

$$w(x) = e^{-x}, \quad 0 \le x < \infty,$$

the corresponding orthogonal polynomials are the **Laguerre polynomials**, which satisfy

$$L_n(x) = e^x \frac{d^n}{dx^n}(x^n e^{-x}).$$

The **Hermite polynomials** are orthogonal with respect to the weight function

$$w(x) = e^{-x^2}, \quad -\infty < x < \infty.$$

They satisfy the recurrence relation $H_0(x) = 1$, $H_1(x) = 2x$,

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x).$$

The Hermite polynomials can also be defined by the formula.

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

It can be verified that these polynomials are identical to those defined by the recurrence relation.

In some situations we want some of the abscissas x_i in the quadrature formula to be fixed; the rest are to be chosen freely to maximize the order of accuracy. In the

most common cases the preassigned abscissas are at the endpoints of the interval. We consider here quadrature rules of the form

$$Lf = \sum_{i=1}^{n} w_i f(x_i) + \sum_{j=1}^{m} b_j f(z_j) + E(f) \tag{5.2.21}$$

where z_j, $j = 1 : m$ are fixed nodes and the x_i, a_i and b_j are to be determined. By a generalization of Theorem 5.2.5 the remainder term is given by the formula

$$E(f) = \frac{f^{(2n+m)}(\xi)}{(2n)!} \int_a^b \prod_{i=1}^{m}(x - z_i)\Big[\prod_{i=1}^{n}(x - x_i)\Big]^2 w(x)\,dx, \quad a < \xi < b. \tag{5.2.22}$$

In **Gauss–Lobatto** quadrature $m = 2$, and both endpoints are used as abscissas, $z_1 = a$, $z_2 = b$. Taking $[a, b] = [-1, 1]$ and and the weight function $w(x) = 1$, the quadrature formula has the form

$$\int_{-1}^{1} f(x)\,dx = b_1 f(-1) + b_2 f(1) + \sum_{i=1}^{n} w_i f(x_i) + E_L. \tag{5.2.23}$$

where

$$b_1 = b_2 = 2/((n + 2)(n + 1)).$$

The remaining n abscissas are the zeros $P'_{n+1}(x)$, where $P_n(x)$ denotes the Legendre polynomial. They lie symmetric with respect to the origin. The corresponding weights are given by

$$w_i = b_1/(P_{n+1}(x_i))^2,$$

and satisfy $w_i = w_{n+1-i}$. Because two points are fixed we lose two degrees of accuracy and the Lobatto rule (5.2.23) is exact only for polynomials of order $2m-1$. If $f(x) \in C^{2m}[-1, 1]$ then the error term is given by

$$E_L(f) = -\frac{(n + 2)(n + 1)^3 2^{2n+3}(n!)^4}{(2n + 3)[(2n + 2)!]^3} f^{(2n+2)}(\xi), \quad \xi \in (-1, 1). \tag{5.2.24}$$

Nodes and weights for Lobatto quadrature are found in Abramowitz–Stegun [1, Table 25.6].

Example 5.2.3.

The simplest Gauss–Lobatto rule is Simpson's rule with one interior node. Taking $n = 2$ the interior nodes are the zeros of $\phi_2(x)$, where

$$\int_{-1}^{1} (1 - x^2)\phi_2(x)p(x)\,dx = 0, \quad \forall p \in P_2.$$

Thus, ϕ_2 is, up to a constant factor, the Jacobi polynomial $J_2(x, 1, 1) = (x^2 - 1/5)$. Hence the interior nodes are $\pm 1/\sqrt{5}$ and the quadrature formula becomes

$$\int_{-1}^{1} f(x)\,dx = \frac{1}{6}(f(-1) + f(1)) + \frac{5}{6}(f(-1/\sqrt{5}) + f(1/\sqrt{5})) + R(f), \tag{5.2.25}$$

where $R(f) = 0$ for $f \in P_6$.

In **Gauss–Radau** quadrature rules one of the endpoints ± 1 is taken as abscissa, $z_0 = -1$, say. The quadrature formula has the form

$$\int_{-1}^{1} f(x)\,dx = \frac{2}{(n+1)^2} f(-1) + \sum_{i=1}^{n} w_i f(x_i) + E_{R1}. \tag{5.2.26}$$

The n free abscissas are the zeros of

$$\frac{P_n(x) + P_{n+1}(x)}{x - 1},$$

where $P_m(x)$ are the Legendre polynomials. The corresponding weights are given by

$$w_i = \frac{1}{(n+1)^2} \frac{1 - x_i}{(P_n(x_i))^2}.$$

The Gauss–Radau quadrature rule is exact for polynomials of order $2n + 1$. If $f(x) \in C^{2m-1}[-1, 1]$ then the error term is given by

$$E_{R1}(f) = \frac{(n+1)2^{2n+1}}{[(2n+1)!]^3} (n!)^4 f^{(2n+1)}(\xi_1), \quad \xi_1 \in (-1, 1). \tag{5.2.27}$$

A similar formula can be obtained with the fixed point $+1$ by making the substitution $t = -x$. From the error term (5.2.22) it follows that if the derivative $f^{(n+1)}(x)$ has constant sign in $[a, b]$, then the error will have opposite sign. This can be used to obtain lower and upper bounds for the true integral.

A drawback with Gaussian rules is that as we increase the order of the formula all interior abscissas change, except that at the origin. Hence we cannot use the function values computed for the lower order formula. For this reason Kronrod [21] considered the following problem: Given an n-point Gaussian quadrature rule

$$G_n \approx \sum_{i=0}^{n-1} w_i f(x_i),$$

find a new formula using the n old abscissas x_i and $n + 1$ new abscissas y_i

$$K_{2n+1} \approx \sum_{i=0}^{n-1} A_i f(x_i) + \sum_{i=0}^{n} B_i f(y_i).$$

The new abscissas and the weights A_i and B_i are to be chosen so that the rule K_{2n+1} is exact for polynomials of degree $3n + 1$.

The two rules (G_n, K_{2n+1}) are called a **Gauss–Kronrod** pair. Note that the number of new function evaluations are the same as for the Gauss rule G_{n+1}. The error can be estimated by the difference $|G_n - K_{2n+1}|$, but this usually severely overestimates the error.

Gauss–Kronrod rules is one of most effective methods for calculating integrals. Often one takes $n = 7$ and uses the Gauss–Kronrod pair (G_7, K_{15}), together with the realistic but still conservative error estimate $(200|G_n - K_{2n+1}|)^{1.5}$, see Kahaner, Moler, and Nash [20].

A Kronrod extension of the Gauss–Lobatto rule (5.2.25) has been given by Gander and Gautschi [9]:

$$\int_{-1}^{1} f(x)\,dx = \frac{11}{210}(f(-1)+f(1)) + \frac{72}{245}(f(-\sqrt{2/3})+f(\sqrt{2/3})) + \frac{125}{294}(f(-1/\sqrt{5})+f(1/\sqrt{5})) + \frac{16}{35}f(0)) + R(f). \quad (5.2.28)$$

This rule is exact for all $f \in \mathcal{P}_{10}$. Note that the Kronrod points $\pm\sqrt{2/3}$ and 0 interlace the previous nodes.

5.2.3 Matrix Formulas Related to Gauss Quadrature

We collect here some classical results of Gauss, Christoffel, Chebyshev, Stieltjes and others, with a few modern aspects and a notations appropriate for our purpose.

Let $\{p_1, p_2, \ldots, p_n\}$ be a basis for the space $\mathcal{P}_n$, of polynomials of degree $n-1$, where p_j be a polynomial of exact degree $j-1$. We introduce the row vector

$$\pi(x) = [p_1(x),\, p_1(x), \ldots, p_n(x)], \quad (5.2.29)$$

containing these basis functions. The **modified moments** with respect to the basis $\pi(x)$ are

$$\nu_k = (p_k, 1) = \int_a^b p_k(x)w(x)\,dx, \quad k = 1:n, \quad (5.2.30)$$

We define the two symmetric matrices

$$G = \int \pi(x)^T\pi(x)w(x)\,dx, \qquad \hat{G} = \int x\pi(x)^T\pi(x)w(x)\,dx. \quad (5.2.31)$$

associated with the basis defined by π. Here G is the **Gram matrix**[5] with elements $g_{ij} = (p_i, p_j) = (p_j, p_i)$,

$$G = \begin{pmatrix} (p_1,p_1) & (p_1,p_2) & \ldots & (p_1,p_n) \\ (p_2,p_1) & (p_2,p_2) & \ldots & (p_2,p_n) \\ \vdots & \vdots & \ddots & \vdots \\ (p_n,p_1) & (p_n,p_2) & \ldots & (p_n,p_n) \end{pmatrix}. \quad (5.2.32)$$

Two particularly interesting bases are the power basis and the orthonormal basis defined, respectively, by

$$\theta(x) = (1, x, x^2, \ldots, x^{n-1}), \quad (5.2.33)$$

$$\varphi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_n(x)), \quad (5.2.34)$$

[5] Jørgen Pedersen Gram (1850–1916) graduated from Copenhagen University and then worked as company director for a life insurance company. He introduced the Gram determinant in connection with his study of linear independence and his name is also associated with Gram–Schmidt orthogonalization.

where the components of φ are orthonormal polynomials with respect to the weight function w.

Example 5.2.4.

For the power basis $\theta(x)$ we have $g_{ij} = (x^{i-1}, x^{j-1}) = \mu_{i+j-2}$. So the matrices G and $\hat{G}$ become Hankel matrices,

$$G = \begin{pmatrix} \mu_0 & \mu_1 & \cdots & \mu_{n-1} \\ \mu_1 & \mu_2 & \cdots & \mu_n \\ \vdots & \vdots & \cdots & \vdots \\ \mu_{n-1} & \mu_n & \cdots & \mu_{2n-2} \end{pmatrix}, \qquad \hat{G} = \begin{pmatrix} \mu_1 & \mu_2 & \cdots & \mu_n \\ \mu_2 & \mu_3 & \cdots & \mu_{n+1} \\ \vdots & \vdots & \cdots & \vdots \\ \mu_n & \mu_{n+1} & \cdots & \mu_{2n-1} \end{pmatrix}.$$

In particular, for $w(x) \equiv 1$, and $[a, b] = [0, 1]$ we have $\mu_k = \int_0^1 x^{k-1}\, dx = 1/k$ and G is the notoriously ill-conditioned *Hilbert matrix*, for which the spectral condition number grows like $0.014 \cdot 10^{1.5n}$.

Let u, v, be two polynomials in $\mathcal{P}_n$ and set

$$u(x) = \pi(x)u_\pi, \qquad v(x) = \pi(x)v_\pi,$$

where u_π, v_π, are column vectors with the coefficients in the representation of u, v with respect to the basis defined by π. Note that $(u, v) = u_\pi^T G v_\pi$. For $u = v \neq 0$ we find that

$$u_\pi^T G u_\pi = (u, u) > 0,$$

hence the Gram matrix G is positive definite. (The matrix $\hat{G}$ is, however, usually indefinite.)

A polynomial of degree n that is orthogonal to all polynomials of degree less than n can be written in the form

$$\phi_{n+1}(x) = xp_n(x) - \pi(x)c_n, \quad c_n \in \mathbf{R}^n, \tag{5.2.35}$$

Here c_n is determined by the linear equations

$$-\int \pi(x)^T \pi(x) c_n w(x)\, dx + \int x\pi(x)^T p_n(x) w(x)\, dx = 0,$$

or in matrix form

$$Gc_n = \hat{g}_n, \tag{5.2.36}$$

where $\hat{g}_n$ is the last column of the matrix $\hat{G}$. Further, there are coefficients c_{kj} depending on the basis only, such that

$$xp_j(x) = \sum_{k=1}^{j+1} c_{k,j} p_k(x), \quad j = 1 : n-1.$$

Together with (5.2.35) this can be summarized in the vector equation

$$x\pi(x) = \pi(x)(C, c_n) + (0, 0, \ldots, \phi_{n+1}(x)). \tag{5.2.37}$$

Here $C \in \mathbf{R}^{n\times(n-1]}$ is an upper Hessenberg matrix, which depends on the basis only, while c_n also depends on the weight function. If the basis $\pi(x)$ is some family of orthogonal polynomials (with respect to another weight function than w) C is a tridiagonal matrix, obtained by means of the three-term recurrence relation for this family.

After multiplication of (5.2.37) by $\pi(x)^T w(x)$ and integration we obtain by (5.2.31)

$$G\overline{C} = \hat{G}, \quad \overline{C} = (C, c_n). \tag{5.2.38}$$

where the last column of this equation is the same as equation (5.2.36). Let G^*, C^* be defined like G, C, with n increased by one. Note that G and C are principal submatrices of G^* and C^*. Then $\hat{G}$ equals the n first rows of the product G^*C^*. So no integrations are needed for g_n, except for the matrix G.

Theorem 5.2.6.

Denote by R the matrix of coefficients of the expansions of the general basis functions $\pi(x) = [p_1(x), p_1(x), \ldots, p_n(x)]$ into the orthonormal basis polynomials, i.e.

$$\pi(x) = \varphi(x)R. \tag{5.2.39}$$

Then $G = R^T R$, i.e. R is the upper triangular Cholesky factor of the Gram matrix G. Note that this factorization up to the mth row is the same for all $n \geq m$. Further $\hat{G} = R^T JR$, where J is a symmetric tridiagonal matrix.

Proof. R is evidently an upper triangular matrix. Further, we have

$$\begin{aligned} G = \int \pi(x)^T \pi(x) w(x)\, dx &= \int R^T \varphi(x)^T \varphi(x) R w(x)\, dx \\ &= R^T I R = R^T R, \end{aligned}$$

since the elements of $\varphi(x)$ is an orthonormal system. This shows that R is the Cholesky factor of G. We similarly find that

$$\hat{G} = R^T JR, \quad J = \int x\varphi(x)^T \varphi(x) w(x)\, dx,$$

so J clearly is a symmetrical matrix. J is a particular case of $\hat{G}$ and from (5.2.38) and $G = I$ it follows that $J = \overline{C}$, a Hessenberg matrix. Hence J is a symmetric tridiagonal matrix. □

From (5.2.38) and Theorem 5.2.6 it follows that

$$\hat{G} = G\overline{C} = R^T R\overline{C} = R^T JR,$$

Since R is nonsingular we have $R\overline{C} = JR$, or

$$J = R\overline{C}R^{-1}. \tag{5.2.40}$$

It follows that, for every choice of basis, the spectrum of $\overline{C}$ equals the spectrum of J. We shall see that it is equal to the set of zeros of the orthogonal polynomial ϕ_n.

In particular, for the power basis $p_j(x) = x^{j-1}$, the Hessenberg matrix C is a **shift matrix**; the only non-zero elements are ones in the first main subdiagonal. Further, with $c_n^T = (a_1, a_2, \ldots, a_n)$, (5.2.35) reads

$$\phi_{n+1}(x) = x^n - \sum_{k=1}^{n} a_k x^{k-1},$$

and

$$\overline{C} = \begin{pmatrix} 0 & & & & a_1 \\ 1 & 0 & & & a_2 \\ & 1 & \ddots & & \vdots \\ & & \ddots & 0 & a_{n-1} \\ & & & 1 & a_n \end{pmatrix} \in \mathbf{R}^{n\times n},$$

which (after a permutation of rows and columns) is the companion matrix of the polynomial $\phi_{n+1}(x)$ (see Sec. 6.5.1). *Thus the eigenvalues λ_j, $j = 1 : n$, of $\overline{C}$ are the zeros of $\phi_{n+1}(x)$*, and hence the nodes for the Gauss–Christoffel quadrature formula. The row eigenvector corresponding to λ_j is

$$\theta(\lambda_j) = (1, \lambda_j, \lambda_j^2, \ldots, \lambda_j^{n-1}), \tag{5.2.41}$$

i.e. it holds that

$$\theta(\lambda_j)\overline{C} = \lambda_j \theta(\lambda_j), \quad j = 1 : n. \tag{5.2.42}$$

This yields a diagonalization of $\overline{C}$, since, by the general theory of orthogonal polynomials (see Theorem 5.2.4) the roots are simple roots, located in the interior of the smallest interval that contains the weight distribution.

To summarize, we have shown that if C and the Gram matrix G are known, then c_n can be computed by performing the Cholesky decomposition $G = R^T R$ and then solving $R^T R c_n = \hat{g}_n$ for c_n. The zeros of $\phi_{n+1}(x)$ are then equal to the eigenvalues of $\overline{C} = (C, c_n)$, or equivalently the eigenvalues of the symmetric tridiagonal matrix $J = R\overline{C}R^{-1}$. *This is true for any basis $\pi(x)$.* Note that J can be computed by solving the matrix equation $JR = R\overline{C}$ or

$$R^T J = (R\overline{C})^T. \tag{5.2.43}$$

Here R^T is a lower triangular matrix and the right hand side a lower Hessenberg matrix. This and the tridiagonal structure of J considerably simplifies the calculation of J.

For the power basis $\theta(x)$ we saw in Example 5.2.4 that G is a Hankel matrix. Hankel matrices play an important role in the classical theory of orthogonal polynomials, Gauss–Christoffel quadrature, the moment problem, continued fractions, etc. They are less interesting for practical computations since the condition number of H increases rapidly with n. This is due to the by now familiar fact that, when n is large, x^n can be accurately approximated by a polynomial of lower degree. The power basis is thus not a good basis for spaces of polynomials. Similarly the moments for the power basis are not in general a good starting point for the numerical computation of the matrix J.

In particular, for the orthonormal basis, for which $G = I$, and $\hat{G} = G^{-1}\hat{G} = J$, we obtain

$$\varphi(\lambda_j)J = \lambda_j\varphi(\lambda_j), \quad j = 1:n. \tag{5.2.44}$$

where

$$J = \begin{pmatrix} \beta_1 & \gamma_1 & & & 0 \\ \gamma_1 & \beta_2 & \gamma_2 & & \\ & \gamma_2 & & \ddots & \\ & & \ddots & \ddots & \gamma_{n-1} \\ 0 & & & \gamma_{n-1} & \beta_n \end{pmatrix}, \tag{5.2.45}$$

is a symmetric tridiagonal **Jacobi matrix** with nonzero off-diagonal elements. It is well known from linear algebra that such a matrix has n real distinct eigenvalues. Further, the eigenvectors can always be chosen mutually orthogonal.

Setting

$$\Phi = (\varphi(\lambda_1)^T, \ldots, \varphi(\lambda_n)^T), \qquad \Phi_{ij} = (\phi_{i-1}(\lambda_j), \qquad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n),$$

we obtain by (5.2.44) and the symmetry of J the important formula

$$J\Phi = \Phi\Lambda. \tag{5.2.46}$$

It also follows from (5.2.44) that the equation

$$x\varphi(x)^T = J\varphi(x)^T + \gamma_n\phi_{n+1}(x)e_n, \quad e_n = (0, \ldots, 0, 1)^T, \tag{5.2.47}$$

where γ_n is to be chosen so that $\|\phi_{n+1}\| = 1$, holds when $x = \lambda_j$, and $\phi_{n+1}(\lambda_j) = 0$, $j = 1:n$. Since the degree of φ is less than n, it is easily shown that the *equation* (5.2.47) *holds for all* x. As a by-product we obtain the important three term recurrence (**??**)

Let V be an orthogonal matrix that diagonalizes J, i.e.

$$JV = V\Lambda, \quad V^TV = VV^T = I,$$

where Λ is the diagonal in (5.2.46). It follows that $V = \Phi D$ for some diagonal matrix $D = \operatorname{diag}(d_i)$, and

$$V = \Phi D^2\Phi^T = VV^T = I,$$

that is

$$\sum_{k=1}^{n} \phi_i(\lambda_k)d_k^2\phi_j(\lambda_k) = \delta_{ij} = (\phi_i, \phi_j), \quad i, j = 1:n.$$

This equality holds also for $i = n+1$, because $\phi n + 1(\lambda_k) = 0$, for all k, and $(\phi_{n+1}, \phi_j) = 0$, $j = 1:n$.

Since every polynomial p of degree less than $2n$ can be expressed as a linear combination of polynomials of the form $\phi_i\phi_j$ (in infinitely many ways) it follows that

$$\sum_{k=1}^{n} d_k^2 p(\lambda_k) = \int p(x)w(x)\,dx, \tag{5.2.48}$$

for any polynomial p of degree less than $2n$. This yields the **Gauss–Christoffel quadrature rule**:

$$\int f(x)w(x)\,dx = \sum_{k=1}^{n} d_k^2 f(\lambda_k) + R, \tag{5.2.49}$$

where $R = \int (f(x) - p(x))w(x)\,dx$, *for any polynomial* p *of degree less than* $2n$, *such that* $p(\lambda_k) = f(\lambda_k)$, $k = 1 : n$.

The familiar form for the remainder term

$$R = k_n f^{(2n)}(\xi)/(2n)!, \tag{5.2.50}$$

is obtained by choosing a Hermite interpolation polynomial for p and then applying the mean value theorem. The constant k_n is independent of f. The choice $f(x) = A_n^2 x^{2n} + \cdots$ gives $k_n = A_n^{-2}$. A recurrence relation for the leading coefficient A_j is obtained by (**??**). We obtain

$$A_0 = \mu_0^{-1/2}, \quad A_{k+1} = A_k/\gamma_k. \tag{5.2.51}$$

The mean value form for R may be inappropriate, when the interval is infinite. Some other estimate of the above integral for R may then be more adequate.

A simple formula for the weights d_k^2, due to Golub and Welsch [18], is obtained by matching the first rows of the equality $V = \Phi D$. Since the elements in the first row are all equal to the constant $\phi_1 = \mu_0^{-1/2}$, we obtain

$$e_1^T V = \mu_0^{-1/2} d^T, \qquad d_k^2 = \mu_0 v_{1,k}^2, \quad k = 1 : n. \tag{5.2.52}$$

The well known fact that the weights are positive and their sum equals μ_0, follows immediately from this simple formula for the weights. We summarize these results in the following theorem:

Theorem 5.2.7.

Let J *be the symmetric tridiagonal* $n \times n$ *matrix that contains the coefficients in the three term recurrence relation for the orthonormal system of polynomials associated with the weight function* $w(x) > 0$. *Let* f *be an analytic function in a domain that contains the spectrum of* J.

Then the following concise formula, is exact when f *is a polynomial of degree less than* $2n$,

$$\frac{1}{\mu_0} \int f(x)w(x)\,dx \approx e_1^T V^T f(\Lambda) V e_1, \tag{5.2.53}$$

where $f(\Lambda) = \operatorname{diag}(f(\lambda_1, \ldots, f(\lambda_n)$.

Proof. The result follows from the Gauss–Christoffel rule (5.2.49) and (5.2.52). □

When the three-term recurrence relation for the orthonormal polynomials associated with the weight function $w(x)$ is known, the Gauss–Christoffel rule can

elegantly be obtained as follows. The eigenvalues of J are the nodes of the Gauss–Christoffel rule and the weights are obtained from (5.2.52) as the first components of the corresponding eigenvectors.. These quantities can be computed in a stable and efficient way by the QR-algorithm; see Volume II, Sec. 9.7.4. We remark that Golub [17] has shown how to extend this scheme to the computation of nodes and weights for Gauss–Radau and Gauss–Lobatto quadrature rules.

When the coefficients in the three-term relation cannot be obtained by theoretical analysis or numerical computation, we consider the matrices $\overline{C}$ and G as given data about the basis and weight function. As described above R, c_n, and J can then be computed by means of (5.2.40). The nodes and weights are then computed according to the previous case. Note that R and J are determined simultaneously for all $k \leq n$; just take the submatrices of the largest ones.

The computations are most straightforward for the power basis, i.e. with the moments of the weight function as the initial data. Unfortunately, the condition number of this problem increases rapidly with n, which results in inaccurate nodes and weights. Nevertheless, as long as the Choleski factorization of the Gram matrix G does not break down because of a negative pivot, the values of the integral give by the Gauss–Christoffel formula may be much more accurate than the nodes and weights obtained.

5.2.4 Symmetric Weight Functions

In many important cases the weight function $w(x)$ is symmetric about the origin. Then the moments of odd order are zero, and the orthogonal polynomials of odd (even) degree are odd (even) functions. The eigenvalues will appear in pairs, $\pm\lambda_k$. If n is odd, there is also a simple zero eigenvalue. The weights are symmetric so that the weights corresponding to the two eigenvalues $\pm\lambda_i$ are the same.

We shall see that in the symmetric case the eigenvalue problem for the tridiagonal matrix $J \in \mathbf{R}^{n\times n}$ can be reduced to a singular value problem for smaller bidiagonal matrix B, where

$$B \in \begin{cases} \mathbf{R}^{n/2\times n/2}, & \text{if } n \text{ even;} \\ \mathbf{R}^{(n+1)/2\times(n-1)/2}, & \text{if } n \text{ odd.} \end{cases}$$

We permute rows and columns in J, by an odd-even permutation, e.g., if $n = 7$ then $(1, 2, 3, 4, 5, 6, 7) \mapsto (1, 3, 5, 7, 2, 4, 6)$, and

$$\tilde{J} = T^{-1}JT = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}, \quad B = \begin{pmatrix} \beta_1 & 0 & 0 \\ \beta_2 & \beta_3 & 0 \\ 0 & \beta_4 & \beta_5 \\ 0 & 0 & \beta_6 \end{pmatrix},$$

where T be the permutation matrix effecting the odd-even permutation. Then, if the orthogonal matrix V diagonalizes J, i.e. $J = V\Lambda V^T$, then $\tilde{V} = T^{-1}V$, diagonalizes $\tilde{J} = T^TJT$, i.e. $\tilde{J} = T^{-1}JT = T^{-1}V\lambda V^TT$. Note that the first row of V is just a permutation of $\tilde{V}$. We can therefore substitute $\tilde{V}$ for V in equation (5.2.52) that gives the weights in the Gauss–Christoffel formula.

The following relationship between the SVD and a Hermitian eigenvalue problem, exploited by Lanczos [22, Chap. 3] can easily be verified.

Theorem 5.2.8.
Let the singular value decomposition of $B \in \mathbf{R}^{m\times n}$ $(m \geq n)$ be $B = P\Sigma Q^T$, where

$$\Sigma = \operatorname{diag}(\Sigma_1,\ 0), \quad \Sigma_1 = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n),$$

and

$$P = (P_1,\ P_2) \in \mathbf{C}^{m\times m}, \quad P_1 \in \mathbf{C}^{m\times n}, \quad Q \in \mathbf{C}^{n\times n}.$$

Then the symmetric matrix $C \in \mathbf{R}^{(m+n)\times(m+n)}$ has the eigendecomposition

$$C = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} = V \begin{pmatrix} \Sigma_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -\Sigma_1 \end{pmatrix} V^T, \tag{5.2.54}$$

where $V \in$ is orthogonal

$$V = \frac{1}{\sqrt{2}} \begin{pmatrix} P_1 & \sqrt{2}\,P_2 & P_1 \\ Q & 0 & -Q \end{pmatrix}^T. \tag{5.2.55}$$

Hence the eigenvalues of C are $\pm\sigma_1, \pm\sigma_2, \ldots, \pm\sigma_r$, and zero repeated $(m-n)$ times.

The QR-algorithm for symmetric tridiagonal matrices can be adopted to compute the singular values σ_i and the first components of the matrix P of singular vectors of the bidiagonal matrix B; see Vol. II, Sec. 9.7.6.

Review Questions

1. What are orthogonal polynomials? Give a few examples of families of orthogonal polynomials together with the three-term recursion formula, which its members satisfy.
2. Formulate and prove a theorem concerning the location of zeros of orthogonal polynomials.
3. Give an account of Gauss quadrature formulas: accuracy, how the nodes and weights are determined. What important properties are satisfied by the weights?
4. What is the orthogonality property of the Legendre polynomials?

Problems and Computer Exercises

1. Prove that the three-point quadrature formula

$$\int_{-1}^{1} f(x)\,dx \approx \frac{1}{9}\big(5f(-\sqrt{3/5}) + 8f(0) + 5f(\sqrt{3/5})\big),$$

is exact for polynomials of degree 5. Apply it to the computation of

$$\int_0^1 \frac{\sin x}{1+x}\,dx,$$

and estimate the error in the result.

2. (a) Calculate the Hermite polynomials H_n for $n \leq 4$ using the recurrence relation.

(b) Express, conversely, $1, x, x^2, x^3, x^4$ in terms of the Hermite polynomials.

3. Determine the orthogonal polynomials $\phi_n(x)$, $n = 1, 2, 3$, with leading coefficient 1, for the weight function $w(x) = 1 + x^2$, $x \in [-1, 1]$.

(b) Give a two-point Gaussian quadrature formula for integrals of the form

$$\int_{-1}^1 f(x)(1+x^2)\,dx,$$

which is exact when $f(x)$ is a polynomial of degree three.

Hint: Either use the method of undetermined coefficients taking advantage of symmetry, or the three term recurrence relation in Theorem **??**.

4. (W. Gautschi) (a) Construct the quadratic polynomial ϕ_2 orthogonal on $[0, \infty]$ with respect to the weight function $w(x) = e^{-x}$. *Hint*: Use $\int_0^\infty t^m e^{-t}\,dt = m!$.

(b) Obtain the two-point Gauss–Laguerre quadrature formula

$$\int_0^\infty f(x)e^{-x}\,dx = w_1 f(x_1) + w_2 f(x_2) + E_2(f),$$

including a representation for the remainder $E_2(f)$.

(c) Apply the formula in (b) to approximate

$$I = \int_0^\infty (x+1)^{-1} e^{-x}\,dx.$$

Use the remainder term to estimate the error, and compare your estimate with the true error ($I = 0.596347361\ldots$).

5. Show that the formula

$$\int_{-1}^1 f(x)(1-x^2)^{-1/2}\,dx = \frac{\pi}{n}\sum_{k=1}^n f\Big(\cos\frac{2k-1}{2n}\pi\Big)$$

is exact for all polynomials of degree $2n-1$.

6. Derive the Gauss–Lobatto quadrature rule in Example 5.2.3, with two interior points by using the Ansatz

$$\int_{-1}^1 f(x)\,dx = w_1(f(-1)+f(1)) + w_2(f(-x_1)+f(x_1)),$$

and requiring that it be exact for $f(x) = 1, x^2, x^4$.

7. Compute an approximate value of

$$\int_{-1}^{1} x^4 \sin^2 \pi x \, dx = 2 \int_{0}^{1} x^4 \sin^2 \pi x \, dx,$$

using the 5 point Gauss–Legendre quadrature rule on $[0, 1]$ for the weight function $w(x) = 1$. For nodes and weights see Table 5.2.1. (The true value of the integral is 0.11407 77897 39689.)

8. Let $\mu_j = \int_a^b x^j w(x)\, dx$ be the jth **moment** of the weight distribution w. Show that the system of equations

$$\begin{pmatrix} \mu_0 & \mu_1 & \cdots & \mu_{n-1} \\ \mu_1 & \mu_2 & \cdots & \mu_n \\ \vdots & \vdots & \cdots & \vdots \\ \mu_{n-1} & \mu_n & \cdots & \mu_{2n-2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = - \begin{pmatrix} \mu_n \\ \mu_{n+1} \\ \vdots \\ \mu_{2n-1} \end{pmatrix}$$

has as solution the coefficient of a polynomial $x^n + \sum_{j=1}^{n} a_j x^{j-1}$, which is a member of the family of orthogonal polynomials associated with the weight function w.

9. (a) Determine exactly the Lobatto formulas with given nodes at -1 and 1, (and the remaining nodes free), for the weight functions $w(x) = (1 - x^2)^{-\frac{1}{2}}$, $x \in [-1, 1]$. Determine for this weight function also the nodes and weights for the Gauss quadrature formula (i.e. when all nodes are free).

Hint: Set $x = \cos\phi$, and formulate equivalent problems on the unit circle. Note that you obtain (at least) two different discrete orthogonality properties of the Chebyshev polynomials this way.

(b) Lobatto–Kronrod pairs are useful when a long interval has been divided into several shorter intervals (cf. Example 5.2.28). Determine Lobatto–Kronrod pairs (exactly) for $w(x) = (1 - x^2)^{-\frac{1}{2}}$.

10. Apply the formulas in Problem 9 to the case $w(x) = 1$, $x \in [-1, 1]$ and some of the following functions:

(a) $f(x) = e^{kx}$, $k = 1, 2, 4, 8, \ldots$; (b) $f(x) = 1/(k + x)$, $k = 1, 2, 1.1, 1.01$;

(c) $f(x) = k/(1 + k^2x^2)$, $k = 1, 4, 16, 64$.

Compare the actual errors with the error estimates.

11. Write a MATLAB function for the evaluation of the Sievert[6] integral,

$$S(x, \theta) = \int_0^{\theta} e^{-x/\cos\phi}\, d\phi,$$

for any $x \geq 0$, $x \leq \theta \leq 90°$, with at least six decimals relative accuracy. There may be useful hints in Abramowitz–Stegun [1, § 27.4].

[6] Sievert was a Swedish radio-physicist, who was so great that doses of radiation are measured in millisievert, or even microsievert, all over the world.

5.3 Extrapolation Methods

5.3.1 Euler–Maclaurin Formula

Although Newton–Cotes' rules of high orders of accuracy are known, they have the drawback that they do not provide a convenient way of estimating the error. Also, for high order rules negative weights appear. In this section we will derive formulas of high order, based on Euler–Maclaurin's formula (see Sec. 3.5), which do not share these drawbacks.

According to Theorem 3.5.2, if $f \in C^{2r+2}[a,b]$, then

$$T(a:h:b)f - \int_a^b f(x)\,dx = \frac{h^2}{12}\big(f'(b) - f'(a)\big) - \frac{h^4}{720}\big(f'''(b) - f'''(a)\big) + \ldots + \frac{B_{2r}h^{2r}}{(2r)!}\big(f^{(2r-1)}(b) - f^{(2r-1)}(a)\big) + R_{2r+2}(a,h,b)f.$$

Here $x_i = a + ih$, $x_n = b$, and $T(a:h:b)f$ denotes the trapezoidal sum

$$T(a:h:b)f = \sum_{i=1}^{n} \frac{h}{2}\big(f(x_{i-1}) + f(x_i)\big).$$

The remainder $R_{2r+2}(a,h,b)f$ is $O(h^{2r+2})$ is represented by an integral with a kernel of constant sign in (3.5.8). The estimation of the remainder is very simple in certain important particular cases. Note that although the expansion contains derivatives at the boundary points only, the remainder requires tha $|f^{(2r+2)}|$ is integrable on the interval $[a,b]$.

One easily shows the following simple and useful relation of the trapezoidal sum to the midpoint sum

$$R(a,h,b)f = \sum_{i=1}^{n} hf(x_{i-1/2}) = 2T(a:\tfrac{1}{2}h:b)f - T(a:h:b)f. \tag{5.3.1}$$

From this one easily derives the expansion

$$R(a,h,b)f = \int_a^b f(x)\,dx - \frac{h^2}{24}\big(f'(b) - f'(a)\big) + \frac{7h^4}{5760}\big(f'''(b) - f'''(a)\big) + \ldots + \Big(\frac{1}{2^{2r-1}} - 1\Big)\frac{B_{2r}h^{2r}}{(2r)!}\big(f^{(2r-1)}(b) - f^{(2r-1)}(a)\big) + \ldots,$$

which has the same relation to the midpoint sum as the Euler–Maclaurin Formula has to the trapezoidal sum.

The Euler–Maclaurin formulas can be used for highly accurate numerical integration when the values of derivatives of f are known at $x = a$ and $x = b$. It also possible to use difference approximations to estimate the derivatives needed.

A variant with uncentered differences, is **Gregory's**[7] **quadrature formula**

$$\int_a^b f(x)\,dx = h\frac{E^n-1}{hD}f_0 = h\left(\frac{f_n}{-\ln(1-\nabla)} - \frac{f_0}{\ln(1+\Delta)}\right)$$
$$= T(a;h;b) + h\sum_{j=1}^{\infty} a_{j+1}(\nabla^j f_n + (-\Delta)^j f_0),$$

where $T(a:h:b)$ is the trapezoidal sum, as defined in the Euler–Maclaurin Formula. The operator expansion must be truncated at $\nabla^k f_n$ and $\Delta^l f_0$, where $k \le n$, $l \le n$. Concerning the interpretation of ∇^{-1} and Δ^{-1}, see Problem 3.2.13(d).

5.3.2 Romberg's Method

The Euler–Maclaurin formula is the theoretical basis for the application of repeated Richardson extrapolation (see Sec. 3.5.2) to the results of the trapezoidal rule. This method, introduced in [29], is known as **Romberg's method**. It is one of the most widely used methods, because it allows a simple strategy for the automatic determination of a suitable step size and order. A thorough analysis of Romberg's method was carried out by Bauer, Rutishauser and Stiefel [2, 1963] that we shall refer to for proof details.

Let $f \in C^{2m+2}[a,b]$ be a real function to be integrated over $[a,b]$. Set $x_i = a + ih$, $x_n = b$, and denote by

$$T(h)f = \sum_{i=1}^{n} \frac{h}{2}\big(f(x_{i-1}) + f(x_i)\big).$$

the trapezoidal. Then by the Euler–Maclaurin's formula it follows that

$$T(h) - \int_a^b f(x)\,dx = c_2h^2 + c_4h^4 + \cdots + c_mh^{2m} + \tau_{m+1}(h)h^{2m+2},$$

where $c_k = 0$ if $f \in \mathcal{P}_k$. This suggests the use of Repeated Richardson extrapolation applied to the trapezoidal sums computed with step lengths

$$h_0 = \frac{b-a}{n_0}, \quad h_1 = \frac{h_0}{n_0}, \quad \ldots, \quad h_m = \frac{h_{m-1}}{n_m}, \tag{5.3.2}$$

where $n_1, n_2, \ldots, n_m$ are strictly increasing positive integers. Romberg used the special sequence

$$h_i = (b-a)/2^i.$$

In this case Richardson extrapolation can be used with headings $\Delta/3$, $\Delta/15$, $\Delta/63, \ldots$.

[7] James Gregory (1638–1675), Scotch mathematician. This formula was discovered long before the Euler–Maclaurin formula, and seems to have been primarily used for numerical quadrature. It can be used also for summation, but the variants with central differences are typically more efficient.

By (5.3.1) we have the relation

$$T(h/2) = \frac{1}{2}(T(h) + R(h)), \quad R(h)f = \sum_{i=1}^{n} hf(x_{i-1/2}) \tag{5.3.3}$$

where $R(h)$ is he midpoint sum. This makes it possible to reuse the function values that have been computed earlier.

For practical numerical calculations the values of the coefficients c_k are not needed, but they are used, e.g., in the derivation of an error bound, see Theorem 5.3.1. It is also important to remember that the coefficients depend on derivatives of increasing order; the success of repeated Richardson extrapolations is thus related to the behavior in $[a, b]$ of the higher derivatives of the integrand.

According to the discussion of repeated Richardson extrapolation in Sec. 3.5.2, one continues the process, until two values *in the same row* agree to the desired accuracy. If no other error estimate is available, $\min_k |T_{m,k} - T_{m,k-1}|$ is usually chosen as an estimate of the truncation error, even though it is usually a strong overestimate. A feature of the Romberg algorithm is that it also contains exits with lower accuracy at a lower cost.

If the use of the basic asymptotic expansion is doubtful, then the uppermost diagonal of the extrapolation scheme should be ignored, except for its element in the first column. Such a case is detected by inspection of the difference quotients in a column. If for some k, where $T_{k+2,k}$ has been computed and the modulus of the relative irregular error of $T_{k+2,k} - T_{k+1,k}$ is less than (say) 20%, *and*, most important, the difference quotient $(T_{k+1,k} - T_{k,k})/(T_{k+2,k} - T_{k+1,k})$ is is very different from its theoretical value q^{p_k}, then the uppermost diagonal is to be ignored (except for its first element).

Example 5.3.1. *A numerical illustration to Romberg's method.*

Use Romberg's method to compute the integral (cf. Example 5.1.1)

$$\int_0^{0.8} \frac{\sin x}{x}\, dx.$$

The correct value, to ten decimals, is 0.7720957855.

The midpoint and trapezoidal sums computed using IEEE double precision are given below

h	$R(h)f$	$T(h)f$
0.8	0.77883 66846 1730	0.75867 80454 4976
0.4	0.77376 69771 8681	0.76875 73650 3353
0.2	0.77251 27161 1197	0.77126 21711 1017
0.1		0.77188 74436 5335

It can be verified that in this example the error is approximately proportional to h^2 for both $R(h)$ and $T(h)$. We estimate the error in $T(0.1)$ to be $\frac{1}{3}6.26 \cdot 10^{-4} \leq 2.1 \cdot 10^{-4}$.

The trapezoidal sums are then copied to the first column of the Romberg scheme, and repeated extrapolation is applied using the following MATLAB program with $tol = 0.5 \cdot 10^{-10}$ and $q = 5$.

```
function [I, T, md] = romberg(f,a,b,tol,q)
%
% Romberg's method for computing the integral of f over [a,b].
% Stop when two adjacent values in the same column differ by
% less than tol.
%
  T = zeros(q+2,q+1);
  h = b - a;  m = 1; P = 1;
  T(1,1) = h*(feval(f,a) + feval(f,b))/2;
  for i = 2:q+1
    h = h/2; m = 2*m;
    % Compute midpoint sum
    s = 0;
    for j = 1:2:m
      s = s + feval(f, a+j*h)
    end
    R(i-1,1) = 2*h*s;
    T(i,1) = (T(i-1,1) + R(i-1,1))/2;
    % Richardson extrapolation
    jmax = min(i-1,q);
    for j = 1:jmax
      T(i,j+1) = T(i,j) + (T(i,j) - T(i-1,j))/(2^(2*j) - 1)
    end
    % Check accuracy
    [md, jb] = min(abs(T(i,1:jmax) - T(i-1,1:jmax)));
    I = T(i,jb);
    if md <= tol
      T = T(1:i,1:jmax+1); % return active part of T
      return
    end
  end
\vspace{-4mm}
```

The result is given in the table below:

m	T_{m1}	$\Delta/3$	T_{m2}	$\Delta/15$	T_{m3}	T_{44}
1	0.7586780454					
		33597732				
2	0.7687573650		0.7721171382			
		8349354		13355		
3	0.7712621711		0.7720971065		0.7720957710	
		2084242		826		
4	0.7718874437		0.7720958678		0.7720957853	0.7720957855

T_{i1}	T_{i2}	T_{i3}	T_{i4}	T_{i5}
0.758678045450				
0.768757365034	0.772117138228			
0.771262171110	0.772097106469	0.772095771018		
0.771887443653	0.772095867834	0.772095785259	0.772095785485	
0.772043703883	0.772095790626	0.772095785479	0.772095785482	0.772095785482

Since none of the differences $|T_{44} - T_{43}| = 2 \cdot 10^{-10}$, the termination criterion mentioned above requires that the row with $m = 5$ must be computed. Then, the termination criterion is satisfied with a wide margin, since $|T_{55} - T_{54}| = 2.8 \cdot 10^{-12}$, and the irregular errors are less than 10^{-12}. T_{55} is even better than this error bound indicates; the correct result agrees with $T_{55} = 0.772095785482$ to all twelve displayed decimal places.

In cases where the cost of evaluating $F(h)$ is proportional to $1/h$, the standard sequence

$$h_i = (b-a)/n_i, \quad \text{with} \quad n_i = \{1, 2, 4, 8, 16. \ldots\}$$

has the drawback that step sizes decrease rapidly. Bulirsch [4] has proposed the alternative sequence

$$n_i = \{1, 2, 3, 4, 6, 8, 12, 16, 24, \ldots\},$$

for which similar savings can be realized.

In the general case, with $T_{i0} = T(h_i)$ and step lengths given by (5.3.2), repeated Richardson extrapolation using the Neville interpolation scheme takes the form

$$T_{ik} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{(h_{i-k}/h_i)^2 - 1}, \quad 1 \le k \le i \le m.$$

Sometimes rational extrapolation is preferred. This gives rise to a recursion of similar form (see Stoer and Bulirsch [32, Sec. 3.4])

$$T_{ik} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{(h_{i-k}/h_i)^2 \Big[1 - \dfrac{T_{i,k-1} - T_{i-1,k-1}}{T_{i,k-1} - T_{i-1,k-2}}\Big] - 1}, \quad 1 \le k \le i \le m.$$

Theorem 5.3.1. *Error bound for Romberg's method.*

The items T_{mk} in Romberg's method are estimates of the integral $\int_a^{a+h} f(x)\,dx$, that can be expressed as a linear functional,

$$T_{mk} = (b-a) \sum_{j=1}^{n} \alpha_{m,j}^{(k)} f(a + jh), \tag{5.3.4}$$

where $n = 2^{m-1}$, $h = (b-a)/n$, and

$$\sum_{j=1}^{n} \alpha_{m,j}^{(k)} = 1, \qquad \alpha_{m,j}^{(k)} > 0. \tag{5.3.5}$$

The remainder functional for T_{mk} is zero for $f \in \mathcal{P}_{2k}$, and its Peano kernel is positive in the interval (a, b). The truncation error of T_{mk} reads

$$T_{mk} - \int_a^b f(x)dx = r_k h^{2k}(b-a)f^{(2k)}(\tfrac{1}{2}(a+b)) + O(h^{2k+2}(b-a)f^{(2k+2)})$$
$$= r_k h^{2k}(b-a)f^{(2k)}(\xi), \quad \xi \in (a,b), \tag{5.3.6}$$
$$r_k = 2^{k(k-1)}|B_{2k}|/(2k)!, \quad h = 2^{1-m}(b-a).$$

Proof. Sketch: Equation (5.3.4) follows directly from the construction of the Romberg scheme. (It is for theoretical use only; the recursion formulas are better for practical use.) The first formula in (5.3.5) holds, because T_{mk} is exact if $f = 1$. The second formula is easily proved for low values of k. The general proof is more complicated; see [2, Theorem 4].

The Peano kernel for $m = k = 1$ (trapezoidal rule) was constructed in Sec. 3.2. For $m = k = 2$ (Simpson's rule), see Sec. 5.1.2. The general case is more complicated. Recall that, by Corollary 3.3.9 of Peano's Remainder Theorem, a remainder formula with a mean value $\xi \in (a, a+H)$, exists iff the Peano kernel does not change sign.

Bauer, Rutishauser and Stiefel [2, pp. 207–210], constructed a recursion formula for the kernels, and succeeded in proving that they are all positive, by an ingenious use of the recursion. The expression for r_k is also derived there, although with a different notation; see also Problem 3. □

From (5.3.5) it follows that if the magnitude of the irregular error in $f(a+jh)$ is at most ϵ, then the magnitude of the inherited irregular error in T_{mk} is at most $\epsilon(b-a)$.

There is another way of finding r_k. Note that for each value of k, the error of T_{kk} for $f(x) = x^{2k}$ can be determined numerically. Then r_k can be obtained from (5.3.6). T_{mk} is the same formula as T_{kk}, although with a different h.

Sometimes several of the uppermost diagonals are to be ignored. It was mentioned that for the integration of a class of periodic functions the trapezoidal rule is superconvergent. In this case all the difference quotients in the first column are much larger than $q^{p_1} = q^2$. According to the rule just formulated, every element of the Romberg scheme, outside the first column should be ignored. It is all right; *in superconvergent cases Romberg's method is of no use*; it deteriorates the excellent results that the trapezoidal rule has produced. The value $T_{m,k}$ is usually accepted as an estimate of a_0 when $|T_{m,k} - T_{m-1,k}| < \delta$, where δ is the permissible error. Thus one extrapolates until two values *in the same column* agree to the desired accuracy. In most situations, the magnitude of the difference between two values in the same column gives, if h is sufficiently small, with a large margin a bound for the truncation error in the lower of the two values. One cannot, however, get a guaranteed error bound in all situations. Often instead the subdiagonal error criterion $|T_{m,m-1} - T_{m,m}| < \delta$ is used, and T_{mm} taken as the numerical result.

The remainder for the closed Newton–Cotes formulas (with an odd number of

points, i.e., for $k > 0$ in our case), reads

$$d_k h^{2^k+2}(b-a)f^{(2^k+2)}(\xi);$$

for $k = 0$ we have the trapezoidal rule with remainder $d_0 h^2 H f^{(2)}(\xi)$. It follows that for $k = \{0, 1, 2\}$ both methods give, with $k' = \{2, 3, 5\}$, function values, exact results for $f \in \mathcal{P}_{k'}$.[8]

By working algebraically in the Romberg scheme, we obtain the following relations between Romberg's and Newton–Cotes' methods:

$$\begin{aligned}
T_{11} &= \tfrac{1}{2}(b-a)\left(f(a)+f(b)\right),\\
T_{21} &= \frac{1}{4}(b-a)\left(f(a)+f(\tfrac{1}{2}(a+b))+f(\tfrac{1}{2}(a+b))+f(b)\right)\\
&= \tfrac{1}{2}(b-a)\left(\tfrac{1}{2}f(a)+f(\tfrac{1}{2}(a+b))+\tfrac{1}{2}f(b)\right),\\
T_{22} &= \frac{1}{3}(4T_{21}-T_{11}) = \frac{1}{6}(b-a)\left(f(a)+4f(\tfrac{1}{2}(a+b))+f(b)\right). \qquad (5.3.7)
\end{aligned}$$

We see that T_{22} is the same as Simpson's formula. It can also be shown in this way that T_{33} is the same as the five point closed Newton–Cotes formula.

Table 5.3.1. *Data concerning some Romberg and Newton–Cotes formulas.*

$m = k$	n	order T_{kk}	order C_n	error const. r_k	error const. c_n
1	1	2	2	1/12	1/12
2	2	4	4	1/180	1/180
3	4	6	6	2/945	2/945
4	8	8	10	16/4725	296/467775

This equivalence can also be proved by the following argument. By Corollary 3.3.8, there is only one linear combination of the values of the function f at $n+1$ given points that can yield $\int_a^b f(x)\,dx$ exactly for all polynomials $f \in \mathcal{P}_{n+1}$. It follows that the methods of Cotes and Romberg T_{kk} are identical for $k = 0, 1, 2$, but for $k > 2$, $2^k + 2 > 2k + 2$, and the methods are not identical. For $k = 3$ (9 function values), Cotes is exact in $\mathcal{P}_{10}$, while T_{33} is exact in $\mathcal{P}_8$. For $k = 4$ (17 function values), Cotes is exact in $\mathcal{P}_{18}$, while T_{44} is exact in $\mathcal{P}_{10}$; see Table 5.3.1. This sounds like an advantage for Cotes, but one has to be sceptical about formulas that use equidistant points in polynomial approximation of very high degree; see Problem 5 and the discussion of Runge's phenomena in Chapter 4.

Note that the remainder of T_{44} is

$$r_4 h^8 (b-a) f^{(8)}(\xi) \approx r_4 (b-a)\Delta^8 f(a),$$

where $\Delta^8 f(a)$ uses the same function values as T_{44} and C_8. So we can use $r_4(b-a)\Delta^8 f(a)$ as an asymptotically correct error estimate for T_{44}.

[8] For $k = 2, 3$, the results are exact even in $\mathcal{P}_{k'+1}$, due to the symmetry discussed in Example 3.2.7.

When the values in a row of a Richardson scheme converge fast, it is worth to *try, e.g., Aitken extrapolation to this row*, in order to improve the error estimate of the diagonal element $T_{m,m}$. It is important that the irregular errors of the values are small compared to the last Richardson correction. The theoretical support to this is usually rather poor, and the row should therefore contain at least four items, so that one can obtain two Aitken accelerated values. These should not be accepted as results, but they provide two error estimates for $T_{m,m}$. The largest absolute value of these error estimates indicates the order of magnitude of the error of $T_{m,m}$, but it is not a guaranteed error bound. If you want to calculate the indefinite integral of $f(x)$, it may be irritating that the improvements are made only at the endpoints; H may be too big for the application of interpolation of the results afterwards or for graphical output. An idea how to get denser output is developed in [23].

If the function to be integrated has a singularity in the interval. then the expansion no longer is a series in h^2 and Romberg's metod has to be modified.

For example, if the integrand $f(x)$ has an algebraic end-point singularity,

$$f(x) = x^{\beta} h(x), \quad -1 < \beta \leq 0,$$

where $h(x) \in C^{p+1}[a, b]$, then an asymptotic expansion of the form

$$R = \sum_{q=1}^{p} a_q k^{-\beta-q} + \sum_{q=1}^{p} b_q k^{-q} + O(k^{-p-1}) \tag{5.3.8}$$

can be shown to hold for a trapezoidal sum. Similar, but more complicated, expansions can be obtained for other classes of singularities.

The case when the error expansion for the trapezoidal sum has the form

$$T(h) = I + \sum_{m=1}^{n} a_m e_m(h) + R_n(h), \tag{5.3.9}$$

where $e_j(h)$, $j = 1, 2, \ldots$ are known functions satisfying $\lim_{h\to 0} e_{m+1}(h)/e_m(h) = 0$ and the error term satisfies $R_n(h) = O(e_{n+1}(h))$ has been treated by Håvie [19].

Suppose we have computed the trapezoidal sums $T_0^{(k)} = T(h_k)$, for a sequence of steplengths $h_0 > h_1 > h_2 \cdots > h_n > 0$. We want to compute the "best" possible approximation $T_n^{(0)}$ to $I = \lim_{h\to 0} T(h)$, defined by the equations

$$T(h) = T_n^{(0)} + \sum_{m=1}^{n} a'_m e_m(h_k), \quad k = 0 : n. \tag{5.3.10}$$

Håvie showed how the approximations can be computed by a special recurrence relation. Set

$$E_1^{(n)} = \frac{e_1(h_{n+1})T_n - e_1(h_n)T_{n+1}}{e_1(h_{n+1}) - e_1(h_n)}.$$

Replacing T_n and T_{n+1} by their expansion, we obtain

$$e_{1,i}^{(n)} = \frac{e_1(h_{n+1})e_i(h_n) - e_1(h_n)e_i(h_{n+1})}{e_1(h_{n+1}) - e_1(h_n)}.$$

The same process can be repeated for eliminating $e_{1,2}^{(n)}$ in the expansion of $E_1^{(n)}$, and so on. This gives the **E-algorithm**

$$E_k^{(n)} = \frac{e_{k-1.k}^{(n+1)} E_{k-1}^{(n)} - e_{k-1.k}^{(n)} E_{k-1}^{(n+1)}}{e_{k-1.k}^{(n+1)} - e_{k-1.k}^{(n)}}. \tag{5.3.11}$$

The auxiliary quantities $e_{k.i}^{(n)}$ are recursively computed by a similar rule

$$e_{k,i}^{(n)} = \frac{e_{k-1.k}^{(n+1)} e_{k-1,i}^{(n)} - e_{k-1.k}^{(n)} e_{k-1,i}^{(n+1)}}{e_{k-1.k}^{(n+1)} - e_{k-1.k}^{(n)}}, \tag{5.3.12}$$

with $e_{0.i}^{(n)} = e_i(h_n)$. The algorithm can be interpreted in terms of Gaussian elimination for solving the system

$$E_k^{(n)} + b_1 g(h_{n+i}) + \cdots + b_k g(h_{n+i}) = T_{n+i}, \quad i = 0 : k,$$

for the unknown $E_k^{(n)}$.

5.3.3 The Epsilon Algorithm

Richardson extrapolation as used in Romberg's method can only be used to accelerate the rate of convergence if the exponents in the asymptotic expansions are known *explicitly*. In cases when the exponents are unknown a nonlinear extrapolation scheme, like the ϵ-algorithm (see Sec. 3.3.5) has to be used. This is the most important convergence acceleration scheme besides Richardson extrapolation in numerical quadrature.

In the ϵ algorithm a two-dimensional array of numbers $\epsilon_k^{(p)}$ is computed by the recurrence relation,

$$\epsilon_{k+1}^{(p)} = \epsilon_{k-1}^{(p+1)} + \frac{1}{\epsilon_k^{(p+1)} - \epsilon_k^{(p)}}. \tag{5.3.13}$$

using the following boundary conditions

$$\epsilon_{-1}^{(p)} = 0, \quad p = 1, 2, 3, \ldots,$$
$$\epsilon_0^{(p)} = s_p, \quad p = 0, 1, 2, \ldots$$

Example 5.3.2.

Consider the integral

$$\int \sqrt{x}\, dx = 2/3.$$

If Romberg's method is applied to this integral the convergence is very slow. In contrast the ϵ-algorithm is well adapted to accelerating convergence when an asymptotic error expansion of the form (5.3.8) holds.

In the figure below the results from Romberg's method applied to the trapezoidal rule for. This is compared with the results from applying the ϵ-algorithm to the same trapezoidal sums used in Romberg's method. Already for $\epsilon_5^{(0)}$ the order of magnitude of the error is the same as the accumulated rounding error using IEEE double precision.

5.3.4 Infinite Intervals

In general the trapezoidal rule is second order accurate, unless $f'(a) = f'(b)$, but there exist *interesting exceptions*. Suppose that the function f is infinitely differentiable for $x \in \mathbf{R}$, and that f *has* $[a, b]$ *as an interval of periodicity*, i.e.,

$$f(x + (b - a)) = f(x), \quad \forall \quad x \in \mathbf{R}.$$

Then $f^{(k)}(b) = f^{(k)}(a)$, for $k = 0, 1, 2, \ldots$, hence *every term in the Euler–Maclaurin expansion is zero* for the integral over *the whole period* $[a, b]$. One could be led to believe that the trapezoidal rule gives the exact value of the integral, but this is usually not the case; for most periodic functions f, $\lim_{r\to\infty} R_{2r+2} f \neq 0$; *the expansion converges, of course, though not necessarily to the correct result.*

On the other hand, the convergence as $h \to 0$ for a fixed (though arbitrary) r is a different story; the error bound (5.3.6) shows that

$$|R_{2r+2}(a, h, b)f| = O(h^{2r+2}).$$

Since r is arbitrary, this means that *for this class of functions, the trapezoidal error tends to zero faster than any power of* h, *as* $h \to 0$. We may call this **superconvergence**. The application of the trapezoidal rule to an integral over $[0, \infty)$ of a function $f \in C^\infty(0, \infty)$ often yields similar features, sometimes even more striking.

Suppose that the periodic function $f(z)$, $z = x + iy$, is analytic in a strip, $|y| < c$, around the real axis. It can then be shown that the error of the trapezoidal rule is $O(e^{-\eta/h})$ as $h \downarrow 0$; η is related to the width of the strip. A similar result will be obtained in Example 5.3.3, for an annulus instead of a strip.

As a rule, this discussion does *not* apply to periodic functions which are defined by periodic continuation of a function originally defined on $[a, b]$ (like the Bernoulli functions). They usually become non-analytic at a and b, and at all points $a + (b - a)n$, $n = 0, \pm 1, \pm 2, \ldots$.

The **Poisson summation formula** is, even better than the Euler–Maclaurin formula for the quantitative study of the trapezoidal truncation error on an infinite interval. For convenient reference we now formulate the following surprising result:

Theorem 5.3.2. *Suppose that the trapezoidal rule (or, equivalently, the rectangle rule) is applied with* constant *step size* h *to* $\int_{-\infty}^{\infty} f(x)\,dx$. *The Fourier transform of* f *reads*

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x)e^{-i\omega t}\,dt.$$

Then the integration error decreases like $2\hat{f}(2\pi/h)$ *as* $h \downarrow 0$.

Example 5.3.3.

For the normal probability density, we have

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(t/\sigma)^2}, \qquad \hat{f}(\omega) = e^{-\frac{1}{2}(\omega\sigma)^2}.$$

The integration error is thus approximately $2\exp(-2(\pi\sigma/h)^2)$. Roughly speaking, the number of correct digits is doubled if h is divided by $\sqrt{2}$, e.g., the error is approximately $5.4\,10^{-9}$ for $h = \sigma$, and $1.4\,10^{-17}$ for $h = \sigma/\sqrt{2}$.

The application of the trapezoidal rule to an integral over $[0, \infty)$ of a function $f \in C^\infty(0, \infty)$ often yields similar features, sometimes even more striking. Suppose that, for $k = 1, 2, 3, \ldots$,

$$f^{(2k-1)}(0) = 0 \quad \text{and} \quad f^{(2k-1)}(x) \to 0, \quad x \to \infty,$$

and $\int_0^\infty |f^{(2k)}(x)|\,dx < \infty$. (Note that for any function $g \in C^\infty(-\infty, \infty)$ the function $f(x) = g(x) + g(-x)$ satisfies such conditions at the origin.) Then all terms of the Euler–Maclaurin expansion are zero, and one can be misled to believe that the trapezoidal sum gives $\int_0^\infty f(x)\,dx$ exactly for any step size h! We have already seen an example of this in Example 3.5.3. See also Theorem 5.3.2 and Problem 3. The explanation is that the remainder $R_{2r+2}(a, h, \infty)$ will typically not tend to zero, as $r \to \infty$ for fixed h. On the other hand: if we consider the behavior of the truncation error as $h \to 0$ for given r, we find that it is $o(h^{2r})$ for any r, just like the case of a periodic unction.

For a finite subinterval of $[0, \infty)$, however, the remainder is still typically $O(h^2)$, and for each step the remainder is typically $O(h^3)$. So, there is an *enormous cancellation of the local truncation errors*, when a C^∞-function, with vanishing odd-order derivatives at the origin, is integrated by the trapezoidal rule over $[0, \infty)$.

Example 5.3.4.

Infinite intervals of integration occur often in practical problems. For integrals of the form $\int_{-\infty}^\infty f(x)\,dx$, the trapezoidal rule (or the midpoint rule) often gives good accuracy if one integrates over the interval $[-R_1, R_2]$, assuming that $f(x)$ and its lower derivatives are small for $x \le -R_1$ and $x \ge R_2$.

The correct value to six decimal digits of the integral $\int_{-\infty}^\infty e^{-x^2}\,dx$ is $\pi^{1/2} = 1.772454$. For $x \pm 4$, the integrand is less than $0.5\,10^{-6}$. Using the trapezoidal rule for the integral over $[-4, 4]$ we get the estimate 1.772453 with $h = 1/2$, an amazingly good result. (The values of the function have been taken from a six-place table.) The truncation error in the value of the integral is here less than 1/10,000 of the truncation error in the largest term of the trapezoidal sum—a superb example of "cancellation of truncation error". The error that is which is committed when we

replace ∞ by 4 can be estimated in the following way:

$$|R| = 2\int_4^\infty e^{-x^2}\,dx = 2\int_{16}^\infty e^{-t}0.5t^{-1/2}\,dt$$
$$= 2\cdot 0.516^{-1/2}\int_{16}^\infty e^{-t}0.5\,dt = \frac{1}{4}e^{-16} < 10^{-7}.$$

5.3.5 Adaptive Quadrature

It is often the case that the integrand $f(x)$ (or its derivatives) has strongly varying orders of magnitude in different parts of the interval of integration $[a, b]$. One should then choose *different step sizes in different parts of the integration interval*. Since

$$\int_a^b = \int_a^{c_1} + \int_{c_1}^{c_2} + \cdots + \int_{c_1}^b,$$

the integrals on the right hand side can be treated as independent subproblems. Indeed, it is possible to perform the subdivision recursively in several levels. In **adaptive quadrature methods** step sizes are automatically adapts so that the approximation satisfies a prescribed error tolerance

$$\left| I - \int_a^b f(x)\,dx \right| \le \epsilon. \tag{5.3.14}$$

We first remark that evaluation of the integral (**??**) is equivalent to solving

$$\frac{dy}{dx} = f(x), \qquad y(a) = 0, \tag{5.3.15}$$

and taking $I = y(b)$. This is a special case of an initial value problem for an ordinary differential equation, and the methods described in Chapter 13 can be used to solve the problem (5.3.15). These algorithms have been developed to include sophisticated techniques for adaptively choosing step size and order in the integration (see Sec. 13.2), and may therefore be a good choice for handling difficult cases.

We consider first a fixed order adaptive method based on Simpson's rule. For a subinterval $[a, b]$, set $h_=(b-a)$ and compute the trapezoidal approximations

$$T_{00} = T(h), \quad T_{10} = T(h/2), \quad T_{20} = T(h/4).$$

The extrapolated values

$$T_{11} = (4T_{10} - T_{00})/3, \qquad T_{21} = (4T_{20} - T_{10})/3,$$

are equivalent to (the composite) Simpson's rule with step length $h/2$ and $h/4$, respectively. We can also calculate

$$T_{22} = (16T_{21} - T_{11})/15,$$

which is Milne's method with step length $h/4$ with remainder equal to $(2/945)(h/4)^6(b-a)f^{(6)}(\xi)$.

For T_{22} we use the error estimate $R_j = |T_{22} - T_{21}|$, which often is a crude overestimate.

We *accept* the approximation I_j if

$$|T_{21} - T_{11}| < \frac{h_j \epsilon}{b-a}, \tag{5.3.16}$$

that is we require the error to be *less than* $\epsilon/(b-a)$ *per unit step.* Otherwise we *reject* the approximation, and subdivide the interval in two intervals $[a_j, \frac{1}{2}(a_j + b_j)]$, $[\frac{1}{2}(a_j + b_j), b_j]$. The same rule is now applied to these two subintervals.

Note that if the function values computed previously are we have saved, these can be reused for the new intervals. Only We start with one interval $[a, b]$ and carry on subdivisions until the error criterion in (5.3.16) is satisfied for all intervals. Since the total error is the sum of errors for all subintervals we then have the error estimate

$$R_T < \sum_j \frac{h_j \epsilon}{b-a} = \epsilon$$

as required.

Many adaptive quadrature schemes exits. Here we shall only illustrate one simple scheme based on a five point closed Newton–Cotes rule, which applies bisection in a locally adaptive strategy. All function evaluations contribute to the final estimate.

Algorithm 5.3.1 *Adaptive Simpson.*

Let f be a given function to be integrated over $[a, b]$ The algorithm adaptsimp uses a recursive to compute an approximation with an error less than a specified tolerance $\tau > 0$. The parameter is is a crude a priori estimation of I, used in the stopping criterion.

```
function [I,nf] = adaptsimp(f,a,b,tol);
% ADAPTSIMP computes the integral of the
% function vector valued function f over [a,b];
% tol is the desired absolute accuracy
% nf is the number of function evaluations
%
% Initial Simpson approximation
ff = feval(f,[a, (a+b)/2, b]); nf = 3;
I1 = (b - a)*[1, 4, 1]*ff'/6;
% Recursive computation
[I,nf] =  adaptrec(f,a,b,ff,I1,tol,nf);

function [I,nf] = adaptrec(f,a,b,ff,I1,tol,nf);
h = (b - a)/2;
fm = feval(f, [a + h/2, b - h/2]); nf = nf + 2;
```

```
% Simpson approximations from left and right subinterval
fL = [ff(1); fm(1); ff(2)];
fR = [ff(2); fm(2); ff(3)];
IL = h*[1, 4, 1]*fL/6;
IR = h*[1, 4, 1]*fR/6;;
% Compute Extrapolated approximation
I2 = IL + IR;
I  = I2 + (I2 - I1)/15;
if  abs(I - I2) > tol
% Refine both subintervals
   [IL,nf] = adaptrec(f,a,a+h,fL,IL,tol/2,nf);
   [IR,nf] = adaptrec(f,b-h,b,fR,IR,tol/2,nf);
   I  = IL + IR;!
 end
```

In many situations it might be preferable to specify a *relative error tolerance*

$$tol = \eta \Big| \int_a^b f(x)\,dx \Big|.$$

Note that in a **locally adaptive** algorithm using a recursive partitioning scheme, the subintervals are processed from left to right until the integral over each subinterval satisfies some error requirement. This means that an a priori initial estimate of the whole integral, needed for use in a relative local error estimate cannot be updated until all subintervals are processed and the computation is finished. Hence, if a relative tolerance is specified then a estimate of the integral is needed before the recursion starts. This is complicated by the fact that the initial estimate might be zero, e.g. if a periodic integrand is sampled at equidistant intervals. Hence a combination of relative and absolute criterion might be preferable.

Example 5.3.5.

This algorithm was used to compute the integral

$$\int_{-4}^{4} \frac{dx}{1+x^2} = 2.65163532733607.$$

with an absolute tolerance 10^{-p}, $p = 4, 5, 6$. The following approximations were obtained.

I	tol	n	error
2.65162 50211	10^{-4}	41	$1.0\,10^{-5}$
2.65163 52064	10^{-5}	81	$1.2\,10^{-7}$
2.65163 5327353	10^{-6}	153	$-1.7\,10^{-11}$

Note that the actual error is much smaller than the required tolerance.

The possibility that a user might try to integrate a non-integrable function (e.g., $f(x) = x^{-1}$ on $[0, 1]$) cannot be neglected. In principle it is not possible to decide whether or not a function $f(x)$ is integrable on the basis of a finite sample $f(x_1), \ldots, f(x_N)$ of function values. Therefore it is necessary to impose

1. an upper limit on the computational effort, i.e. the number of function evaluation.

2. a lower limit on the size of the subregions

This means that premature termination may occur even when the function is close to being non-integrable, e.g., $f(x) = x^{-0.99}$.

So far we have considered adaptive routines, which use fixed quadrature rules on each subinterval but where the partition of the interval depends on the integrand. Such an algorithm is said to be **partition adaptive**. We can also consider **doubly adaptive** integration algorithms. These can choose from a sequence of increasingly higher order rules to be applied to the current subinterval. Such algorithms uses a selection criterion to decide to decide at each stage whether to subdivide the current subinterval or to apply a higher order rule. Doubly adaptive routines copes more efficiently with smooth integrands.

Many variations on the simple scheme outlined above are possible. For example, we could base the method on a higher order Romberg scheme, or even try to choose an optimal order for each subinterval. Adaptive methods work even when the integrand $f(x)$ is badly behaved. However, if f has singularities or unbounded derivatives, the error criterion may never be satisfied. For guard against such cases it is necessary to include some bound of the number of recursion levels that are allowed. It should be kept in mind that although adaptive quadrature algorithms are convenient to use they are in general less efficient than methods which have been specially adapted for a particular problem.

A collection of computer subroutines for adaptive quadrature is given by Piessens et al. [28]. We finally warn the reader that *no automatic quadrature routine can be guaranteed always to work*. Indeed any estimate of $\int_a^b f(x)\,dx$ based solely on the value of $f(x)$ on finitely many points can fail. The integrand $f(x)$ may, for example, be nonzero only on a small subset of $[a, b]$. An adaptive quadrature rule based only on samples $f(x)$ in a finite number of points theoretically may return the value zero in such a case!

Review Questions

1. Give an account of the theoretical background of Romberg's method and its use.

2. Romberg's method uses extrapolation of a sequence of trapezoidal approximations computed for a sequence of step sizes $h_0, h_1, h_2, \ldots$. What sequences have been suggested and what are their relative merits?

Problems and Computer Exercises

1. Is it true that (the short version of) Simpson's formula is a particular case of Gregory's formula? (Simpson lived 1710-1761.)

2. Use Romberg's method to compute the integral $\int_0^4 f(x)\,dx$, using the following (correctly rounded) values of $f(x)$. Need all the values be used?

x	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
$f(x)$	−4271	−2522	−499	1795	4358	7187	10279	13633	17247

3. (a) Suppose that the form of the error of Romberg's method is known, but the error constant r_k is not known. Determine r_k numerically for $K = 3$ and $k = 4$, by computing the Romberg scheme for $f(x) = x^{2k}$.

(b) Prove the formula for the error constant of Romberg's method.

4. Compute by the Euler–Maclaurin formula, or rather the trapezoidal rule,

$$\text{(a)}\ \int_0^\infty e^{-x^2/2}dx, \qquad \text{(b)}\ \int_0^\infty \frac{dx}{\cosh(\pi x)},$$

as accurately as you can with the normal precision of your computer (or software). Then find out empirically how the error depends on h. Make semi-logarithmic plots on the same screen. How long range of integration do you need?

goodbreak

5. (a) Compute $\int_1^\infty (1+x^2)^{-1}\,dx$. In the notation of Example 5.3.5, compute $\int_1^2$, $\int_2^4$, $\int_4^8, \ldots$; choose yourself where to stop. Use, e.g., Aitken acceleration to find $\int_1^\infty$. Compare with the exact result; and think of an error estimate that can be used if the exact result is not known.

(b)*Romberg+Aitken* Treat in the same way $\int_1^\infty \frac{1}{\sqrt{x+x^3}}$. Compare the computational effort for the computation of the tail $\int_R^\infty$ by acceleration and by series expansion with the same accuracy.

6. Compute the integral

$$\frac{1}{2\pi}\int_0^{2\pi} e^{\frac{1}{\sqrt{2}}\sin x}dx$$

by the trapezoidal rule, using $h = \pi/2$ and $h = \pi/4$ (for hand-held calculator). Continue on a computer with smaller values of h, until the error is on the level of the rounding errors. Observe how the number of correct digits vary with h? Notice that Romberg is of no use in this problem.

7. (a) Show that the trapezoidal rule, with $h = 2\pi/(n+1)$, is exact for all trigonometric polynomials of period 2π—i.e., for functions of the type

$$\sum_{k=-n}^{n} c_k e^{ikt}, \qquad i^2 = -1.$$

—when it is used for integration over a whole period.

(b) Show that if $f(x)$ can be approximated by a trigonometric polynomial of degree n so that the magnitude of the error is less than ϵ, in the interval $(0, 2\pi)$, then the error with the use of the trapezoidal rule with $h = 2\pi/(n+1)$

on the integral $(2\pi)^{-1}\int_0^{2\pi} f(x)\,dx$ is less than 2ϵ.

(c) Use the above to explain the sensationally good result in Problem 2 above, when $h = \pi/4$.

Hint: First estimate how well the function $g(x) = e^{x/\sqrt{2}}$ can be approximated by a polynomial in $\mathcal{P}_8$ for $x \in [-1, 1]$. The estimate found by the truncated Maclaurin expansion is not quite good enough. Theorem 3.1.5 provides a sharper estimate with an appropriate choice of R; remember Scylla and Charybdis.

8. (J. N. Lyness) The integral

$$I(f, g) = \int_0^{nh} f(x)g'(x)\,dx \tag{5.3.17}$$

is called a **Stieltjes integral**. An approximation related to the trapezoidal rule is

$$S_m = \tfrac{1}{2}\sum_{j=0}^{n-1} \big(f(jh) + f((j+1)h)\big)\big(g((j+1)h) - (g(jh)\big),$$

which requires $2(m+1)$ function evaluations. Similarly an analogue to the "mid-point rule" is

$$R_m = \tfrac{1}{2}\sum_{j=0}^{n-1}{}'' f(jh)\big(g((j+1)h) - (g((j-1)h)\big),$$

where the double prime on the summation indicates that the extreme values $j = 0$ and $j = m$ are assigned a weighting factor $\frac{1}{2}$. This rule requires $2(m+2)$ function evaluations, two of which lie outside the interval of integration.

(a) Show that the difference $S - m - R_m$ is of order $O(h^2)$.

9. Apply the programs handed out for Romberg's method (also longromb) and repeated averages on the integral

$$\int_0^{1000} x\cos(x^3)\,dx.$$

Try to obtain the results with 10 decimal places.

5.4 Multiple Integrals

5.4.1 Product Rules

The ideas of numerical quadrature can be generalized to multiple integrals. Consider the two-dimensional integral

$$I = \int_D f(x, y)\,dxdy \tag{5.4.1}$$

For regions D, such as a square, cube, cylinder, etc., which are the Cartesian product of lower dimensional regions, integration rules can be developed by multiplying together the lower dimensional rules. For example, if

$$\int_0^1 f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i)$$

is a one dimensional rule, then

$$\int_0^1 \int_0^1 f(x,y)\,dxdy = \sum_{i,j=1}^{n} w_i w_j f(x_i, y_j)$$

is a two-dimensional rule for a square. Such rules are not necessarily the most economical rules.

Example 5.4.1.

Consider a quadrature rule of the form

$$\int_{-h}^{h} \int_{-h}^{h} f(x,y)\,dxdy = 4h^2 \sum_{i,j=1}^{n} w_i f(x_i, y_i).$$

The product Simpson's rule uses 9 function values, with abscissas and weights given by

(x_i, y_i)	(0,0)	$(\pm h, \pm h)$	$(\pm h, 0)$	$(0, \pm h)$
w_i	4/9	1/36	1/9	1/9

A more efficient rule is the product 2-point Legendre rule, using the four points

$$(x_i, y_i) = \Big(\pm \frac{h}{\sqrt{3}}, \pm \frac{h}{\sqrt{3}} \Big) \quad w_i = 1/4.$$

For both rules the error is $O(h^4)$. Some quadrature rules for circles, triangles, hexagons, spheres, cubes, etc., are given in Abramowitz–Stegun [1, § 25].

Since the amount of work will increase rapidly with the number of dimensions. It is therefore advisable to try to reduce the number of dimensions by applying analytic techniques to parts of the task.

Example 5.4.2.

The following triple integral can be reduced to a single integral:

$$\int_0^\infty \int_0^\infty \int_0^\infty e^{-(x+y+z)} \sin(xz) \sin(yz)\,dxdydz$$

$$\int_0^\infty e^{-x}\,dx \int_0^\infty e^{-y} \sin(yx)dy \int_0^\infty e^{-z} \sin(zx) = \int_0^\infty \Big(\frac{x}{1+x^2} \Big)^2 e^{-x}\,dx,$$

because

$$\int_0^\infty e^{-z} \sin(zx)dz = \int_0^\infty e^{-y} \sin(yx)dz = \frac{x}{1+x^2}.$$

The remaining single integral is simply evaluated by the techniques previously studied.

Often a transformation of variable is needed for such a reduction (see Problem 1 at the end of this section), but sometimes that does not help either. Several approaches are then possible:

(a) numerical integration in one direction at a time—see Sec. 5.4.2;

(b) the use of a rectangular grid, mainly if the boundary of the region is composed of straight lines–see Sec. 5.4.3.

(c) the use of an irregular triangular grid—possible for more general boundaries—see Sec. 5.4.4.

(d) Monte Carlo methods, mainly for problems with complicated boundaries and a large number of dimensions—see Sec. 5.4.5.

5.4.2 Successive One-Dimensional Quadrature

For simplicity we restrict ourselves below to the two-dimensional case, although the ideas are more general. Consider the integral (5.4.1) where D is a domain in the x-y plane. The simplest way to compute an approximation to I is by repeated use of one dimensional quadrature rules. If lines parallel with the x-axis have at most one segment in common with D, then I can be written in the form

$$I = \int_a^b \Big(\int_{c(x)}^{d(x)} f(x,y)dy \Big)\, dx,$$

or

$$I = \int_a^b \varphi(x)\, dx, \qquad \varphi(x) = \int_{c(x)}^{d(x)} f(x,y)dy. \tag{5.4.2}$$

For a sequence of values x_i, $i = 1, \ldots, n$ we can evaluate the function $\varphi(x)$ by the one-dimensional quadrature methods described previously. These function values are then used in another one-dimensional quadrature rule to evaluate I. Note that if D is a more general domain, it might be possible to decompose D into the union of simpler domains on which these methods can be used.

Figure 5.4.1. *Region D of integration.*

Example 5.4.3.
Compute

$$I = \int\int_D \sin^2 y \sin^2 x (1 + x^2 + y^2)^{-1/2}\, dxdy,$$

where

$$D = \{(x, y) \mid x^2 + y^2 \le 1\} \cup \{(x, y) \mid 1 \le x \le 3, |y| \le 0.5\}.$$

is the composite region shown in Fig. 8.4.1. Then

$$I = \int_{-1}^{3} \varphi(x) \sin^2 x\, dx, \tag{5.4.3}$$

$$\varphi(x) = \int_{-c(x)}^{c(x)} \sin^2 y (1 + x^2 + y^2)^{-1/2} dy, \tag{5.4.4}$$

where

$$c(x) = \begin{cases} (1 - x^2)^{1/2}, & x \le \frac{1}{2}\sqrt{3}; \\ \frac{1}{2}, & x \ge \frac{1}{2}\sqrt{3}. \end{cases}$$

Values of $\varphi(x)$ were obtained by the application of Romberg's method to (5.4.4) and numerical integration applied to the integral (5.4.3) yielded the value of $I = 0.13202 \pm 10^{-5}$. Ninety-six values of x were needed, and for each value of x, twenty function evaluations used, on the average. The grid is chosen so that $x = \frac{1}{2}\sqrt{3}$, where $\varphi'(x)$ is discontinuous, is a grid point.

5.4.3 Product Rules

Consider a double integral over a rectangular region $D = \{(x, y) \mid a \le x \le b, c \le y \le d\}$. Decomposing the integral as in (5.4.2) and using one-dimensional quadrature rules we can write

$$I \approx \sum_{i=1}^{n} u_i \varphi(x_i), \qquad \varphi(x_i) \approx \sum_{j=1}^{n} v_j f(x_i, y_j),$$

or, combining the rules

$$I \approx \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} f(x_i, y_j), \qquad w_{ij} = u_i v_j. \tag{5.4.5}$$

This is called a **product rule** for the double integral I, and it uses mn function values $f_{ij} = f(x_i, y_j)$.

In particular we can use values of f and an equidistant **rectangular grid** in the (x, y)-plane with grid spacings h and k in the x and y directions, respectively. Let $x_0 = a$, $h = (b-a)/n$, $y_0 = c$, $k = (d-c)/m$, and use the notation $x_i = x_0 + ih$,

$y_j = y_0 + jk$. Then the following formulas can be used, generalizing the compound rectangle rule and trapezoidal rule, respectively:

$$I \approx hk \sum_{i=1}^{M} \sum_{j=1}^{N} f_{i-\frac{1}{2},j-\frac{1}{2}}, \tag{5.4.6}$$

$$I \approx hk \sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij} f_{ij} \tag{5.4.7}$$

Here, for the trapezoidal rule $w_{ij} = 1$ for the interior grid points—i.e., when $0 < i < M$ and $0 < j < N$, $w_{ij} = \frac{1}{4}$ for the four corner points, while $w_{ij} = \frac{1}{2}$ for the other boundary points. Both formulas are exact for bilinear functions, and the error can be expanded in even powers of h, k so that repeated Richardson extrapolation can be used.

Formulas of higher accuracy can also be obtained by using Gaussian quadrature rules in the x and y direction. Note that if the one-dimensional formulas are exact for polynomials of degree d_1 and d_2, respectively, then the product rule will be exact for bivariate polynomials $x^p y^q$ where $p \leq d_1$ and $q \leq d_2$.

Higher accuracy formulas can also be derived by **operator** techniques, based on an operator formulation of Taylor's expansion, see equation (4.8.2),

$$u(x_0 + h, y_0 + k) = e^{(hD_x + kD_y)} u(x_0, y_0). \tag{5.4.8}$$

It is possible to use product rules on non-rectangular regions, if these can be mapped into a rectangle. This can be done, e.g., for a triangle. For nonrectangular regions, the rectangular lattice may also be bordered by triangles or "triangles" with one curved side, which may be treated with the techniques outlined in the next section.

5.4.4 Irregular Triangular Grids

A grid of triangles of arbitrary form is a convenient means for approximating a complicated plane region. It is fairly easy to program a computer to refine a coarse triangular grid automatically; see Fig. 8.4.2. It is also easy to adapt the density of points to the behavior of the function.

Triangular grids are thus more flexible than rectangular ones. On the other hand, the administration of a rectangular grid requires less storage and a simpler program. Sometimes the approximation formulas are also a little simpler. Triangular grids have an important application in the **finite element method** (FEM) for problems in continuum mechanics and other applications of partial differential equations; see Chapter 14.

Let $P_i = (x_i, y_i)$, $i = 1, 2, 3$, be the vertices of a triangle T. Then any point $P = (x, y)$ in the plane can be uniquely expressed by the vector equation

$$P = \theta_1 P_1 + \theta_2 P_2 + \theta_3 P_3, \qquad \theta_1 + \theta_1 + \theta_1 = 1. \tag{5.4.9}$$

Figure 5.4.2. *Refinement of a triangular grid.*

In fact, the θ_i, which are called **barycentric coordinates** of P, are determined from the following nonsingular set of equations:

$$\begin{aligned} \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 &= x, \\ \theta_1 y_1 + \theta_2 y_2 + \theta_3 y_3 &= y, \\ \theta_1 + \theta_2 + \theta_3 &= 1, \end{aligned} \tag{5.4.10}$$

The interior of the triangle is characterized by the inequalities $\theta_i > 0$, $i = 1, 2, 3$. In this case P is the center of mass (centroid) of the three masses $\theta_1, \theta_2, \theta_3$ located at the vertices of the triangle (see Fig. 8.4.3). This explains the term "barycentric coordinates". $\theta_1 = 0$ is the equation for the side P_2P_3, and similarly for the other sides.

Figure 5.4.3. *Center of mass of a triangle.*

If f is a *nonhomogeneous linear function* of P, i.e., if $f(P) = a^T P + b$, then the reader can verify that

$$f(P) = \theta_1 f(P_1) + \theta_2 f(P_2) + \theta_3 f(P_3). \tag{5.4.11}$$

this is a form of *linear interpolation on triangular grids.* In order to obtain *quadratic interpolation*, we define

$$\Delta'' = f(P_i) + f(P_j) - 2f\big(\frac{1}{2}(P_i + P_j)\big), \qquad i \neq j. \tag{5.4.12}$$

Theorem 5.4.1.

The interpolation formula

$$f(P) = \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 - 2(\theta_2\theta_3\Delta''_{23} + \theta_3\theta_1\Delta''_{31} + \theta_1\theta_2\Delta''_{12})$$

where $f_i = f(P_i)$, is exact for all quadratic functions.

Proof. The right-hand is a quadratic function of P, since it follows from (5.4.10) that the θ_i are (nonhomogeneous) linear functions of x, y. (See also Problem 8.) It remains to show that the right hand side is equal to $f(P)$ for $P = P_i$, and $P = (P_i + P_j)/2$, $i, j = 1, 2, 3$.

For $P = P_i$, $\theta_i = 1$, $\theta_j = 0$, $i \neq j$, hence the right hand side equals f_i. For $P = (P_i + P_j)/2$,

$$\theta_i = \theta_j = \frac{1}{2}, \quad \theta_k = 0, \qquad k \neq i, k \neq j,$$

and hence the right hand side becomes

$$\frac{1}{2}f_i + \frac{1}{2}f_j + -2 \cdot \frac{1}{2}\Big(f_i + f_j - 2u\big(\frac{1}{2}(P_i + P_j)\big)\Big) = f\big(\frac{1}{2}(P_i + P_j)\big).$$

□

The following theorem is equivalent to a rule which has been used in mechanics for the computation of moments of inertia since the nineteenth century:

Theorem 5.4.2.

Let A be the area of a triangle T, with vertices P_1, P_2, P_3. Then the quadrature formula

$$\int\int_T f(x, y)\, dxdy \tag{5.4.13}$$
$$= \frac{1}{3}A\left(f\Big(\frac{1}{2}(P_1 + P_2)\Big) + f\Big(\frac{1}{2}(P_2 + P_3)\Big) + f\Big(\frac{1}{2}(P_3 + P_1)\Big)\right.$$

is exact for all quadratic functions.

PROOF: By symmetry, $\int_T \int \theta_i\, dxdy$ is the same for $i = 1, 2, 3$. Similarly $\int_T \int \theta_i\theta_j\, dxdy$ is the same for all three (i, j)-combinations. Hence for the quadratic function

$$\begin{aligned}\int_T \int f(x, y)\, dxdy &= a(f_1 + f_2 + f_3) - 2b(\Delta''_{23} + \Delta''_{31} + \Delta''_{12}) \\ &= (a - 4b)(f_1 + f_2 + f_3) \\ &+ 4b\Big(f\big(\frac{1}{2}(P_1 + P_2)\big) + f\big(\frac{1}{2}(P_2 + P_3)\big) + f\big(\frac{1}{2}(P_3 + P_1)\big)\Big),\end{aligned}$$

where

$$a = \int_T \int \theta_1\, dxdy, \qquad b = \int_T \int \theta_1\theta_2\, dxdy.$$

Using θ_1, θ_2 as new variables of integration, we get by (5.4.10) and the relation $\theta_3 = 1 - \theta_1 - \theta_2$,

$$\begin{aligned}x &= \theta_1(x_1 - x_3) + \theta_1(x_1 - x_3) + x_3, \\ y &= \theta_1(y_1 - y_3) + \theta_1(y_1 - y_3) + y_3.\end{aligned}$$

Figure 5.4.4. *Correction for curved boundary segment.*

Hence the functional determinant is equal to

$$\begin{vmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{vmatrix} = 2A,$$

and (check the limits of integration!)

$$a = \int_{\theta_1=0}^{1} \int_{\theta_2=0}^{1-\theta_1} 2\theta_1 d\theta_1 d\theta_2 = 2A \int_0^1 \theta_1(1-\theta_1)d\theta_1 = \frac{A}{3},$$

$$b = \int_{\theta_1=0}^{1} \int_{\theta_2=0}^{1-\theta_1} 2\theta_1\theta_2 d\theta_1 d\theta_2 = 2A \int_0^1 \theta_1 \frac{(1-\theta_1)^2}{2} d\theta_1 = \frac{A}{3}.$$

The results now follows by insertion of this into (5.4.13). □

A numerical method can be based on Theorem 5.4.1, by covering the domain D by triangles. For each curved boundary segment (Fig. 8.4.4) the correction

$$\frac{4}{3} f(S) A(PRQ) \tag{5.4.14}$$

is to be added, where $A(PRQ)$ is the area of the triangle with vertices P, R, Q. The error of the correction can be shown to be $O(\|Q - P\|^5)$ for each segment, if R is close to the midpoint of the arc PQ. If the boundary is given in parametric form, $x = x(x)$, $y = y(x)$, where x and y are twice differentiable on the arc PQ, then one should choose $t_R = \frac{1}{2}(t_P + t_Q)$. Richardson extrapolation can be used to increase the accuracy, see the examples.

Figure 5.4.5. *The grids for I_4 and I_{16}.*

Example 5.4.4.
Consider the integral

$$I = \int\int_D (x^2 + y^2)^k \, dxdy$$

where D is the region shown in Fig. 8.4.5. Let I_n be the result obtained with n triangles. The grids for I_4 and I_{16} are shown in Fig. 8.4.5. Put

$$R'_n = I_{4n} + \frac{1}{15}(I_{4n} - I_n), \qquad R''_n = R'_{4n} + \frac{1}{63}(R'_{4n} - R'_n).$$

The following results were obtained. In this case the work could be reduced by a factor of 4, because of symmetry.

k	I_4	I_{16}	I_{64}	R'_4	R'_{16}	R''_4
2	0.250000	0.307291	0.310872	0.311111	0.311111	0.311111
3	0.104167	0.161784	0.170741	0.165625	0.171338	0.171429
4	0.046875	0.090678	0.104094	0.093598	0.104988	0.105169

The exact values are 0.311111, 0.171429, and 0.105397. It is seen that R'-values have full accuracy for $k = 2$ and the R''-values have high accuracy even for $k = 4$. In fact, it can be shown that R'-values are exact for any fourth-degree polynomial and R''-values are exact for any sixth-degree polynomial, when the region is covered exactly by the triangles.

Example 5.4.5.
The integral

$$a \int\int (a^2 - y^2)^{-1/2} \, dxdy$$

over a quarter of the unit circle is computed with the grids shown in Fig. 8.4.2, and with boundary corrections according to (5.4.9). The following results, using the notation of the previous example, were obtained and compared with the exact values:

a	I_8	I_{32}	R'_8	Exact
2	0.351995	0.352077	0.352082	0.352082
4	0.337492	0.337608	0.337615	0.337616
6	0.335084	0.335200	0.335207	0.335208
8	0.334259	0.334374	0.334382	0.334382

Note, however, that Richardson extrapolation may not always give improvement, e.g., when the rate of convergence of the basic method is *more rapid* than usual.

We mention also that some progress has been made in developing quadrature rules of optimal order for rectangles and triangles. In one dimension this led to Gaussian quadrature rules. In two dimensions the problem is much more difficult. Non-product rules for simple regions like a circle, equilateral triangle, regular hexagon, etc., can be found in Abramowitz and Stegun [1, pp. 891–895]. For a thorough treatment of multiple integrals the reader is referred to the book by Stroud [33].

5.4.5 Monte Carlo Methods

Quasi-Monte Carlo methods for numerical integration; see Niederreiter [27] Low discrepancy sequences

Lattice rules are equal weight rules for integration of periodic functions over the d-dimensional unit cube $[0,1]^d$. Thus the problem is to approximate the integral

$$If = \int_0^1 \cdots \int_0^1 f(x_1, \ldots, x_d)\, dx_1 \ldots dx_d, \tag{5.4.15}$$

by a rule

$$Q_N f = \frac{1}{N} \sum_{j=0}^{N-1} f\left(\left\{ \frac{j}{N} g \right\} \right), \tag{5.4.16}$$

where g is an d-dimensional integer vector that does not have N as a factor and by $\{x\} = \{x_1, \ldots, x_d$ we denote the vector whose jth component is the fractional part of x_j.

For numerical integration in high dimensions the number of function values needed to obtain an acceptable approximation tends to increase exponentially in the number of dimensions d. This is often referred to as *the curse of dimensionality*, a phrase coined by Richard Bellman. The exponential increase is clearly inevitable with any form of product integration rule. Recently it has been shown that the curse can be lifted by using a class of randomly shifted lattice rules by Ian H. Sloane.

One of the most important application of the Monte Carlo method described in Section 1.4.2 is in the numerical calculation of multiple integrals. If we use product rules to evaluate a multiple integral in d dimensions the work will depend exponentially on d. This means that the problem may quickly becomes intractable when d increases. On the other hand, for the Monte Carlo method the complexity always is proportional to $1/\epsilon$, where ϵ is the required tolerance *independent of the dimension* d. Hence the Monte Carlo method can be said to break "the curse of dimension" inherent in other approaches!

We shall briefly describe some ideas used in integration by the Monte Carlo method. For simplicity, we first consider integrals in *one* dimension, even though the Monte Carlo method cannot really compete with traditional numerical methods for this problem.

Let $R_1, R_2, \ldots, R_n$ be a sequence of random numbers rectangularly distributed on $[0,1]$, and set

$$I = \int_0^1 f(x)\, dx \approx I_1 = \frac{1}{n} \sum_{i=1}^{n} f(R_i).$$

This generalizes to multiple integrals. For example, to approximate a two dimensional integral over the domain $0 \le x, y \le 1$, we sample the integrand $f(x, y)$ in points (R_{2i-1}, R_{2i}), for $i = 1, 2, \ldots, n$. The technique can be applied to an integral over a general region D, provided that we can sample the integrand f randomly over D.

One can show that the expectation of the variable I_1 is I and that the standard deviation of this estimate decreases in proportion to $n^{-1/2}$. This is very slow even compared to the trapezoidal rule—where the error decreases as n^{-2}. To get one extra decimal place of accuracy we must increase the number of points by a factor of 100. To get three digit accuracy the order of one million points may be required! However, if we consider, e.g., a six-dimensional integral this is not exorbitant. Using a product rule with 10 subdivisions in each dimension would also require 10^6 points.

The above estimate is a special case of a more general one. Suppose X_i $i = 1, 2, \ldots, n$, has density function $g(x)$. Then

$$I_2 = \frac{1}{n} \sum_{i=1}^{n} \frac{f(X_i)}{g(X_i)}$$

has expected value I, since

$$E\left(\frac{f(X_i)}{g(X_i)}\right) = \int_0^1 \frac{f(x)}{g(x)} f(x)\, dx = \int_0^1 f(x)\, dx = I.$$

If one can find a frequency function $g(x)$ such that $f(x)/g(x)$ fluctuates less than $f(x)$, then I_2 will have smaller variance than I_1. This procedure is called **importance sampling**; it has proved very useful in particle-physics problems, where important phenomena (e.g., dangerous radiation which penetrates a shield) are associated with certain events of low probability.

We have previously mentioned the method of using a simple comparison problem. The Monte Carlo variant of this method is called the **control variate method**. Suppose that $\varphi(x)$ is a function whose integral has a known value K, and suppose that $f(x) - \varphi(x)$ fluctuates much less than $f(x)$. Then

$$I = K + \int_0^1 (f(x) - \varphi(x))\, dx \approx K + I_3, \qquad I_3 = \frac{1}{n} \sum_{i=1}^{n} (f(R_i) - \varphi(R_i)),$$

where I_3 has less variance than I_1.

Review Questions

1. How is bilinear interpolation performed? What is the order of accuracy?
2. Define barycentric coordinates, and give the formula for linear interpolation on a triangular grid.

3. Describe the methods for numerical integration with rectangular or triangular grids.

Problems

1. Let D be the unit circle. Introduce polar coordinates in the integral

$$I = \int\int_D \frac{y\sin(ky)}{x^2+y^2}\,dxdy$$

and reduce it analytically to a single integral.

2. Let E be the ellipse $\{(x,y) \mid (x/a)^2 + (y/b)^2 \le 1\}$. Transform

$$I = \int\int_E f(x,y)\,dxdy$$

into an integral over a rectangle in the (r,t)-plane with the transformation $x = ar\cos t$, $y = br\sin t$.

3. Compute by bilinear interpolation $u(0.5, 0.25)$ when

$$u(0,0) = 1, \quad u(1,0) = 2, \quad u(0,1) = 3, \quad u(1,1) = 5.$$

4. Show that, using the notation for equidistant rectangular grids, the formula

$$\int_{x_0-h}^{x_0+h}\int_{y_0-k}^{y_0+k} f(x,y)\,dxdy = \frac{4hk}{6}(f_{1,0} + f_{0,1} + f_{-1,0} + f_{0,-1} + 2f_{0,0})$$

is exact for all cubic polynomials.

5. Is a quadratic polynomial uniquely determined, given six functions values at the vertices and midpoints of the sides of a triangle?

6. Show that the boundary correction of (5.4.9) is exact if $f \equiv 1$, and if the arc is a parabola where the tangent at R is parallel to PQ.

7. Formulate generalizations to several dimensions of the integral formula of Theorem 5.4.1, and convince yourself of their validity.

Hint: The formula is most simply expressed in terms of the values in the vertices and in the centroid of a simplex.

8. (a) Write a program which uses the Monte Carlo method to compute $\int_0^1 e^x\,dx$. Take 25, 100, 225, 400 and 635 points. Plot the error on a loglog-scale. How does the error depend (approximately) on the number of points?

(b) Compute the integral in (a) using the control variate method. Take $\varphi(x) = 1 + x + x^2/2$. Use the same number of points as in (a).

Notes and References

A comprehensive treatment of the numerical evaluation of integrals is given in Davis and Rabinowitz [6]. Alternatively the Newton–Cotes and other quadrature rules can be derived using computer algebra systems, see [10].

For a history of Gauss-type quadrature rules, see Gautschi [13]. Gaussian quadrature rules were derived by Gauss in 1814 using a continued fraction expansion related to the hypergeometric series. In 1826 Jacobi showed that the nodes were the zeros of the Legendre polynomials and that they were real, simple and in $[-1, 1]$. The convergence of Gaussian quadrature methods was first studied by Stieltjes in 1884. A software package in the public domain by Gautschi [14] includes routines for generating Gauss-type formulas and orthogonal polynomials not only for classical but also for essentially arbitrary weight functions. The presentation in Sec. 5.2.3 is inspired by the work of Gautschi [12], [15], Golub and co-authors. Related ideas can be traced to Mysovskih [26].

The classical reference on orthogonal polynomials is Szegö [35]. Tables of abscissas and weights for Gaussian quadrature rules with various weight functions are given in Abramowitz and Stegun [1, Sec. 25] and in Gautschi [13]. A computer package for computing the tridiagonal Jacobi matrix and generating the corresponding Gauss quadrature rule has been developed by Gautschi [14]. Maple programs for Gauss quadrature rules ar given by von Matt [25].

The idea of adaptive Simpson quadrature is old and treated fully by Lyness [24]. Further schemes, computer programs and examples are given in Davis and Rabinowitz [6]. For a recent discussion of error estimates and reliability of different codes see Espelid [8].

Multivariate integration formulas and lattice rules are discussed in [36].

Bibliography

[1] Milton Abramowitz and Irene A. Stegun (eds.). *Handbook of Mathematical Functions*. Dover, New York, NY, 1965.

[2] F. L. Bauer, H. Rutishauser, and E. Stiefel. New aspects in numerical quadrature. In *Proc. of Symposia in Appl. Math.*, volume 15, pages 199–218. Amer. Math. Soc., 1963.

[3] Claude Brezinski. *Padé-Type Approximations and General Orthogonal Polynomials*. Birkhäuser Verlag, Basel, 1980.

[4] R. Bulirsch. Bemerkungen zur Romberg-Integration. *Numer. Math.*, 6:6–16, 1858.

[5] E. B. Christoffel. Über die Gaussische Quadratur und einige Verallgemenierung derselbern. *J. Reine Angew. Math.*, 55:61–82, 1858.

[6] P. J. Davis and Philip Rabinowitz. *Methods of Numerical Integration*. Academic Press, Orlando, FL, second edition, 1984.

[7] H. Engels. *Numerical Quadrature and Cubature*. Academic Press, London, 1980.

[8] Terje O. Espelid. Doubly adaptive quadrature routines based on Newton–Cotes rules. *BIT*, 43:319–337, 2003.

[9] Walter Gander and Walter Gautschi. Adaptive quadrature—revisited. *BIT*, 40:84–101, 2000.

[10] Walter Gander and Dominik Gruntz. Derivation of numerical methods using computer algebra. *SIAM Review*, 41:3:577–593, 1999.

[11] C. F. Gauss. Methodus nova integralium valores per approximationem inveniendi. *Comm. Soc. R. Sci. Gött. Recens*, 3:39–76, 1814.

[12] Walter Gautschi. On the construction of Gaussian quadrature rules from modified moments. *Math. Comp.*, 24:245–260, 1970.

[13] Walter Gautschi. A survey of Gauss–Christoffel quadrature formulae. In P. L. Butzer and F. Fehér, editors, *E. B. Christoffel: The influence of his work on mathematics and the physical sciences*, pages 72–147. Birkhäuser, Basel, 1981.

[14] Walter Gautschi. Algorithm 726: ORTHPOL—a package for generating orthogonal polynomials and Gauss-type quadrature rules. *Trans. Math. Software*, 20:21–62, 1994.

[15] Walter Gautschi. Orthogonal polynomials: applications and computation. *Acta Numerica*, 5:45–119, 1996.

[16] Walter Gautschi. *Numerical Analysis, an Introduction*. Birkhäuser, Boston, MA, 1997.

[17] Gene H. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15:318–334, 1973.

[18] Gene H. Golub and J. H. Welsch. Calculation of Gauss quadrature rules. *Math. Comp.*, 23:221–230, 1969.

[19] Tore Håvie. Generalized Neville type extrapolation schemes. *BIT*, 19:204–213, 1979.

[20] David Kahaner, Cleve B. Moler, and Stephen Nash. *Numerical Methods and Software*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[21] A. S. Kronrod. *Nodes and Weights of Quadrature Formulas*. Consultants Bureau, New York, 1965. Translation from Russian.

[22] Cornelius Lanczos. *Linear Differential Operators*. D. Van Nostrand, London, UK, 1961.

[23] Bengt Lindberg. A simple interpolation algorithm for improvement of the numerical solution of a differential equation. *SIAM J. Numer. Anal.*, 9:662–668, 1972.

[24] J. N. Lyness. Notes on the adaptive Simpson quadrature routine. *J. Assoc. Comput. Mach.*, 16:483–495, 1969.

[25] Urs von Matt. Gauss quadrature. In W. Gander and J. Hřebiček, editors, *Solving Problems in Scientific Computing using MAPLE and MATLAB*, pages 251–279. Springer-Verlag, Berlin, 1997.

[26] I. P. Mysovskih. On the construction of cubature formulas with the smallest number of nodes. *Soviet Math. Dokl.*, 9:277–280, 1968.

[27] H. Niederreiter. Quasi-Monte Carlo methods for numerical integration. In G. Hämmerling and H. Brass, editors, *Numerical Integration III*, pages 157–171. Birkhäuser Verlag, 1988.

[28] R. Piessens, E. de Doncker, C. W. Überhuber, and D. K. Kahaner. *QUADPACK, A Subroutine Package for Automatic Integration*. Springer-Verlag, Berlin, 1983.

[29] W. Romberg. Vereinfachte numerische Integration. *Kong. Norske Videnskabers Selskab Forhandlinger*, 28:7, 1955.

[30] I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration.* Clarendon Press, Oxford, UK, 1994.

[31] J. F. Steffensen. *Interpolation.* Chelsea, New York, second edition, 1950.

[32] Joseph Stoer and Roland Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, New York, third edition, 2002.

[33] A. H. Stroud. *Approximate Calculation of Multiple Integrals.* Prentice-Hall, Englewood Cliffs, NJ, 1972.

[34] A. H. Stroud and D. Secrest. *Gaussian Quadrature Formulas.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[35] Gabor Szegö. *Orthogonal Polynomials*, volume 23 of *Colloq. Publ.* Amer. Math. Soc., Providence, RI, fourth edition, 1975.

[36] Christoph W. Ueberhuber. *Numerical Computation: Methods, Software, and Analysis. Volumes 1 & 2.* Springer-Verlag, Berlin, 1997.

For the derivation of error estimates for numerical integration we shall require the following result on the continuity of divided differences. For this purpose the following representation of divided differences is useful. If $x, x_1, \ldots, x_n$ be $n+1$ are distinct points, then

$$[x_1, \ldots, x_n, x]f = \sum_{j=1}^{n} \frac{[x, x_j]f}{\prod\limits_{\substack{k=1 \\ k \neq j}}^{n} (x_j - x_k)}. \tag{5.4.17}$$

This formula follows by substituting the Lagrange form of the interpolation polynomial into the exact remainder (4.2.19) in Newton's interpolation formula.

Lemma 5.4.3.

Let $f(x)$ be continuous on $[a, b]$ and let $f'(x)$ be continuous in arbitrary small intervals about some distinct fixed points $x_i \in [a, b]$, $i = 1 : n$. Then

$$[x_1, \ldots, x_n, x]f$$

is a continuous function of x in $[a, b]$.

Index

algorithm
    adaptive Simpson, 53

barycentric coordinates, 62

companion matrix, 34
coordinates
    barycentric, 62

epsilon algorithm, 49
Euler-Maclaurin's formula, 42, 43

FEM, *see* finite element method
finite element method, 61
Fourier transform, 42

Gauss' quadrature, 22–38
    remainder, 26
Gauss–Christoffel quadrature, 22–38
Gauss–Hermite quadrature, 29
Gauss–Jacobi quadrature, 28
Gauss–Kronrod quadrature, 30
Gauss–Laguerre quadrature, 29
Gauss–Legendre quadrature, 27
Gauss–Lobatto quadrature, 30
Gauss–Radau quadrature, 30
Gram matrix, 32
Gregory's quadrature formula, 42
grid
    irregular triangular, 61–66
    rectangular, 61

Hankel matrix, 32
Hermite interpolation, 26
Hermite polynomials, 29, 39
Hilbert matrix, 32

importance sampling, 67
integral
    over infinite interval, 51
    with singularity, 13–15
integration
    by parts, 13
interpolatory quadrature formula, 2

Jacobi polynomials, 28

Laguerre polynomials, 29
Legendre polynomials, 27
linear interpolation
    on triangular grid, 62
Low discrepancy sequences, 66

midpoint rule
    composite, 5
modified moments, 31
moment, 13
multiple integrals, 57–67

Newton–Cotes'
    9-point formula, 18
    quadrature rule, 8–22, 47
node polynomial, 18, 20, 21
node polynomials, 19
numerical quadrature
    Newton–Cotes, 47

order of accuracy, 1
orthogonal polynomials, 22–24
oscillating integrand, 15

Poisson summation formula, 42, 50

quadratic interpolation
    on triangular grid, 62
quadrature

Monte Carlo methods, 66–67
quadrature rule
adaptive, 52
closed, 5
midpoint, 5
Newton–Cotes', 8–22
open, 5
product, 60–61
Simpson's, 6–8
successive one-dimensional, 59–60
trapezoidal, 3
Quasi-Monte Carlo methods, 66

Richardson extrapolation, 42
Romberg's method, 41–49
error bound, 45

shift matrix, 34
Simpson's formula, 16
singular integrand, 13
Stieltjes integral, 57

transform
Fourier, 42
trapezoidal rule
composite, 4
superconvergence, 50
triangular grid
linear interpolation on, 62
quadratic interpolation on, 62
refinement of, 61
truncation error
global, 4
local, 4

weight function, 12
weighted quadrature rules, 12

Contents

6 Solving Scalar Nonlinear Equations **1**

6.1 Some Basic Concepts and Methods 1
6.1.1 Introduction 1
6.1.2 The Bisection Method 2
6.1.3 Attainable Accuracy and Termination Criteria . . . 6
6.1.4 Fixed-Point Iteration 10
6.1.5 Convergence Order and Efficiency 13
Review Questions 16
Problems and Computer Exercises 17
6.2 Methods Based on Interpolation 18
6.2.1 The Method of False Position 18
6.2.2 The Secant Method 20
6.2.3 Higher Order Interpolating Methods 23
6.2.4 A Robust Hybrid Method 26
Review Questions 26
Problems and Computer Exercises 27
6.3 Methods Using Derivatives 28
6.3.1 Newton's method 28
6.3.2 Global Convergence of Newton's Method 32
6.3.3 Newton Method for Complex Roots 34
6.3.4 An Interval Newton Method 36
6.3.5 Higher Order Methods 38
Review Questions 42
Problems and Computer Exercises 43
6.4 Local Minimum of a Scalar Function 46
6.4.1 Unimodal Functions 47
6.4.2 Golden Section Search 47
6.4.3 Minimization by Interpolation 50
Review Questions 52
Problems and Computer Exercises 52
6.5 Zeros of Polynomials 53
6.5.1 Introduction 53
6.5.2 Some Basic Formulas 56
6.5.3 Sturm Sequences 58

6.5.4 Deflation and Zero Suppression 61
6.5.5 Simultaneous Determination of Roots 62
6.5.6 A Modified Newton Method 64
6.5.7 Laguerre's Method 66
Review Questions 67
Problems and Computer Exercises 68

Index **74**

Chapter 6

Solving Scalar Nonlinear Equations

6.1 Some Basic Concepts and Methods

6.1.1 Introduction

In this chapter we study numerical methods for computing accurate approximations to the roots of a scalar nonlinear equation

$$f(x) = 0, \tag{6.1.1}$$

where $f(x)$ is a real-valued function of one variable. This problem has occupied mathematicians for many centuries and many of the basic methods date back a long time. In general the roots of (6.1.1) cannot be expressed in closed form. Even when an explicit solution is available (as, e.g., for the reduced cubic equation), this is often so complicated that using, e.g., Newton's method, is more practical; see Problem 2.3.8.

Numerical methods are iterative in nature. Starting from one or more initial approximations, they produce a sequence of approximations, which presumably converges to the desired root. Note that the function $f(x)$ need not be known by a closed analytical expression. For numerical methods to be applicable it suffices that $f(x)$, and preferably some of its derivatives, can be evaluated for given numerical values of x. It is not uncommon with applications where each function value is obtained by a complicated computation, e.g., by the numerical solution of a differential equation. The object is then to use as few function evaluations as possible in order to approximate the root with a prescribed accuracy.

Iterative methods have to be truncated after a finite number of steps and therefore can yield only approximations to the desired roots. Further, the roundoff errors that occur in the evaluation of $f(x)$ will limit the accuracy attainable by *any numerical method*. The effect of such rounding errors depends on the conditioning of the roots and is discussed in Section 6.1.3.

With certain methods it is sufficient for convergence to know an initial interval $[a, b]$, which contains the desired root (and no other root). An important example

is the bisection method described in Section 6.1.2. It is often suitable to use a hybrid method in which the bisection method is used to roughly locate the root. A more rapidly convergent method is then used to refine this approximation. These latter methods make more use of regularity assumptions about $f(x)$, and usually also require an initial approximation close to the desired root.

The theory of fixed point iteration methods is treated in Sec. 6.1.4 and the concepts of convergence order and efficiency introduced in Sec. 6.1.5. The secant method, and other methods based on interpolation are described in Sec. 6.2. In Sec. 6.4 we briefly consider methods for solving the related problem of finding the minimum or maximum of a real-valued function $g(x)$. Newton's method and other methods of higher order are analyzed in Sections 6.3. A classical problem is that of determining all real or complex roots of a algebraic equation. Special features and methods for this problem are taken up in Section 6.5.

Many of the methods for a single equation, such as Newton's method, are easily generalized for *systems of nonlinear equations*. However, unless good approximations to the roots are known, several modifications of the basic methods are required, see Vol. II, Chapter 11.

6.1.2 The Bisection Method

It is often advisable to start with collecting some qualitative information about the roots to be computed. One should try to determine how many roots there are and their approximate location. Such information can often be obtained by graphing the function $f(x)$. This can be a useful tool for determining the number of roots and intervals containing each root.

Example 6.1.1.
Consider the equation

$$f(x) = (x/2)^2 - \sin x = 0.$$

In Figure 6.1.2 the graphs of $y = (x/2)^2$ and $y = \sin x$ are shown. Observing the intersection of these we find that the unique positive root lies in the interval $(1.8, 2)$, probably close to $\alpha \approx x_0 = 1.9$.

The following **intermediate-value theorem** can be used to infer that an interval $[a, b]$ contains *at least one root* of $f(x) = 0$.

Theorem 6.1.1.
Assume that the function $f(x)$ is continuous for $a \le x \le b$, $f(a) \neq f(b)$, and k is between $f(a)$ and $f(b)$. Then there is a point $\xi \in (a, b)$, such that $f(\xi) = k$. In particular, if $f(a)f(b) < 0$ then the equation $f(x) = 0$ has at least one root in the interval (a, b).

A systematic use of the intermediate-value theorem is made in the **bisection method**. Assume that $f(x)$ is continuous in the interval (a_0, b_0) and that

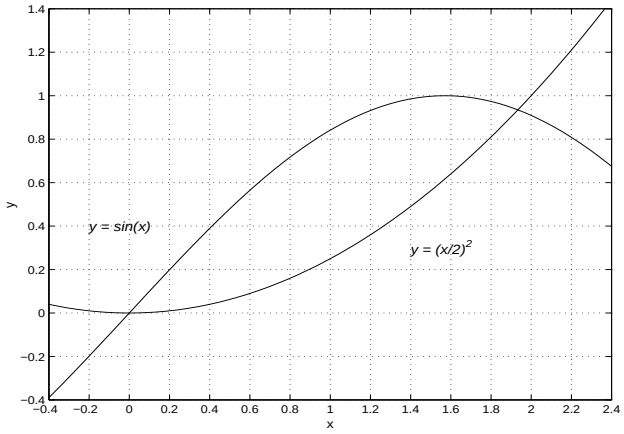


Figure 6.1.1. *Graph of curve* $y = (x/2)^2 - \sin x$.

$f(a_0)f(b_0) < 0$. We shall determine a nested sequence of intervals $I_k = (a_k, b_k)$, $k = 1, 2, 3, \ldots$, such that

$$(a_0, b_0) \supset (a_1, b_1) \supset (a_2, b_2) \subset \cdots$$

and which all contain a root of the equation $f(x) = 0$. The intervals are determined recursively as follows. Given $I_k = (a_k, b_k)$ compute the midpoint

$$m_k = \frac{1}{2}(a_k + b_k) = a_k + \tfrac{1}{2}(b_k - a_k).. \tag{6.1.2}$$

and $f(m_k)$. The latter expression has the advantage that using this to compute the midpoint no rounding error occurs in the subtraction (see Theorem 2.2.2).

We can assume that $f(m_k) \neq 0$, since otherwise we have found a root. The new interval $I_{k+1} = (a_{k+1}, b_{k+1})$ is then determined by the rule

$$(a_{k+1}, b_{k+1}) = \begin{cases} (m_k, b_k), & \text{if } f(m_k)f(a_k) > 0; \\ (a_k, m_k), & \text{if } f(m_k)f(a_k) < 0. \end{cases} \tag{6.1.3}$$

From the construction it follows immediately that $f(a_{k+1})f(b_{k+1}) < 0$ (see also Figure 6.1.2) and therefore the interval I_{k+1} also contains a root of $f(x) = 0$.

After n bisection steps we have contained a root in the interval (a_n, b_n) of length $2^{-n}(b_0 - a_0)$. If we take m_n as an estimate of the root α, we have the error estimate

$$|\alpha - m_n| < 2^{-(n+1)}(b_0 - a_0). \tag{6.1.4}$$

At each step we gain one binary digit in accuracy or, since $10^{-1} \approx 2^{-3.3}$, on the average one decimal digit per 3.3 steps. To find an interval of length δ which includes a root will require about $\log_2((b-a)/\delta)$ evaluations of f. Note that the bisection algorithm makes no quantitative use of the magnitude of computed function values.

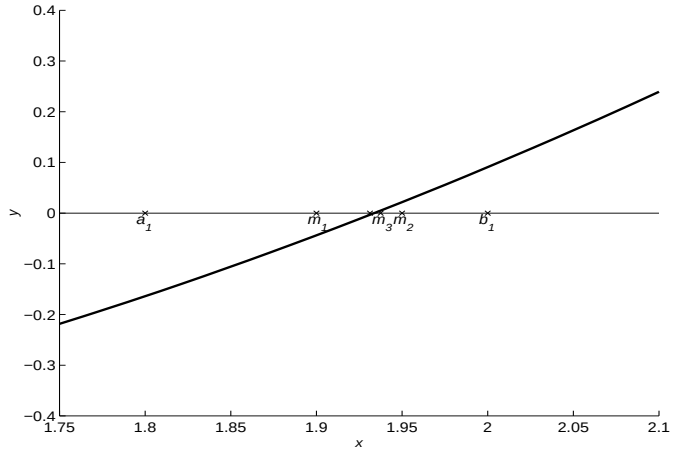


Figure 6.1.2. *The bisection method.*

Example 6.1.2.

The bisection method applied to the equation $(x/2)^2 - \sin x = 0$, with $I_0 = (1.8, 2)$ gives the sequence of intervals $[a_n, b_n], where$:

k	a_k	b_k	m_k	$f(m_k)$
1	1.8	2	1.9	<0
2	1.9	2	1.95	>0
3	1.9	1.95	1.925	<0
4	1.925	1.95	1.9375	>0
5	1.925	1.9375	1.93125	<0
6	1.93125	1.9375	1.934375	>0

Here after six function evaluations we have $\alpha \in (1.93125, 1.934375)$ an interval of length $0.2 \cdot 2^{-6} = 0.003125$.

Example 6.1.3.

The inequalities $a \leq \frac{1}{2}(a+b) \leq b$, where a and b are floating point numbers with $a \leq b$ can be violated in base 10 arithmetic. For example, assume that floating point arithmetic with six decimal digits is used. Taking $a = 0.742531$ and $b = 0.742533$ we obtain $fl(a+b) = 1.48506$ (rounded) and $\frac{1}{2}(a+b) = 0.742530$. On the other hand the inequalities $a \leq a + \frac{1}{2}(b-a) \leq b$ are true in base β arithmetic, for any β. With a and p as given we get the correct value 0.742532.

An algorithmic description of the bisection method is given below. In this the tolerance τ is increased by the amount $u \max(|a|, |b|)$, where u is the machine precision. This is to guard against the possibility that δ has been chosen smaller than the spacing between the floating point numbers between a and b.

Algorithm 6.1.1 *The Bisection Method.*

Let f be a given function and $I = [a, b]$ an interval such that $b > a$ and $f(a)f(b) \leq 0$. The algorithm bisect attempts to compute an approximation to a root $m \in I$ of $f(x) = 0$, with an error less than a specified tolerance $\tau > 0$.

```
function  r = bisect(f, a, b, τ);
fa = f(a);
fb = f(b);
while |b − a| > τ + u · max((|a|, |b|);
   m = a + (b − a)/2;
   fm = f(m);
   if fm · fa ≤ 0
     b = m;   fb = fm;
   else
     a = m;   fa = fm;
   end;
end;
r = a + (b − a)/2;
```

The time required by the bisection algorithm is typically proportional to the number of function evaluations, other arithmetic operations being insignificant. The correct subinterval will be chosen in the algorithm as long as the sign of the computed function value $f(m)$ is correctly determined. If the tolerance τ is taken too "small" or the root is ill-conditioned this may fail to be true in the later steps. Even then the computed midpoints will stay within a certain domain of uncertainty. Due to rounding errors there is a limiting accuracy, with which a root can be determined from approximate function values; see in Section 6.1.3.

The bisection method is optimal for the class of functions that changes sign on $[a, b]$ in the sense that it minimizes the maximum number of steps over all such functions. The convergence is rather slow, but *independent of the regularity of* $f(x)$. For other classes of functions, e.g., functions that continuously differentiable on $[a, b]$, methods like Newton's method, which assume some regularity of $f(x)$ can achieve significantly faster convergence.

If $f(a)f(b) < 0$ then by the intermediate value theorem the interval (a, b) contains *at least one root* of $f(x) = 0$. If the interval (a, b) contains several roots of $f(x) = 0$, then the bisection method will converge to just one of these. (Note that there may be one or several roots in (a, b), also in case $f(a)f(b) > 0$.)

If we only know (say) a lower bound $a < \alpha$ for the root to be determined we can proceed as follows. We choose an initial steplength d and in the first **hunting phase** compute successively function values $f(a + h)$, $f(a + 2h)$, $f(a + 4h), \ldots$, i.e. we double the step, until a function value is found such that $f(a)f(a + 2^k h) < 0$. At this point we have bracketed a root and can initiate the bisection algorithm.

In the bisection method the interval of interest is in each step split into *two* subintervals. An obvious generalization is to partition instead into k subintervals,

for $p \geq 2$. In such a **multi-section method** of order p the interval $I = [a, b]$ is divided into k subintervals $I_i = [x_i, x_{i+1}]$, where

$$x_i = a + i\,[(b-a)/p], \quad i = 0 : p.$$

If there exists only one root in the interval I and we wish to compute it with an absolute error ϵ, then it is necessary to perform

$$n_k = \log_2 \Big(\frac{b-a}{2\epsilon}\Big) \Big/ \log_2(p)$$

multi-sections of order p. Thus, the efficiency of multi-section of order p compared to bisection ($p = 2$) is

$$n_2/(pn_p) = \log_2(p)/p.$$

Hence if there is a single root in the interval bisection is always preferable. If there are several roots in the interval multi-section may perform better if the subintervals can be processed in parallel.

There are several other applications of the bisection algorithm. For example, in Section 4.4.5 we considered evaluating the nonzero B-splines for a given argument x. Then we first have to search an ordered sequence of knots $\tau_0, \ldots, \tau_m$ to find the interval such that $\tau_j \leq x < \tau_{j+1}$. This can be achieved by a slight modification of the bisection method. A similar problem, important in computer science, is *searching in an ordered register*, e.g., a register of employees ordered according to increasing Social Security number. If the nth number in the register is denoted by $f(n)$, then searching for a certain number a means that an equation $f(n) = a$ is to be solved (here f is an increasing, discontinuous function). The bisection method can also be used in searching an *alphabetically* ordered register.

In later sections we will study methods for solving a nonlinear equation, which make more efficient use of computed function values than the bisection method and possibly also use values of derivatives of $f(x)$. If $f(x)$ is sufficiently regular such methods can achieve significantly faster convergence.

6.1.3 Attainable Accuracy and Termination Criteria

In the following we denote by $\overline{f}(x) = fl(f(x))$ the limited-precision approximation obtained when $f(x)$ is evaluated in floating point arithmetic. When a monotone function $f(x)$ is evaluated in floating point arithmetic the resulting approximation $\overline{f}(x)$ is not in general monotone. The effect of rounding errors in evaluating a certain polynomial of fifth degree with a simple zero at $x = 1$ is illustrated in Figure 6.1.4. Note the loss of monotonicity caused by rounding errors. This figure also shows that even if $\overline{f}(a)\overline{f}(b) < 0$, the true equation $f(x) = 0$ may not have a zero in $[a, b]$!

Even if the true function value $|f(x_n)|$ is "small" one cannot deduce that x_n is close to a zero of $f(x)$ without some assumption about the size of the derivative of f. We recall some basic results from analysis; for proofs see, e.g., Ostrowski [19, Chapter 2].

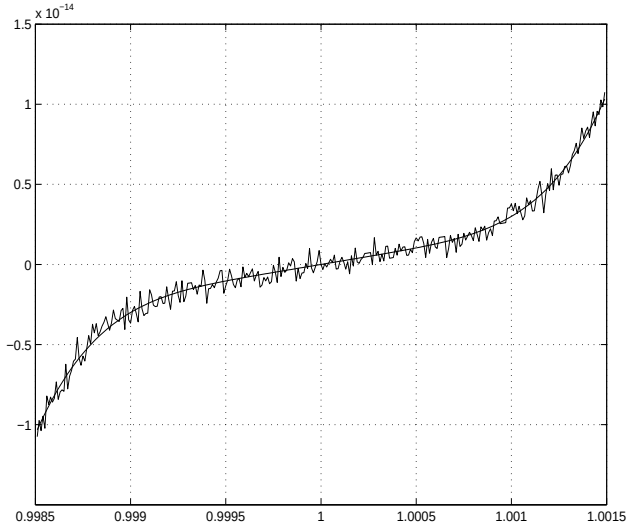


Figure 6.1.3. *Limited-precision approximation of a continuous function.*

Theorem 6.1.2.
Let $f(x)$ be continuous and differentiable in the interval $J = [x_n - \eta, x_n + \eta]$ for some $\eta > 0$. If $|f'(x)| \geq m_1$ for all $x \in J$ and $|f(x_n)| \leq \eta m_1$ then $f(x)$ has exactly one zero in J.

A root α of $f(x) = 0$ is said to be **simple** root if $f'(\alpha) \neq 0$. We now derive an error estimate for a simple root α of $f(x)$, which takes into account errors in the computed values of $f(x)$. Assume that

$$\overline{f}(x) = f(x) + \delta(x), \quad |\delta(x)| \leq \delta, \quad x \in J, \tag{6.1.5}$$

where δ is an upper bound for rounding and other errors in computed function values of $f(x)$. Using Theorem 6.1.2 we obtain

$$|x_n - \alpha| \leq \eta = (|\overline{f}(x_n)| + \delta)/m_1, \quad |f'(x)| \geq m_1, \quad x \in J. \tag{6.1.6}$$

Obviously the best we can hope for is to find an approximation x_n such that the computed function value $\overline{f}(x_n) = 0$. It follows that *for any numerical method*, δ/m_1 is an approximate limit for the accuracy with which a simple zero α can be determined. If $f'(x)$ does not vary much near $x_n = \alpha$, then we have the approximate error bound

$$|x_n - \alpha| \leq \delta/m_1 \approx \epsilon_\alpha, \quad \epsilon_\alpha = \delta/|f'(\alpha)|. \tag{6.1.7}$$

Since this is the best error bound for *any* method, we call ϵ_α, the **attainable accuracy** for the simple root α, and the interval $[\alpha - \epsilon_\alpha, \alpha + \epsilon_\alpha]$ the **domain of uncertainty** for the root α. If $|f'(\alpha)|$ is small, then ϵ_α is large and the problem of computing the root α is ill-conditioned (see again Figure 6.1.3).

Example 6.1.4.

Suppose we have computed the approximation $x = 1.93375$ to the positive root to the equation $f(x) = \sin x - (x/2)^2$. We have $f'(x) = \cos x - x/2$ and it is easily verified that $|f(x)| > 1.31 = m_1$, $x \in [1.93, 1.94]$. Further, using six decimals we have $\sin 1.93375 = 0.934852 \pm 0.510^{-6}$, and $(x/2)^2 = 0.966875)^2 = 0.934847 \pm 0.510^{-6}$. Then from (6.1.6) follows the strict error estimate

$$|x - \alpha| < 6 \cdot 10^{-6}/1.31 < 5.6 \cdot 10^{-6}.$$

Using the following theorem, an analogous result can be shown for zeros of a complex function $f(z)$ of a complex variable z.

Theorem 6.1.3.

Let $f(z)$ be analytic in the disc $K = \{z \mid |z - z_0| \leq \eta\}$ for some $\eta > 0$. If $|f'(z)| \geq m$ in K and $|f(z_0)| \leq \eta m$ then $f(z)$ has a zero inside K.

The **multiplicity** of a root is defined as follows:

Definition 6.1.4.

Suppose that $f(x)$ is q times continuously differentiable in a neighborhood of a root α to the equation $f(x) = 0$. Then α is said to have multiplicity q if

$$0 \neq \lim_{x \to \alpha} |f(x)/(x - \alpha)^q| < \infty. \tag{6.1.8}$$

If a root α has multiplicity q then by (6.1.8) $f^{(j)}(\alpha) = 0$, $j < q$ and from Taylor's formula

$$f(x) = \frac{1}{q!}(x - \alpha)^q f^{(q)}(\xi), \quad \xi \in \text{int}(x, \alpha). \tag{6.1.9}$$

Assuming that $|f^{(q)}(x)| \geq m_q$, $x \in J$, and proceeding as before, we find that the attainable accuracy for a root of multiplicity q is given by

$$|x_n - \alpha| \leq (q!\,\delta/m_q)^{1/q} \approx \epsilon_\alpha, \quad \epsilon_\alpha = (q!\,\delta/|f^{(q)}(\alpha)|)^{1/q}. \tag{6.1.10}$$

Comparing this with (6.1.7), we see that because of the exponent $1/q$ multiple roots are in general very ill-conditioned. A similar behavior can be expected also when there are several distinct but "close" roots. An instructive example is the Wilkinson polynomial, studied in Example 6.5.1.

Example 6.1.5.

The equation $f(x) = (x - 2)x + 1 = 0$ has a double root $x = 1$. The (exact) value of the function at $x = 1 + \epsilon$ is

$$f(x + \epsilon) = (\epsilon - 1)(1 + \epsilon) + 1 = -(1 - \epsilon^2) + 1 = \epsilon^2.$$

Now, suppose that we use a floating point arithmetic with eight decimal digits in the mantissa. Then

$$fl(1-\epsilon^2)=1, \quad |\epsilon| < \frac{1}{2}\sqrt{2}\cdot 10^{-4},$$

and for $0.99992929 \le x \le 1.0000707$, the computed value of $f(x)$ will be zero when $f(x)$ is evaluated using Horner's rule. Hence the root can only be computed with about four correct digits, that is, with a relative error equal to the *square root of the machine precision.*

Suppose that we want to compute an approximation to a simple root α to a prescribed accuracy. Provided that the absolute value of the derivative is easy to estimate it may be possible to interrupt the iterations on the basis of the error estimate (6.1.7). However, on a computer it is usually more effective to iterate a few extra times, rather than make the effort to use a special formula for error estimation.

In subroutines for solving a nonlinear equation it is common practice to use a termination criterion of the following form. Assuming the method produces a seqeunce of bracketing intervals $[a_k, b_k]$ the iterations are terminated if

$$|b_k - a_k| \le 2u|x_n| + \tau, \tag{6.1.11}$$

where τ is a user specified absolute tolerance and u is the rounding unit (see Section 2.2).

We must also deal with the possibility that the user specified tolerance is too small and cannot be attained. If this is the case, then from some step onwards rounding errors will dominate in the evaluation of $f(x_n)$ and the computed values of $f(x)$ may vary quasi-randomly in the interval $(-\delta, \delta)$ of attainable accuracy. If we are using a method like the bisection method, the iterations will continue until the criterion (6.1.11) is satisfied, but this, of course, does *not* ensure that the root actually has been determined to this precision!

The following alternative termination criterion can be used for superlinearly convergent methods:[1] *Accept the approximation x_n when for the first time the following two conditions are satisfied:*

$$|x_{n+1} - x_n| \ge |x_n - x_{n-1}|, \qquad |x_n - x_{n-1}| < tol. \tag{6.1.12}$$

Here *tol* is a coarse tolerance, used only to prevent the iterations from being terminated before x_n even has come close to α. When (6.1.12) is satisfied the attainable accuracy has been reached and the quantity $|x_{n+1} - x_n|$ usually is a good estimate of the error $|x_n - \alpha|$. Using this criterion the risk of never terminating the iterations for an ill-conditioned root is quite small. Note also that iteration methods of superlinear convergence ultimately converge so fast that the cost of always iterating until the attainable accuracy is obtained may be small, even if the user specified tolerance is much larger than ϵ_α.

[1] This criterion was suggested by the Norwegian computer scientist Jan Garwick.

6.1.4 Fixed-Point Iteration

We now introduce a very general class of iteration methods, which includes many important root finding methods as special cases.

Let ϕ be a continuous function and $\{x_n\}$ the sequence generated by

$$x_{n+1} = \phi(x_n), \quad n = 0, 1, 2, \ldots . \tag{6.1.13}$$

for some initial value x_0. Assuming that $\lim_{n\to\infty} x_n = \alpha$, it follows that

$$\alpha = \lim_{n\to\infty} x_n = \lim_{n\to\infty} \phi(x_n) = \phi(\alpha), \tag{6.1.14}$$

i.e., the limiting value α is a root of the equation $x = \phi(x)$. We call α a **fixed point** of the mapping $x \to \phi(x)$ and the iteration (6.1.13) a **fixed point iteration**.

An iterative method for solving an equation $f(x) = 0$ can be constructed by rewriting it in the equivalent form $x = \phi(x)$, which then defines a fixed point iteration (6.1.13). Clearly this can be done in many ways. For example, let $g(x)$ be any function such that $g(\alpha) \neq 0$ and set

$$\phi(x) = x - f(x)g(x). \tag{6.1.15}$$

Then α is a solution to $f(x) = 0$ if, and only if, α is a fixed point of ϕ

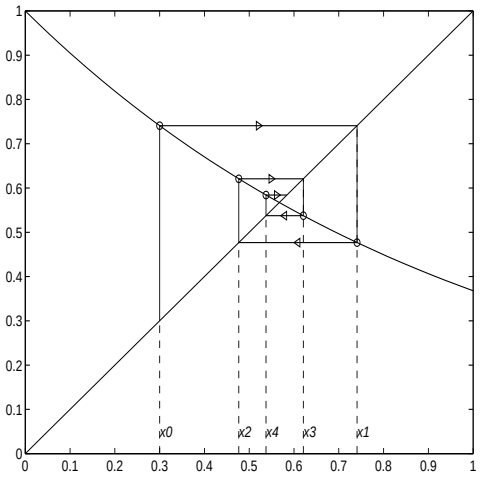


Figure 6.1.4. *The fixed point iteration $x_{k+1} = e^{-x_k}$, $x_0 = 0.3$.*

Example 6.1.6.

The equation $x + \ln x = 0$ can, for example, be written as:

(i) $x = -\ln x$; (ii) $x = e^{-x}$; (iii) $x = (x + e^{-x})/2$.

Each of these give rise to a different fixed point iteration. Results from the first eight iterations

$$x_{n+1} = e^{-x_n}, \quad x_0 = 0.3,$$

are pictured in Figure 6.2.1. The convergence is slow and we get $x_9 = 0.5641$ (correct value 0.567143).

As was shown already in Section 1.2, the iteration (6.1.13) may not converge even if the initial value x_0 is chosen arbitrarily close to a root. If $\lim_{n\to\infty} x_n = \alpha$ for all x_0 in a sufficiently close neighborhood of α the α is called a **point of attraction** otherwise α is a **point of repulsion**.

We shall see that under certain conditions the fixed-point problem has a unique solution and that the iteration defined by (6.1.13) converges to this solution. A sufficient condition for (6.1.13) to generate a convergent sequence is given in the following theorem.

Theorem 6.1.5.

Suppose that the function $\phi(x)$ has a real fixed point α, and that in the closed interval

$$J = \{x \mid |x - \alpha| \le \rho\}$$

$x \to \phi(x)$ is a **contraction mapping**, *i.e.,*

$$|\phi(s) - \phi(t)| \le C|s - t|, \quad 0 \le C < 1, \tag{6.1.16}$$

for arbitrary points s and t in J. Then for all $x_0 \in J$ the fixed-point iteration $x_n = \phi(x_{n-1})$, generates a sequence $\{x_n\}$ such that:

(a) $x_n \in J$, $n = 1, 2, \ldots$;

(b) $\lim_{n\to\infty} x_n = \alpha$,

(c) *α is the only root in J of $x = \phi(x)$.*

Proof. We first prove assertion (a), by induction. Suppose that $x_{n-1} \in J$. Then by (6.1.16) it follows that

$$|x_n - \alpha| = |\phi(x_{n-1}) - \phi(\alpha)| \le C|x_{n-1} - \alpha| \le C\rho.$$

Hence $x_n \in J$ and (a) is proved. Repeated use of the inequality above gives

$$|x_n - \alpha| \le C|x_{n-1} - \alpha| \le \cdots \le C^n|x_0 - \alpha|,$$

and since $C < 1$, the result (b) follows. Suppose, finally, that $x = \phi(x)$ has another root $\beta \in J$, $\beta \ne \alpha$. Then, by (6.1.16)

$$|\alpha - \beta| = |\phi(\alpha) - \phi(\beta)| < |\alpha - \beta|,$$

a contradiction; thus (c) follows. □

Observe that the contractive Lipschitz condition (6.1.16) implies the continuity of ϕ. If $\phi'(x)$ exists, then a sufficient condition for (6.1.16) to hold is that

$$|\phi'(x)| \le C < 1, \quad \forall\, x \in J, \tag{6.1.17}$$

since then by the mean value theorem we have for $x, y \in J$ that

$$|\phi(x) - \phi(y)| = |\phi'(\zeta_n)||x - y| < |x - y|, \quad \zeta_n \in J.$$

On the other hand if $|\phi'(\alpha)| > 1$ then the iterative method (6.1.13) diverges. The four different cases that occur, depending on the sign and magnitude of $\phi'(\alpha)$ were illustrated in Figures 1.2.1a–d.

There is an analogue result valid for functions $\phi(z)$ of a complex variable z. Assume that $\phi(z)$ is defined and analytic in the circle

$$K = \{z \mid |z - \alpha| \leq \rho\},$$

where α is a fixed point. Then if $|\phi'(z)| \leq C < 1$, for all $z \in K$, the fixed point iteration

$$z_0 \in K, \qquad z_n = \phi(z_{n-1}), \quad n = 1, 2, \ldots$$

converges to α, which is the only fixed point in K.

In Theorem 6.1.5 *we assumed the existence of a fixed point* α of $\phi(x)$. It is remarkable that the theorem can be modified so that it can be used to *prove the existence of a fixed point*, and hence of a root of the equation $x = \phi(x)$.

Theorem 6.1.6.

Let x_0 be a starting point, and consider the fixed point iteration $x_{n+1} = \phi(x_n)$, $n = 1, 2, \ldots$. Assume that J is a closed interval such that $x_0 \in J$ and

$$|\phi(s) - \phi(t)| \leq C|s - t|, \quad 0 \leq C < 1, \tag{6.1.18}$$

for all $s, t \in J$. Then if

$$x_1 + \frac{C}{1 - C}(x_1 - x_0) \in J, \tag{6.1.19}$$

(a), (b) *and* (c) *of Theorem* 6.1.5 *are true.*

Proof. The theorem will be proved in a more general setting in Vol. II, Chapter 11 (see Theorem 11.2.1). □

We remark that (6.1.18) is satisfied if $|\phi(x)| \leq C < 1$ in J. Further, there is an analogue of this theorem also for complex functions $\phi(z)$ analytic in a circle $K = \{z \mid |z - \alpha| \leq \rho\}$ containing the initial approximation z_0.

An estimate of the error in x_n, which depends only on x_0, x_1 and the Lipschitz constant m, may be derived as follows. For arbitrary positive integers m and n we have

$$x_{m+n} - x_n = (x_{m+n} - x_{m+n-1}) + \cdots (x_{n+2} - x_{n+1}) + (x_{n+1} - x_n).$$

From the Lipschitz condition we conclude that $|x_{i+1} - x_i| \leq C^i|x_1 - x_0|$, and hence

$$|x_{m+n} - x_n| \leq (C^{m-1} + \cdots + C + 1)|x_{n+1} - x_n|.$$

Summing the geometric series and letting $m \to \infty$ we obtain

$$|\alpha - x_n| \leq \frac{1}{1-C}|x_{n+1} - x_n| \leq \frac{C^n}{1-C}|x_1 - x_0|. \tag{6.1.20}$$

Note that if C is close to unity, then the error in x_n can be much larger than $|x_{n+1} - x_n|$. Clearly it is not always safe to terminate the iterations when $|x_{n+1} - x_n|$ is less than the required tolerance!

Example 6.1.7.

For a linearly convergent fixed point iteration the sequence $\{x_j - \alpha\}$ approximately forms a geometric series. Then, as seen in Sec. 3.3.2, a more rapidly convergent sequence $\{x'_j\}$ can be obtained by Aitken extrapolation,

$$x'_j = x_j - (\nabla x_j)^2 / \nabla^2 x_j. \tag{6.1.21}$$

Note that if the convergence is *not* linear, then the sequence $\{x'_n\}$ will usually converge *slower* than $\{x_n\}$!

The equation $x = e^{-x}$ has one root $\alpha \approx 0.567$. Using the fixed point iteration $x_{n+1} = e^{-x_n}$ combined with Aitken extrapolation we obtain the result shown in the table below.

j	x_j	∇x_j	$\nabla^2 x_j$	x'_j
0	**0.567**00 00000			
1	**0.567**22 45624	2245624		
2	**0.567**09 71994	-1273630	-3519254	**0.56714 329**25
3	**0.5671**6 94312	722318	1995948	**0.56714 329**11

It is seen that in this example the extrapolated sequence $\{x'_j\}$ converges much more rapidly, and nine correct decimals are obtained.

In the above example Aitken extrapolation was used in a *passive* way to transform the sequence $\{x_n\}$ into $\{x'_j\}$. It is also possible to use Aitken extrapolation in an *active way* (cf. Example 3.3.6). We start as before by computing $x_1 = \phi(x_0)$, $x_2 = \phi(x_1)$ and apply the formula (6.3.22) to compute x'_2. Next we continue the iterations from x'_2, i.e., compute $x_3 = \phi(x'_2)$, $x_4 = \phi(x_3)$. We can now extrapolate from x'_2, x_3 and x_4 to get x'_4, etc. It is easily verified that the sequence $z_n = x'_{2n}$ is generated by the fixed-point iteration

$$z_{n+1} = \psi(z_n), \quad \psi(z) = z - \frac{(\phi(z) - z)^2}{(\phi(\phi(z)) - \phi(z)) - (\phi(z) - z)}.$$

This iteration may converge even when the basic iteration $x_{n+1} = \phi(x_n)$ diverges!

6.1.5 Convergence Order and Efficiency

In general we will be given an equation $f(x) = 0$ to solve and want to construct a fixed point iteration such converges rapidly. Basic concepts to quantify the rate of convergence will now be introduced.

Definition 6.1.7.

Consider a sequence $\{x_n\}_0^\infty$ *with* $\lim_{n\to\infty} x_n = \alpha$, *and* $x_n \neq \alpha$ *for* $n < \infty$. *The sequence is said to have* **convergence order** *equal to* $q \geq 1$ *if for some constant* $0 < C < \infty$ *it holds that*

$$\lim_{n\to\infty} \frac{|x_{n+1} - \alpha|}{|x_n - \alpha|^q} = C. \tag{6.1.22}$$

Here C *is called the* **asymptotic error constant**.

If $q = 1$ *then we require that* $C < 1$ *and then* $\{x_n\}$ *is said to converge* **linearly** *and* C *is the rate of linear convergence. For* $q = 2, 3$ *the convergence is called* quadratic, *and* cubic, *but* q *need not be an integer.*

More precisely, the order q in Theorem 6.1.7 is called the Q-order of convergence, where Q stands for quotient. The same definitions can be used also for vector-valued sequences. Then absolute values in (6.1.22) are replaced by a vector norm.

There are types of convergence that are not covered by the above definition of order. A sequence may converge more slowly than linear so that (6.1.22) holds with $q = 1$ and $C = 1$. Then convergence is called **sublinear**. If (6.1.22) holds with $q = 1$ and $C = 0$, but not for any value of $q > 1$ then convergence is called **superlinear**.

Example 6.1.8.

Examples of sublinear, linear and superlinear convergence are

$$x_n = 1/n, \quad x_n = 2^{-n}, \quad \text{and} \quad x_n = n^{-n},$$

respectively.

Alternative definitions of convergence order are considered by Ortega and Rheinboldt [18, Chap. 9] and Brent [2, Sec. 3.2]. For example, if

$$\lim_{n\to\infty} \inf(-\log|x_n - \alpha|)^{1/n} = q, \tag{6.1.23}$$

then q is called weak order of convergence for x_n, since (6.1.22) implies (6.1.23), but not vice versa. For example, the sequence $x_n = \exp(-p^n)(2 + (-1)^n)$ converges to 0 with weak order p. However, the limit in (6.1.22) does not exist if $q = p$, is zero if $q < p$ and infinite if $q > p$.

Consider a fixed point iteration $x_{n+1} = \phi(x_n)$. Assume that $\phi'(x)$ exists and is continuous in a neighborhood α. It then follows from the proof of Theorem 6.1.5, that if $0 < |\phi'(\alpha)| < 1$ and x_0 is chosen sufficiently close to α, then the sequence x_n generated by $x_{n+1} = \phi(x_n)$ satisfies (6.1.22) with $q = 1$ and $C = |\phi'(\alpha)|$.

The number of accurate decimal places in the approximation x_n equals $\delta_n = -\log_{10}|x_n - \alpha|$. Equation (6.1.22) implies that

$$\delta_{n+1} \approx q\delta_n - \log_{10}|C|.$$

Hence for linear convergence ($q = 1$) as $n \to \infty$ each iteration gives a fixed (fractional) number of additional decimal places. For a method with convergence of

order $q > 1$ each iteration increases the number of correct decimal places q-fold as $n \to \infty$. This shows that eventually a method with larger order of convergence will converge faster.

Example 6.1.9.

Consider a sequence x_n with quadratic convergence with $C = 1$. Set $\epsilon_n = |x_n - \alpha|$ and assume that $\epsilon_0 = 0.9$. From $\epsilon_{n+1} \leq C\epsilon_n^2$, it follows that ϵ_n, for $n = 2, 3, \ldots$, is bounded by

$$0.81, 0.66, 0.43, 0.19, 0.034, 0.0012, 1.4 \cdot 10^{-6}, 1.9 \cdot 10^{-12}, \ldots,$$

respectively. For $n \geq 6$ the number of significant digits is approximately doubled at each iteration!

Consider an iteration method with convergence order $q \geq 1$. If each iteration requires m units of work (usually the work involved in computing a function value or a value of one of its derivatives) then the **efficiency index** of the iteration is defined as

$$E = q^{1/m}. \tag{6.1.24}$$

The efficiency index gives a basis for comparing the efficiency of iterative methods of different order of superlinear convergence. Assuming that the cost of evaluating $f(x_n)$ and $f'(x_n)$ is two units the efficiency index for Newton's method is $E = 2^{1/2} = \sqrt{2}$. (Methods that converge linearly all have $E = 1$.)

The order of the fixed-point iteration $x_{n+1} = \phi(x_n)$ can be determined if $\phi(x)$ is sufficiently many times continuously differentiable in a neighborhood of α.

Theorem 6.1.8. *Assume that $\phi(x)$ is p times continuously differentiable. Then the iteration method $x_{n+1} = \phi(x_n)$ is of order p for the root α if and only if*

$$\phi^{(j)}(\alpha) = 0, \quad j = 1 : p-1, \quad \phi^{(p)}(\alpha) \neq 0. \tag{6.1.25}$$

Proof. If equation (6.1.25) holds, then according to Taylor's theorem we have

$$x_{n+1} = \phi(x_n) = \alpha + \frac{1}{p!}\phi^{(p)}(\zeta_n)(x_n - \alpha)^p, \quad \zeta_n \in \operatorname{int}(x_n, \alpha).$$

Hence for a convergent sequence x_n the error $\epsilon_n = x_n - \alpha$ satisfies

$$\lim_{n\to\infty} |\epsilon_{n+1}|/|\epsilon_n|^p = |\phi^{(p)}(\alpha)|/p! \neq 0,$$

and the order of convergence equals p. It also follows that if $\phi^{(j)}(\alpha) \neq 0$ for some j, $1 \leq j < p$, or if $\phi^{(p)}(\alpha) = 0$, then the iteration cannot be of order p. ∎

Example 6.1.10.

We remarked before that to compute a root α of $f(x) = 0$, we can use a fixed point iteration with $\phi(x) = x - f(x)g(x)$, where $g(x)$ is an arbitrary function such that $g(\alpha) \neq 0$. we evaluate the derivative

$$\phi'(x) = 1 - f'(x)g(x) - f(x)g'(x).$$

To achieve quadratic convergence we take $g(x) = 1/f'(x)$. Assuming that $f'(\alpha =\neq 0$ we find, using $f(\alpha) = 0$, that $\phi'(\alpha) = 1 - f'(\alpha)g(\alpha) = 0$, Hence the iteration

$$x_{n+1} = x_n - f(x_n)/f'(x_n), \tag{6.1.26}$$

achieves quadratic convergence. This is Newton's method, which will be treated at length in Sec. 6.3

Review Questions

1. What does limit the final accuracy of a root computed by the bisection algorithm? Discuss suitable termination criteria.
2. (a) Given a nonlinear scalar equation $f(x) = 0$ with a simple root α. How can a fixed point iteration $x_{n+1} = \phi(x_n)$ be constructed, which converges to α?

   (b) Assuming that a fixed point α exists for the mapping $x = \phi(x)$. Give sufficient conditions for convergence of the sequence generated by $x_{n+1} = \phi(x_n)$.

   (c) How can the conditions in (b) be modified so that the *existence* of a fixed point can be proved?
3. (a) Define the concepts order of convergence and asymptotic error constant for a convergent sequence $\{x_n\}$ with $\lim_{n\to\infty} x_n = \alpha$.

   (b) What is meant by sublinear and superlinear convergence? Give examples of sequences with sublinear and superlinear convergence.
4. (a) Define the efficiency index of a given iterative method of order p and asymptotic error constant $C \neq 0$.

   (b) Determine the order of a new iterative method consisting of m consecutive steps of the method in (a). What is the order and error constant of this new method? Show that it has the same efficiency index as the first method.
5. (a) When can (passive) Aitken extrapolation be applied to speed up the convergence of sequence.

   (b) Describe the difference between active and passive Aitken extrapolation?
6. What two quantities determines the attainable accuracy of a simple root α to the equation $f(x) = 0$. Give an example of an ill-conditioned root.
7. Discuss the choice of termination criteria for iterative methods.

Problems and Computer Exercises

1. Use graphic representation to determine the zeros of the following functions to one correct decimal:

(a) $4\sin x + 1 - x$; (b) $1 - x - e^{-2x}$; (c) $(x+1)e^{x-1} - 1$;

(d) $x^4 - 4x^3 + 2x^2 - 8$; (e) $e^x + x^2 + x$; (f) $e^x - x^2 - 2x - 2$;

(g) $3x^2 + \tan x$.

2. Show analytically that the equation $xe^{-x} = \gamma$ has exactly two real roots when $\gamma < e^{-1}$.

3. Plot the functions $f(x) = \cosh x$ and $g(x) = 1/\cos x$ and deduce that the equation $\cosh x \cos x = 1$ has its smallest positive root in the interval $(3\pi/2, 2\pi)$. Determine this root using the bisection method.

4. The following equations all have a root in the interval $(0, 1.6)$ Determine these with an error less than 10^{-8} using the bisection method.

(a) $x\cos x = \ln x$; (b) $2x = e^{-x}$; (c) $e^{-2x} = 1 - x$.

5. Locate the real root of the equation

$$e^x(x-1) = e^{-x}(x+1),$$

by graphing both sides. Then compute the root with an error less than 10^{-8} using bisection. How many bisection steps are needed?

6. Let k be a given non-negative number and consider the equation $\sin x = -k\cos x$. This equation has infinitely many roots. Separate the roots, i.e., partition the real axis into intervals which contain exactly one root.

7. The choice of m_k as the *arithmetic* mean of a_{k-1} and b_{k-1} in the bisection method minimizes the worst case maximum *absolute* error. If in the case that $ab > 0$ we take instead

$$m_k = \sqrt{a_k b_k}$$

i.e., the *geometric* mean, then the worst case *relative* error is minimized. Do Example 6.1.2 using this variation of the bisection method.

8. In Example 6.1.6 three different fixed point iterations were suggested for solving the equation $x + \ln x = 0$. (a) Which of the formulas *can* be used?

(b) Which of the formulas *should* be used?

(c) Give an even better formula!

9. Investigate if and to what limit the iteration $x_{n+1} = 2^{x_n - 1}$ sequence converges for various choices of x_0.

10. (L. Wittmeyer-Koch) (a) A fixed point iteration $x_{n+1} = \phi(x_n)$ can converge also when $|\phi'(\alpha)| = 1$. Verify this by graphing the iteration for $\phi(x) = x + (x-1)^2$, $x_0 = 0.6$. which has the fixed point $\alpha = 1$.

(b) Show that for the iteration in (a) if $x_n = 1 - \epsilon$, then

$$\frac{|x_{n+1} - 1|}{|x_n - 1|} = 1 - \epsilon,$$

i.e. the asymptotic rate of convergence is sublinear.

11. (a) Consider the fixed point iteration $x_{n+1} = \phi(x_n)$, where $\phi(x) = x+(x-1)^2$. Show that this has a fixed point for $\alpha = 1$ and that $\phi'(\alpha) = 1$.

(b) Show that the iteration in (a) is convergent for $x_0 < 1$.

12. Consider the iteration $x_{n+1} = 1-\lambda x_n^2$. Illustrate graphically how the iteration for $\lambda = 0.7, 0.9, 2$. (For $\lambda = 2$ the iteration is chaotic.)

13. Use active Aitken extrapolation on the (divergent) iterative method $x_{n+1} = 5\ln x_n$ to compute the smallest root of the equation $x = 5\ln x$. Start with $x_0 = 1.3$.

6.2 Methods Based on Interpolation

6.2.1 The Method of False Position

Assume that we have two initial approximations $a_0 = a$ and $b_0 = b$ such that $f(a)f(b) < 0$. As in the bisection method we generate a nested sequence of intervals $(a_0, b_0) \supset (a_1, b_1) \supset (a_2, b_2) \subset \cdots$ such that $f(a_n)f(b_n) < 0$, $n = 0, 1, 2, \ldots$. Given (a_n, b_n), we take x_{n+1} to be the intersection of the secant through the point $(a_n, f(a_n))$ and $(b_n, f(b_n))$. Then by Newton's interpolation formula x_{n+1} satisfies

$$0 = f(a_n) + (x_{n+1} - a_n)\frac{f(a_n) - f(b_n)}{a_n - b_n}$$

giving

$$x_{n+1} = a_n - f(a_n)\frac{a_n - b_n}{f(a_n) - f(b_n)}. \tag{6.2.1}$$

If $f(x_{n+1})f(a_n) > 0$, set $a_{n+1} = x_{n+1}$ and $b_{n+1} = b_n$; otherwise set $b_{n+1} = x_{n+1}$ and $a_{n+1} = a_n$. This is the **false-position method** or in Latin **regula falsi**.[2] Note that if $f(x)$ is linear we obtain the root in just one step, but sometimes the rate of convergence can be much slower than for bisection.

Suppose now that $f(x)$ is convex on $[a, b]$, $f(a) < 0$, and $f(b) > 0$, as in Figure 6.2.1. Then the secant through $x = a$ and $x = b$ will lie above the curve and hence intersect the x-axis to the left of α. The same is true for all subsequent secants and therefore the right endpoint b will be kept. The approximations $x_1, x_2, x_3, \ldots$ will all lie on the convex side of the curve and cannot go beyond the root α. A similar behavior, with monotone convergence and one of the points a or b fixed, will occur whenever $f''(x)$ exists and has constant sign on $[a, b]$.

Example 6.2.1.

We apply the method of false position to the $f(x) = (x/2)^2 - \sin x = 0$ from Example 6.1.2 with initial approximations $a_0 = 1.5$, $b_1 = 2$. We have $f(1.5) = -0.434995 < 0$ and $f(2.0) = +0.090703 > 0$ and successive iterates are

[2] The method of regula falsi is very old, originating in 5th century Indian texts, and was used in medieval Arabic mathematics. It got its name from the Italian mathematician Leonardi Pisano in the 13th century.

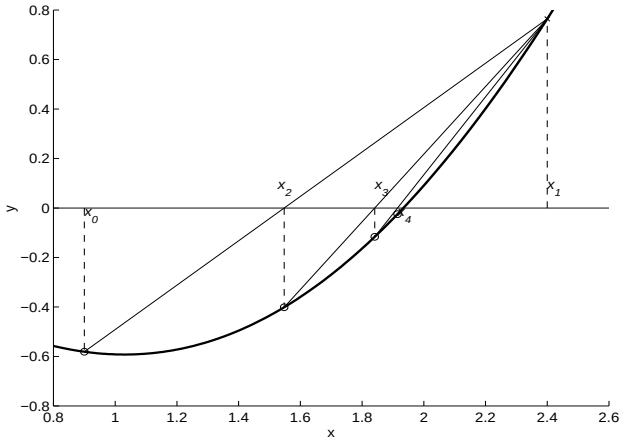


Figure 6.2.1. *The false-position method.*

n	x_n	$f(x_n)$	h_n
1	**1.9**13731 221035	−0.026180060742	−0.019322989205
2	**1.933**054 210240	−0.000924399645	−0.000675397892
3	**1.93372**9 608132	−0.000031930094	−0.000023321005
4	**1.93375**2 929137	−0.000001102069	−0.000000804916
5	**1.933753 7**34053		

Note that $f(x_n) < 0$ for all $n \geq 0$ and consequently $b_n = 2$ is fixed. In the limit convergence is linear with rate approximately equal to $C \approx 0.034$.

If f is twice continuously differentiable and $f''(\alpha) \neq 0$, then eventually an interval will be reached on which $f''(x)$ does not change sign. Then, as in the example above, one of the endpoints (say b) will be retained and $a_n = x_n$ in all future steps. By (6.2.1) the successive iterations are

$$x_{n+1} = x_n - f(x_n)\frac{x_n - b}{f(x_n) - f(b)}.$$

To determine the speed of convergence subtract α and divide by $\epsilon_n = x_n - \alpha$ to get

$$\frac{\epsilon_{n+1}}{\epsilon_n} = 1 - \frac{f(x_n)}{x_n - \alpha}\frac{x_n - b}{f(x_n) - f(b)}.$$

Since $\lim_{n\to\infty} x_n = \alpha$ and $f(\alpha) = 0$, it follows that

$$\lim_{n\to\infty} \frac{\epsilon_{n+1}}{\epsilon_n} = C = 1 - (b - \alpha)\frac{f'(\alpha)}{f(b)}, \tag{6.2.2}$$

which shows that convergence is linear. Convergence will be very slow if $f(x)$ is very flat near the root α, $f(b)$ is large, and α near b since then $(b-\alpha)f'(\alpha) \ll f(b)$ and $C \approx 1$.

6.2.2 The Secant Method

A serious drawback with the method of false position is that ultimately one endpoint of the sequence of bracketing intervals will become fixed and therefore the length $(b_n - a_n)$ will not tend to zero. This can be avoided and convergence substantially improved by always using the secant through *the last two points* $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$. The next approximation x_{n+1} is determined as the abscissa of the point of intersection between this secant and the x-axis; see Figure 6.2.2.

Given initial approximations $x_{-1} = a$ and $x_0 = b$, approximations $x_1, x_2, , x_3, \ldots$ are computed by

$$x_{n+1} = x_n + h_n, \quad h_n = -f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}, \quad n \geq 1, \tag{6.2.3}$$

assuming that $f(x_n) \neq f(x_{n-1})$. This is the **secant method**, which historically predates Newton's method.

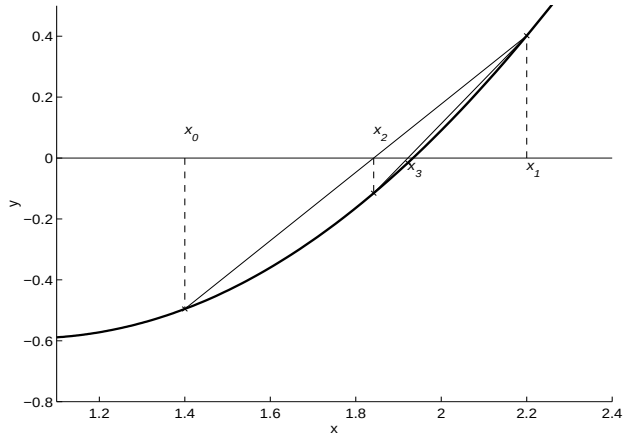


Figure 6.2.2. *The secant method.*

Notice that although regula falsi and the secant method require two initial approximations to the root, only *one function evaluation per step* is needed. The iteration, which is of the form $x_{n+1} = \phi(x_n; x_{n-1})$, is not a fixed point iteration as defined in Section 6.1.4. Sometimes methods of this form, which use old information at x_{n_1}, are called fixed point iterations with memory.

When the secant method converges $|x_n - x_{n-1}|$ will eventually become small. The quotient $(x_n - x_{n-1})/(f(x_n) - f(x_{n-1}))$ will then be determined with poor relative accuracy. If x_n and x_{n-1} both are very close to the root α and not bracketing α, then the resulting rounding error in x_{n+1} can then become very large. Fortunately, from the error analysis below it follows that the approximations in general are such that $|x_n - x_{n-1}| \gg |x_n - \alpha|$ and the dominant contribution to the round-off error in x_{n+1} comes from the error in $f(x_n)$. Note that (6.2.3) should *not* be rewritten

in the form

$$x_{n+1} = \frac{x_{n-1}f(x_n) - x_n f(x_{n-1})}{f(x_n) - f(x_{n-1})},$$

since this formula can give rise to severe difficulties with *cancellation* when $x_n \approx x_{n-1}$ and $f(x_n)f(x_{n-1}) > 0$. Even (6.2.3) is not always safe to use. We must take care to avoid overflow or division by zero. Without restriction we can assume that $|f(x_{n-1})| \geq |f(x_n)| > 0$ (otherwise renumber the two points). Then, s_n can be computed without risk of overflow from

$$x_{n+1} = x_n + \frac{s_n}{1 - s_n}(x_n - x_{n-1}), \quad s_n = \frac{f(x_n)}{f(x_{n-1})}. \tag{6.2.4}$$

where the division with $1 - s_n$ is only carried out if $1 - s_n$ is large enough.

Example 6.2.2.

To illustrate the improved convergence of the secant method we consider once again the equation $f(x) = (x/2)^2 - \sin x = 0$ with initial approximations $x_0 = 1.5$, $x_1 = 2$. The result is:

n	x_n	$f(x_n)$	h_n
-1	1.5	−0.434994 986604	
0	2.0	+0.090702 573174	−0.086268 778965
1	**1.9**13731 221035	−0.026180 060742	+0.019322 989205
2	**1.933**054 210240	−0.000924 399645	+0.000707 253882
3	**1.9337**61 464122	+0.000010 180519	−0.000007 704220
4	**1.933753 75**9902	−0.000000 003867	+0.000000 002925
5	**1.933753 762827**		

Note that the approximations x_1 and x_2 are the same as for the false position. method, but here x_4 is correct to eight decimals and x_5 to twelve decimals. The rapid convergence is partly because $x_1 = 2$ is quite a good initial approximation. However, note that although the root is bracketed by the initial intervals $[x_0, x_1]$ and $[x_1, x_2]$ it lies outside the interval $[x_2, x_3]$.

Assume that f is twice continuously differentiable. Then according to Newton's interpolation formula with error term (Theorem 4.3.1) we have

$$f(x) = f(x_n) + (x - x_n)[x_{n-1}, x_n]f + (x - x_{n-1})(x - x_n)\frac{f''(\zeta_n)}{2}, \tag{6.2.5}$$

where $\zeta_n \in \text{int}(x, x_{n-1}, x_n)$

$$f[x_{n-1}, x_n] = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

To derive an asymptotic formula for the secant method we put $x = \alpha$ in (6.2.5) and subtract the secant equation $0 = f(x_n) + (x_{n+1} - x_n)[x_{n-1}, x_n]f$. Since $f(\alpha) = 0$ we get

$$(\alpha - x_{n+1})[x_{n-1}, x_n]f + (\alpha - x_{n-1})(\alpha - x_n)f''(\zeta_n)/2 = 0,$$

where $\zeta_n \in \text{int}(\alpha, x_{n-1}, x_n)$. According to the mean-value theorem, we have

$$[x_{n-1}, x_n]f = f'(\zeta_n'), \quad \zeta_n' \in \text{int}(x_{n-1}, x_n),$$

and it follows that

$$\epsilon_{n+1} = \frac{1}{2}\frac{f''(\zeta_n)}{f'(\zeta_n')}\epsilon_n\epsilon_{n-1}. \tag{6.2.6}$$

Example 6.2.3.

The ratios $\epsilon_{n+1}/(\epsilon_n\epsilon_{n-1})$ in Example 6.2.2 are equal to 0.697, 0.527, 0.550, $n = 1:3$, which compares well with the limiting value 0.543 of $\frac{1}{2}f''(\alpha)/f'(\alpha)$.

From (6.2.6) it can be deduced that the secant method always converges from starting values x_0, x_1 sufficiently close to α. For this to be true it suffices that the first derivative $f'(x)$ is continuous, since then

$$\epsilon_{n+1} = \left(1 - \frac{f'(\xi_n)}{f'(\zeta_n)}\right)\epsilon_n, \quad \xi_n \in \text{int}(x_{n-1}, \alpha), \quad \zeta_n \in \text{int}(x_n, x_{n-1}).$$

However, *in the secant method there is no guarantee that the computed sequence of approximations stay in the initial interval* $[x_0, x_1]$. Unlike the steady convergence of the bisection method things can go seriously wrong using the secant method! A remedy will be discussed in Sec. 6.2.4.

The following theorem gives the order of convergence for the secant method.

Theorem 6.2.1. *Suppose that $f(x)$ is twice continuously differentiable and that in a neighborhood I of the root α, containing $x_0, x_1, x_2, \ldots, x_n$, we have*

$$\frac{1}{2}\left|\frac{f''(y)}{f'(x)}\right| \le M, \quad x, y \in I.$$

Let $q = (1+\sqrt{5})/2 = 1.618\ldots$ be the unique positive root of the equation $\mu^2 - \mu - 1 = 0$ and set

$$K = \max\left(M|\epsilon_0|, (M|\epsilon_1|)^{1/q}\right), \quad n = 0, 1, 2, \ldots. \tag{6.2.7}$$

Then it holds that

$$|\epsilon_n| \le \frac{1}{M}K^{q^n}, \tag{6.2.8}$$

i.e., the iteration has convergence order q.

Proof. The proof is by induction. From the choice of K it follows that (6.2.8) is trivially true for $n = 0, 1$. Suppose that (6.2.8) holds for n and $n-1$. Then since $q^2 = q + 1$ it follows using the assumption and (6.2.6) that

$$|\epsilon_{n+1}| \le M|\epsilon_n||\epsilon_{n-1}| \le \frac{1}{M}K^{q^n}K^{q^{n-1}} = \frac{1}{M}K^{q^n+q^{n-1}} = \frac{1}{M}K^{q^{n+1}}. \tag{6.2.9}$$

□

To compare the efficiency of the secant method and Newton's method, which is quadratically convergent, we use the efficiency index introduced in Section 6.1.5. Assume that the work to compute $f'(x)$ is θ times the amount of work required to compute $f(x)$. Then, with the same amount of work we can perform $k(1+\theta)$ iterations with the secant method and k iterations with Newton's method. Equating the errors we get $(m\epsilon_0)^{2^k} = (m\epsilon_0)^{p^{k(1+\theta)}}$, where $q = 1.618\ldots$. Hence the errors are the same for both methods when $p^{k(1+\theta)} = 2^k$ or

$$(1+\theta)\log\left(\tfrac{1}{2}(1+\sqrt{5})\right) = \log 2,$$

which gives $\theta = 0.4404\ldots$. Thus, from this analysis we conclude that if $\theta > 0.44$, then the secant method is asymptotically more efficient than Newton's method.

In Example 6.2.2 we can observe that the error $\epsilon_n = x_n - \alpha_n$ changes sign at every third step. Hence, in this example,

$$\alpha \in \operatorname{int}(x_{n+1} - x_n), \quad n = 0, 1, 3, 4, \ldots.$$

That is, *the root α is bracketed by x_n and x_{n+1} except for every third step.* We shall show that this is no coincidence. Assume that $x_n \in (a, b)$, $n = 0, 1, 2, \ldots$, and that $f'(x) \neq 0$ and $f''(x)$ does not change sign in (a, b). Then from (6.2.6) it follows that the ration

$$\frac{\epsilon_{n+1}}{\epsilon_n \epsilon_{n-1}}$$

will have constant sign for all n. Then if $\alpha \in \operatorname{int}(x_0, x_1)$ and $\epsilon_0\epsilon_1 < 0$, and it follows that the sign of ϵ_n must change every third step according to one of the following two schemes (verify this!):

$$\cdots + - + + - + + - + + \cdots$$
$$\cdots + - - + - - + - - + \cdots$$

Hence convergence for the secant method, if it occurs, will take place in a waltz rhythm! This means that at every third step the last two iterates x_{n-1} and x_n will not always bracket the root.

6.2.3 Higher Order Interpolating Methods

In the secant method linear interpolation through (x_{n-1}, f_{n-1}) and (x_n, f_n) is used to determine the next approximation to the root. A natural generalization is to use an interpolating method of higher order. Let $x_{n-r}, \ldots, x_{n-1}, x_n$ be $r+1$ distinct approximations and determine the (unique) polynomial $p(x)$ of degree r interpolating $(x_{n-j}, f(x_{n-j}))$, $j = 0 : r$. By Newton's interpolation formula (Sec. 4.2.1) the interpolating polynomial is

$$p(x) = f_n + [x_n, x_{n-1}]f \cdot (x - x_n) + \sum_{j=2}^{r} [x_n, x_{n-1}, \ldots, x_{n-j}]f\, \Phi_j(x),$$

where

$$\Phi_j(x) = (x - x_n)(x - x_{n-1}) \cdots (x - x_{n-j}).$$

The next approximation x_{n+1} is taken as the real root to the equation $p(x) = 0$ closest to x_n and x_{n-r} is deleted. Suppose the interpolation points lie in an interval J, which contains the root α and in which $f'(x) \neq 0$. It can be shown that if there is at least one interpolation point on each side of α then $p(x) = 0$ has a real root in J. Further the following formula for the error holds (Traub [25, pp. 67–75])

$$\epsilon_{n+1} = -\frac{f^{(r+1)}(\zeta_n)}{(r+1)!p'(\eta_n)} \prod_{i=0}^{r} \epsilon_{n-i}, \tag{6.2.10}$$

where $\zeta_n \in \text{int}\,(\alpha, x_{n-1}, x_n)$ and $\eta_n \in \text{int}\,(\alpha, x_{n+1})$.

In the special case $r = 2$ we get the quadratic equation

$$p(x) = f_n + (x - x_n)[x_n, x_{n-1}]f + (x - x_n)(x - x_{n-1})[x_n, x_{n-1}, x_{n-2}]f. \tag{6.2.11}$$

We assume that $[x_n, x_{n-1}, x_{n-2}]f \neq 0$ since otherwise the method degenerates into the secant method. Setting $h_n = (x - x_n)$ and writing $(x - x_{n-1}) = h_n + (x_n - x_{n-1})$, this equation becomes

$$h_n^2[x_n, x_{n-1}, x_{n-2}]f + \omega h_n + f_n = 0, \tag{6.2.12}$$

where

$$\omega = [x_n, x_{n-1}]f + (x_n - x_{n-1})[x_n, x_{n-1}, x_{n-2}]f. \tag{6.2.13}$$

The root closest to x_n corresponds to the root h_n of smallest absolute value to the equation (6.2.12). To express this root in a numerically stable way the standard formula for the roots of a quadratic equation should be multiplied by its conjugate quantity (see Example 2.3.3). Using this formula we get

$$x_{n+1} = x_n + h_n, \quad h_n = -\frac{2f_n}{\omega \pm \sqrt{\omega^2 - 4f_n\,[x_n, x_{n-1}, x_{n-2}]f}}, \tag{6.2.14}$$

where the sign in the denominator should be chosen so as to minimize $|h_n|$. This is the **Muller–Traub method**.

A drawback is that the equation (6.2.12) may not have a real root even if a real zero is being sought. On the other hand, this means that the Muller–Traub method has the useful property that complex roots may be found from real starting approximations.

By (6.2.10) it follows that

$$\epsilon_{n+1} = -\frac{f'''(\zeta_n)}{3!\,p'(\eta_n)} \epsilon_n \epsilon_{n-1} \epsilon_n. \tag{6.2.15}$$

It can be shown that the convergence order for the Muller–Traub method is at least $q = 1.839\ldots$, which equals the largest root of the equation $\mu^3 - \mu^2 - \mu - 1 = 0$ (cf.Theorem 6.2.1). Hence this method does not quite achieve quadratic convergence. In fact, it can be shown under very weak restrictions that *no iterative method using only one function evaluation can have* $q \geq 2$.

For $r > 2$ there are no useful explicit formulas for determining the zeros of the interpolating polynomial $p(x)$. Then we can proceed as follows. We write the equation $p(x) = 0$ in the form $x = x_n + F(x)$, where

$$F(x) \equiv \frac{-f_n - \sum_{j=2}^{r} [x_n, x_{n-1}, \dots x_{n-j}]f \, \Phi_j(x)}{[x_n, x_{n-1}]f}$$

(cf. Sec. 4.2.4). Then a fixed point *iteration* can be used to solve for x. To get the first guess x^0 we ignore the sum (this means using the secant method) and then iterate, $x^i = x_n + F(x^{i-1})$, $i = 1, 2, \dots$ until x^i and x^{i-1} are close enough.

Suppose that $x_{n-j} - x_n = O(h)$, $j = 1 : r$, where h is some small parameter in the context (usually some step size). Then $\Phi_j(x) = O(h^j)$, $\Phi_j'(x) = O(h^{j-1})$. The divided differences are $O(1)$, and we assume that $[x_n, x_{n-1}]f$ is bounded away from zero. Then the terms of the sum decrease like h^j. The convergence ratio $F'(x)$ is here approximately

$$\frac{\Phi_2'(x)[x_n, x_{n-1}, x_{n-2}]f}{[x_n, x_{n-1}]f} = O(h).$$

So, if h is small enough, the iterations converge rapidly.

A different way to extend the secant method is to use **inverse interpolation**. Assume that $y_n, y_{n-1}, \dots, y_{n-r}$ are distinct and let $q(y)$ be the unique polynomial in y interpolating the values $x_n, x_{n-1}, \dots, x_{n-r}$. Reversing the rule of x and y and using Newton's interpolation formula this interpolating polynomial is

$$q(y) = x_n + [y_n, y_{n-1}]g \cdot (y - y_n) + \sum_{j=2}^{r} [y_n, y_{n-1}, \dots, y_{n-j}]f \, \Psi_j(y),$$

where $g(y_{n-j}) = x_{n-j}$, $j = 0 : r$.

$$\Psi_j(y) = (y - y_n)(y - y_{n-1}) \cdots (y - y_{n-j}).$$

The next approximation is then taken to be $x_{n+1} = q(0)$, that is

$$x_{n+1} = x_n - y_n \, [y_n, y_{n-1}]g + \sum_{j=2}^{r} [y_n, y_{n-1}, \dots, y_{n-j}]g \, \Psi_j(0),$$

For $r = 1$ there is no difference between direct and inverse interpolation and we recover the secant method. For $r > 1$ inverse interpolation as a rule gives different results. Inverse interpolation has the advantage of not requiring the solution of a polynomial equation. (For other ways of avoiding this see Problems 3 and 4.) The case $r = 2$ corresponds to inverse quadratic interpolation

$$x_{n+1} = x_n - y_n \, [y_n, y_{n-1}]g + y_n y_{n-1} \, [y_n, y_{n-1}, y_{n-2}]g, \tag{6.2.16}$$

This method has the same order of convergence as the Muller–Traub method.

Note that this method requires that y_n, y_{n-1}, and y_{n-2} are distinct. Even if this is the case it is not always safe to compute x_{n+1} from (6.2.16). Care has to be

taken in order to avoid overflow and possibly division by zero. If we assume that $0 \neq |y_n| \leq |y_{n-1}| \leq |y_{n-2}|$ then it is safe to compute

$$s_n = y_n/y_{n-1}, \quad s_{n-1} = y_{n-1}/y_{n-2}, \quad r_n = y_n/y_{n-2} = s_n s_{n-1}.$$

We can rewrite (6.2.16) in the form $x_{n+1} = x_n + p_n/q_n$, where

$$\begin{aligned} p_n &= s_n[(1-r_n)(x_n - x_{n-1}) - s_{n-1}(s_{n-1} - r_n)(x_n - x_{n-2})], \\ q_n &= (1-s_n)(1-s_{n-1})(1-r_n). \end{aligned}$$

The final division p_n/q_n is only carried out if the correction is sufficiently small.

6.2.4 A Robust Hybrid Method

Efficient and robust root finders can be constructed by combining the secant method (or some higher order interpolation method) with bisection, A simple combination of Newton method with bisection will be discussed in Section 6.3.2.

A particularly elegant combination of bisection and the secant method was developed in the 1960s by van Wijngaarden, Dekker and others at the Mathematical Center in Amsterdam. A related algorithm, called zeroin, which combines bisection, the secant method and inverse quadratic interpolation, was developed by Brent [2]. The Matlab function "fzero", which finds a zero near a given approximation x_0, is based on Zeroin. A discussion of a slightly simplified version of fzero is given in Moler [16, Ch. 4.7].

We now outline the basic ideas used in zeroin. Start with a and b such that $f(a)f(b) < 0$ and use a secant step to get c in (a, b). We the repeat the following steps until $|b - a| < tol$ or $f(b) = 0$:

- Arrange a, b, and c so that $f(a)$ and $f(b)$ have opposite sign, $|f(b)| \leq |f(a)|$, and c is the value of b in the previous step.
- If $c \neq a$ compute the step using inverse quadratic interpolation; otherwise compute a secant step.
- If the computed step gives an approximation in $[a, b]$ take it; otherwise take a bisection step.

Review Questions

1. Sketch a function $f(x)$ with a root in (a, b), such that regula falsi converges very slowly.

2. Outline how the secant method can be safeguarded by combining it with the bisection method.

3. What property should the function $f(x)$ have to be unimodal on the interval $[a, b]$?

4. Discuss root finding methods based on quadratic interpolation (Muller–Traub's method) and inverse quadratic interpolation. What are the merits of these two approaches?

Problems and Computer Exercises

1. Use the secant method to determine the roots of the following equations to six correct decimals

(a) $2x = e^{-x}$; (b) $\tan x + \cosh x = 0$.

2. Assume that we have $f_n f_{n-1} < 0$, and have computed x_{n+1}. If $f_{n+1}f_n < 0$ then in the next step we compute x_{n+2} by a secant step otherwise we use a line through (x_{n+1}, f_{n+1}) and $(x_{n-1}, \theta f_{n-1})$, where $0 < \theta < 1$. Clearly, $\theta = 1$ corresponds to a step with the method of false position and will usually give $f_{n+2}f_{n+1} > 0$. On the other hand, $\theta = 0$ gives $x_{n+1} = x_n$, and thus $f_{n+1}f_n < 0$. Hence a suitable choice of θ will always give $f_{n+2}f_{n+1} < 0$.
Show that with $\theta = 0.5$ in a modified step it holds asymptotically $\epsilon_{n+1} \approx -\epsilon_n$. Deduce that the resulting algorithm gives cubic convergence with three function evaluations and hence has efficiency index $E = 3^{1/3} = 1.4422\ldots$. [3]

3. Another modification of the secant method can be derived by estimating $f'(x_n)$ in Newton's method by quadratic interpolation through the points x_n, x_{n-1}, x_{n-2}. Show that the resulting method can be written $x_{n+1} = x_n - f(x_n)/\omega$, where

$$\omega = f[x_n, x_{n-1}] + (x_n - x_{n-1})f[x_n, x_{n-1}, x_{n-2}].$$

4. The Muller–Traubś method uses three points to determine the coefficient of an interpolating parabola. The same points can also be interpolated by a rational function of the form

$$g(x) = \frac{x - a}{bx + c}.$$

An iterative method is devised by taking x_{n+1} equal to the root a of $g(x) = 0$.
(a) Show that this is equivalent to calculating x_{n+1} from the "modified secant formula"

$$x_{n+1} = x_n - f_n \frac{x_n - x_{n-2}}{f_n - \tilde{f}_{n-2}}, \qquad \tilde{f}_{n-2} = f_{n-2}\frac{f[x_n, x_{n-1}]}{f[x_{n-1}, x_{n-2}]}.$$

Hint: Use a theorem in projective geometry, according to which the cross ratio of any four values of x is equal to the cross ratio of the corresponding values of $g(x)$ (see Householder [10, p. 159]). Hence

$$\frac{(0 - f_n)/(0 - f_{n-2})}{(y_{n-1} - f_n)/(y_{n-1} - f_{n-2})} = \frac{(x_{n+1} - x_n)/(x_{n+1} - x_{n-2})}{(x_{n-1} - x_n)/(x_{n-1} - x_{n-2})}.$$

[3] The resulting modified rule of false position is often called after its origin the **Illinois method**. It is due originally to the staff of the computer center at the University of Illinois in the early 1950's.

(b) Use the result in (a) to show that $x_{n-1} \in \text{int}(x_{n-2}, x_n)$ if

$$\text{sign}(y_n) = -\text{sign}(y_{n-2}), \qquad \text{sign}(y[x_n, x_{n-1}]) = \text{sign}(y[x_{n-1}, x_{n-2}]).$$

5. The result in Problem 4 suggests that the Illinois method in Problem 2 is modified by taking

$$\beta = f[x_{n+1}, x_n]/f[x_n, x_{n-1}], \qquad \theta = \begin{cases} \beta, & \text{if } \beta > 0; \\ \frac{1}{2}, & \text{if } \beta \le 0. \end{cases}.$$

Implement this modified method. Compare it with the unmodified Illinois method and with the safeguarded secant algorithm. As test equations use the following:
(a) A curve with one inflection point on $[0, 1]$:
$f(x) = x^2 - (1-x)^n$, $a = 25$, $b = 1$, $n = 2, 5, 10$.
(b) A family of curves which lie increasingly close to the x-axis for large n:
$f(x) = e^{-nx}(x-1) + x^n$, $a = 0.25$, $b = 1$, $n = 5, 10, 15$.
(c) A family of curves with the y-axis asymptotic:
$f(x) = (nx-1)/((n-1)x)$, $a = 0.01$, $b = 1$, $n = 2, 5, 10$.

6.3 Methods Using Derivatives

6.3.1 Newton's method

When $f'(x)$ is available then **Newton's method**[4] (6.1.26) is usually the method of choice for solving an equation $f(x) = 0$. As shown already in Section 1.2.1, Newton's method is based on approximating the curve $y = f(x)$ by its tangent at the point $(x_n, f(x_n))$, where x_n is the current approximation to the root. Assuming that $f'(x_n) \neq 0$, the next approximation x_{n+1} is determined as the abscissa of the point of intersection of the tangent with the x-axis (see Figure 1.2.3). This is equivalent to replacing the equation $f(x) = 0$ by

$$T_n(x) = f(x_n) + (x - x_n)f'(x_n) = 0, \tag{6.3.1}$$

where $T_n(x)$ is obtained by truncating the Taylor expansion of $f(x)$ at x_n after the first two terms. Hence x_{n+1} is determined from

$$x_{n+1} = x_n + h_n, \quad h_n = -f(x_n)/f'(x_n). \tag{6.3.2}$$

Clearly Newton's method can be viewed as the limit of the secant method when the interpolation points coalesce,

[4] Newton's original method was more complicated and not very similar to what is now known as his method.

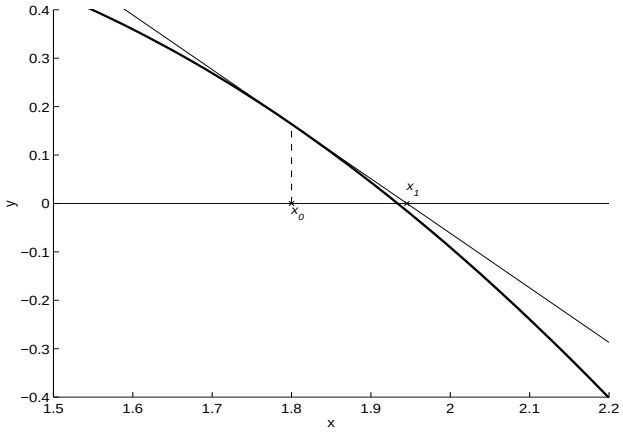


Figure 6.3.1. *Newton's method for the equation* $f(x) = (x/2)^2 - \sin x = 0$.

Example 6.3.1.

We want to compute the unique positive root of the equation $f(x) = (x/2)^2 - \sin x = 0$ (cf. Example 6.2.2) for which $f'(x) = x/2 - \cos x$. Starting from $x_0 = 1.8$ the first step of Newton's method is illustrated in Figure 6.3.1. The following Newton iterates are given in the table below (correct digits in x_n shown in bold):

n	x_n	$f(x_n)$	$f'(x_n)$	h_n
0	**1**.8	−0.163847 630878	1.127202 094693	−0.145357 812631
1	**1.9**45357 812631	0.015436 106659	1.338543 359427	0.011532 018406
2	**1.933**825 794225	0.000095 223283	1.322020 778469	0.000072 028582
3	**1.933753 76**5643	0.000000 003722	1.3219174 29113	0.000000 002816
4	**1.933753 762827**			

The number of correct digits approximately double in each iteration until the limiting precision is reached. Although the initial approximation is not very good, already x_4 is correct to twelve decimals!

If the iterations are broken off when $|h_n| < \delta$ it can be shown (see the error analysis below) that the truncation error is less than δ, provided that $|Kh_n| \leq 1/2$, where K is an upper bound for $|f''/f'|$ in the neighborhood of the root. This restriction is seldom of practical importance. However, rounding errors made in computing h_n must also be taken into account.

Note that when we approach the root, the *relative* precision in the computed values of $f(x_n)$ usually becomes less and less. Since $f'(x_n)$ is only used for computing h_n it need not be computed to much greater *relative precision* than $f(x_n)$, In the above example we could have used $f'(x_2)$ instead of $f'(x_n)$ instead for $n > 2$ without much affecting the convergence. Such a simplification is of great importance when Newton's method is used on *systems* of nonlinear equations; see Volume II, Section 11.2.4.

We first consider the **local** convergence of Newton's method, that is the convergence in a neighborhood of a root α. Assume that $f'(x)$ is continuous in a neighborhood of α and that $f'(\alpha) \neq 0$. Expanding f in a Taylor series about x_0 we get

$$0 = f(\alpha) = f(x_n) + (\alpha - x_n)f'(\xi_n), \quad \xi_n \in \text{int}(x_n, \alpha).$$

We let $\epsilon_n = x_n - \alpha$ denote the error in the approximation x_n. Subtracting (6.3.1) with $x = x_{n+1}$ and using $x_{n+1} - x_n = \epsilon_{n+1} - \epsilon_n$, we have

$$-\epsilon_n f'(\xi_n) = (\epsilon_{n+1} - \epsilon_n)f'(x_n),$$

or after dividing by $f'(x_n)$

$$\epsilon_{n+1} = \left(1 - \frac{f'(\xi_n)}{f'(x_n)}\right)\epsilon_n, \quad n = 0, 1, 2, \ldots.$$

Hence, if x_0 is sufficiently close to α, then $\lim_{n\to\infty} x_n = \alpha$. In other words α is a point of attraction of the Newton iteration and *Newton's method always converges (to a simple root) from a sufficiently good starting approximation.*

For convergence f need only have *one* continuous derivative. To get a more precise relation between ϵ_{n+1} and ϵ_n we assume in what follows that f has *two* continuous derivatives.

Theorem 6.3.1. *Assume that α is a simple root of the equation $f(x) = 0$, i.e., $f'(\alpha) \neq 0$. If f' exists and is continuous in a neighborhood of α, then the convergence order of Newton's method is at least equal to two.*

Proof. A Taylor expansion of f yields

$$0 = f(\alpha) = f(x_n) + (\alpha - x_n)f'(x_n) + \frac{1}{2}(\alpha - x_n)^2 f''(\zeta_n), \quad \zeta_n \in \text{int}(x_n, \alpha).$$

Subtracting $f(x_n) + (x_{n+1} - x_n)f'(x_n) = 0$ and solving for $\epsilon_{n+1} = x_{n+1} - \alpha$ we get

$$\epsilon_{n+1} = \frac{1}{2}\epsilon_n^2 \frac{f''(\zeta_n)}{f'(x_n)}, \quad \zeta_n \in \text{int}(x_n, \alpha). \tag{6.3.3}$$

Provided that $f'(\alpha) \neq 0$, it follows that (6.1.22) is satisfied with $p = 2$ and the asymptotic error constant is

$$C = \frac{1}{2}\left|\frac{f''(\alpha)}{f'(\alpha)}\right|. \tag{6.3.4}$$

If $f''(\alpha) \neq 0$, then $C > 0$ and the rate of convergence is quadratic. □

Note that *the relation* (6.3.3) *between the errors only holds as long as the round-off errors in the calculations can be ignored.* As pointed out in Section 6.1.3, the limiting factor for the accuracy, which can be achieved in calculating the root, is always limited by the accuracy of the computed values of $f(x)$.

So far we have assumed that α is a simple root. Suppose now that α is a root of multiplicity $q > 1$. Then by Taylor's formula we have (cf. (6.1.9))

$$f'(x) = \frac{1}{(q-1)!}(x-\alpha)^{q-1} f^{(q)}(\xi'), \quad \xi' \in \operatorname{int}(x, \alpha).$$

It follows that if x_n is close to α, then the Newton correction will satisfy

$$h_n = \frac{f(x_n)}{f'(x_n)} \approx \frac{1}{q}(x_n - \alpha) = \epsilon_n/q.$$

For the corresponding errors we have

$$\epsilon_{n+1} = \epsilon_n - \epsilon_n/q = (1 - 1/q)\epsilon_n,$$

which shows that for a root of multiplicity $q > 1$ Newton's method only converges *linearly* with rate $C = 1 - 1/q$. (The same is true of other methods which have quadratic or higher rate of convergence for simple roots.) For $q > 2$ this is much slower even than for the bisection method! Note also that when $x_n \to \alpha$ both $f(x_n) \to 0$ and $f'(x_n) \to 0$. Therefore rounding errors may seriously affect the Newton correction when evaluated close to α, and some safeguarding is essential; see Section 6.3.2.

When the multiplicity p of a root is known a priori the modified Newton's method

$$x_{n+1} = x_n - p\frac{f(x_n)}{f'(x_n)}, \tag{6.3.5}$$

is easily shown to have quadratic convergence. For a root of unknown multiplicity we can use the following observation. From (6.1.9) it follows that the equation $u(x) = 0$, where

$$u(x) = f(x)/f'(x), \tag{6.3.6}$$

always has *a simple root at* $x = \alpha$. Hence if Newton's method is applied to this equation it will retain its quadratic rate of convergence independent of the multiplicity of α as a root to $f(x) = 0$. The iteration becomes

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{(f'(x_n))^2 - f(x_n)f''(x_n)}, \tag{6.3.7}$$

and thus requires the evaluation also of $f''(x_n)$.

Newton's method applied to the equation $f(x) = x^p - c = 0$ can be used to compute $c^{1/p}$, $p = \pm 1, \pm 2, \ldots$. The sequence $x_1, x_2, x_3, \ldots$, is then computed recursively from

$$x_{n+1} = x_n - \frac{x_n^p - c}{p x_n^{p-1}},$$

which can be written as

$$x_{n+1} = \frac{1}{p}\left((p-1)x_n + \frac{c}{x_n^{p-1}}\right) = \frac{x_n}{(-p)}[(1-p) - c x_n^{-p}]. \tag{6.3.8}$$

It is convenient to use the first expression in (6.3.8) when $p > 0$ and the second when $p < 0$. This iteration formula is often used for calculating, e.g., $\sqrt{c}$, $\sqrt[3]{c}$, and $1/\sqrt{c}$, corresponding to $p = 2, 3$, and -2 respectively. Note also that $1/c$, $(p = -1)$ can be computed by the iteration

$$x_{n+1} = x_n + x_n(1 - cx_n) = x_n(2 - cx_n),$$

using only multiplications and addition. In some early computers, which lacked built-in division, this iteration was used to implement division, i.e. b/c was computed as $b \times (1/c)$.

Example 6.3.2.

We want to construct an algorithm based on Newton's method for efficiently computing the square root of any given floating point number a. If we first shift the mantissa so that the exponent becomes even, $a = c \cdot 2^{2e}$, and $1/2 \le c < 2$, then

$$\sqrt{a} = \sqrt{c} \cdot 2^e.$$

We need only consider the reduced range $1/2 \le c \le 1$ since for $1 < c \le 2$ we can compute $\sqrt{1/c}$ and invert.

To find an initial approximation x_0 to start the Newton iterations when $1/2 \le c < 1$, we can use linear interpolation of $x = \sqrt{c}$ between the endpoints $1/2, 1$, giving

$$x_0(c) = \sqrt{2}(1 - c) + 2(c - 1/2)$$

($\sqrt{2}$ is precomputed). The iteration then proceeds with

$$x_{n+1} = \frac{1}{2}\Big(x_n + \frac{c}{x_n}\Big), \quad n = 0, 1, 2, \ldots. \tag{6.3.9}$$

For $c = 3/4$ ($\sqrt{c} = 0.86602540378444$) the result is $x_0 = (\sqrt{2} + 2)/4$ and (correct digits in boldface)

$$x_0 = \mathbf{0.8}5355339059327, \quad x_1 = \mathbf{0.866}11652351682,$$
$$x_2 = \mathbf{0.86602540}857756, \quad x_3 = \mathbf{0.86602540378444},$$

Three iterations suffice to give full IEEE double precision accuracy and the quadratic rate of convergence is apparent.

6.3.2 Global Convergence of Newton's Method

It is easy to construct examples where Newton's method converges very slowly or not at all. Recall, e.g., that for a root of multiplicity $q > 1$ convergence is linear with rate $1 - 1/q$. For $q = 20$ it will take 45 iterations to gain one more decimal digit.

Example 6.3.3.

The equation $f(x) = \sin x = 0$, has exactly one root $\alpha = 0$ in the interval $|x| < \pi/2$. Newton's method becomes

$$x_{n+1} = x_n - \tan x_n, \quad n = 0, 1, 2, \ldots.$$

If we choose the initial value $x_0 = x^*$ such that $\tan x^* = 2x^*$, then $x_1 = -x_0$, $x_2 = -x_1 = x_0$. Hence the successive approximations show a cyclic behavior!

Newton's method will converge for any starting value such that $|x_0| < x^*$. The critical value can be shown to be $x^* = 1.16556\ldots$.

In some simple cases the global convergence of Newton's method may be easy to verify. Two examples are given in the following theorems.

Theorem 6.3.2. *Suppose that $f'(x)f''(x) \neq 0$ in an interval $[a, b]$, where $f''(x)$ is continuous and $f(a)f(b) < 0$. Then if*

$$\left|\frac{f(a)}{f'(a)}\right| < b - a, \qquad \left|\frac{f(b)}{f'(b)}\right| < b - a,$$

Newton's method converges from an arbitrary $x_0 \in [a, b]$.

Proof. The theorem follows easily by inspecting Figure 6.4.2. □

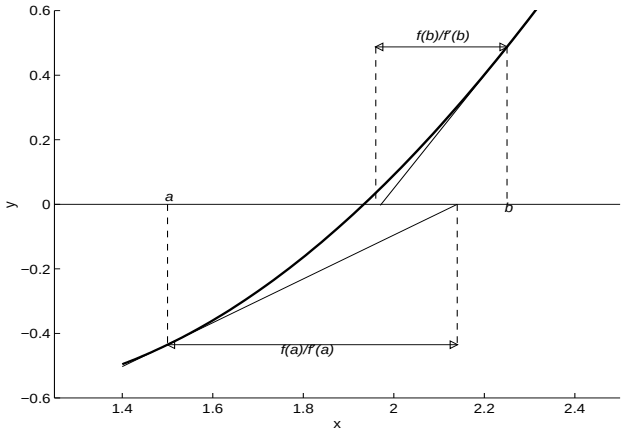


Figure 6.3.2. *A situation where Newton's method converges from any $x_0 \in [a, b]$.*

Lemma 6.3.3. *Let $[a, b]$ be an interval such that $f(a)f(b) < 0$. Assume that the so-called* **Fourier conditions** *are satisfied, i.e., $f'(x)f''(x) \neq 0$ in $[a, b]$, with $f''(x)$ continuous and $f(x_0)f''(x_0) > 0$, for $x_0 \in [a, b]$. Then the sequence $\{x_0, x_1, x_2, \ldots\}$ generated by Newton's method converges monotonically to a root $\alpha \in [a, b]$.*

Proof. We can assume that $f''(x) > 0$; otherwise consider the equation $-f(x) = 0$. Assume first that $f'(x) < 0$ in the interval. Since by assumption $f(x_0) \geq 0$, this corresponds to the situation in Figure 6.4.2, with $b = x_0 > \alpha$. Clearly it holds that $x_1 > x_0$ and since the curve lies to the left of the tangent in x_0 we also have $x_1 > \alpha$. The case when $f'(x) < 0$ can be treated similarly. The theorem now follows by induction. □

Newton's method can be safeguarded by taking a bisection step whenever a Newton step "fails" in some sense. Assume that initially $a < b$ and $f(a)f(b) < 0$ and x is either equal to a or b. At each step a new approximation x' is computed and a, b are updated to a', b' as follows:

- If the Newton iterate $x' = x - f(x)/f'(x)$ lies in (a, b), then accept x'; otherwise take a bisection step, i.e. set $x' = (a + b)/2$.
- Set either $a' = x$, $b' = b$ or $a' = a$, $b' = x$, where the choice is made so that $f(a')f(b') \leq 0$.

This ensures that at each step the interval $[a', b']$ contains a root.

When checking if $z \in (a, b)$, it is important to avoid division by $f'(x)$, since this may cause overflow or division by zero. Hence, we note $z \in (a, b)$ if and only if

$$b - z = b - x + f(x)/f'(x) > 0 \quad \text{and} \quad z - a = x - a - f(x)/f'(x) \geq 0.$$

If $f'(x) > 0$ these two inequalities are equivalent to

$$(b - x)f'(x) > -f(x) \quad \text{and} \quad (x - a)f'(x) > f(x).$$

The case when $f'(x) < 0$ is analyzed similarly, giving

$$(b - x)f'(x) < -f(x) \quad \text{and} \quad (x - a)f'(x) < f(x).$$

In either case only one of the inequalities will be nontrivial depending on whether $f(x) > 0$ or not.

6.3.3 Newton Method for Complex Roots

Newton's method is based on approximating f with the linear part of its Taylor expansion. Taylor's theorem is valid for a complex function $f(z)$ around a point of analyticity a (see Sec. 3.1.3). Hence Newton's method applies also to an equation $f(z) = 0$, where $f(z)$ is a complex function, analytic in a neighborhood of a root α. An important example is when f is a polynomial; see Sec. 6.5.

Let $z = x + iy$, $f(z) = u(x, y) + iv(x, y)$, and consider the absolute value of $f(z)$

$$\phi(x, y) = |f(x + iy)| = \sqrt{u(x, y)^2 + v(x, y)^2}.$$

This is a differentiable function as a function of (x, y), except where $f(z) = 0$. The gradient of $\phi(x, y)$ is

$$\operatorname{grad} \phi = (\phi_x, \phi_y) = \frac{1}{\phi}\,(uu_x + vv_x, uu_y + vv_y) \tag{6.3.10}$$

where $u_x = \partial u/\partial x$, $u_y = \partial u/\partial y$, etc. Using the Cauchy–Riemann equations $u_x = v_y$, $u_y = -v_x$ we calculate (see Henrici [9, §6.1.4]).

$$\frac{f(z)}{f'(z)} = \frac{u+iv}{u_x+iv_x} = \frac{(uu_x+vv_x)+i(uu_y+vv_y)}{u_x^2+v_x^2},$$

A comparison with (6.3.10) shows that the Newton step

$$z_{k+1} - z_k = -f(z_k)/f'(z_k), \tag{6.3.11}$$

is in the direction of the negative gradient of $|f(z_k)|$, i.e., in the direction of strongest decrease of $|f(z)|$.

The geometry of the complex Newton iteration is considered in [29].

Theorem 6.3.4.

Let the function $f(z) = f(x+iy)$ be analytic and $z_k = x_k + iy_k$ be a point such that $f(z_k)f'(z_k) \neq 0$. Let z_{k+1} be the next iterate of Newton's method (6.3.11). *Then $z_{k+1} - z_k$ is in the direction of the negative gradient of $\phi(x,y) = |f(x+iy)|$ and therefore orthogonal to the level set of $|f|$ at (x_k, y_k). If T_k is the tangent plane of $\phi(x,y)$ at (x_k, y_k) and L_k is the line of intersection of T_k with the xy-plane then (x_{k+1}, y_{k+1}) is the point on L_k closest to (x_k, y_k).*

Proof. See [29]. □

Newton's method is very efficient if started from an initial approximation sufficiently close to a simple zero. If this is not the case Newton's method may converge slowly or even diverge. In general, there is no guarantee that z_{n+1} is closer to the root than z_n, and if $f'(z_n) = 0$ the next iterate is not even defined.

The following important theorem gives rigorous sufficient conditions for the global convergence of Newton's method. For generality we formulate it for complex zeros of a complex valued function $f(z)$.

Theorem 6.3.5.

Let $f(z)$ be a complex function of a complex variable. Let $f(z_0)f'(z_0) \neq 0$ and set $h_0 = -f(z_0)/f'(z_0)$, $x_1 = x_0 + h_0$. Assume that $f(z)$ is twice continuously differentiable in the disk $K_0 : |z - z_1| \leq |h_0|$, and that

$$2\,|h_0|M_2 \leq |f'(z_0)|, \quad M_2 = \max_{z\in K_0} |f''(x)|. \tag{6.3.12}$$

Let z_k be generated by Newton's method

$$z_{k+1} = z_k - \frac{f(z_k)}{f'(z_k)}, \quad k = 1, 2, \ldots.$$

Then $z_k \in K_0$ and we have $\lim_{k\to\infty} z_k = \zeta$, where ζ is the only zero of $f(z)$ in K_0. Unless ζ lies on the boundary of K_0, ζ is a simple zero. Further we have the relations

$$|\zeta - z_{k+1}| \leq \frac{M_2}{2|f'(z_k)|}|z_k - z_{k-1}|^2, \quad k = 1, 2, \ldots. \tag{6.3.13}$$

In case of a real function $f(x)$ of a real variable the disk K_0 can be replaced by the closed interval $K_0 = \operatorname{int}[x_0, x_0 + 2h_0]$.

Proof. See Ostrowski [19, Theorem 7.2] □

A generalization of this result, Theorem 11.1.7 (the Newton–Kantorovich Theorem), for the multivariate Newton method, will be proved in Volume II.

Since the Newton step is in the direction of the negative gradient of $|f(z)|$ at $z = z_k$, it will necessarily give a decrease in $|f(z_k)|$ if a short enough step in this direction is taken. A modified Newton method based on the descent property and switching to standard Newton when the condition (6.3.12) is satisfied, will be described in Sec. 6.5.6.

6.3.4 An Interval Newton Method

.

Sometimes it is desired to compute a tiny interval that is guaranteed to enclose a real simple root x^* of $f(x)$, even when rounding errors are taken into account. This can be done using an adaptation of Newton's method to interval arithmetic method due to Moore [17].

Suppose that the function $f(x)$ is continuously differentiable. Using the notation in Sec. 2.3.7, let $f'([x_0])$ denote an interval containing $f'(x)$ for all x in a finite interval $[x] := [a, b]$. Define the Newton operator on $N[x]$ by

$$N([x]) := m - \frac{f(m)}{f'([x])}. \tag{6.3.14}$$

where $m = \operatorname{mid}([x]) = \frac{1}{2}(a + b)$.

Theorem 6.3.6.

If $\alpha \in [x]$ is a zero of $f(x)$, then $\alpha \in N([x])$. If $N([x]) \subseteq [x]$, then $f(x)$ has one and only one zero in $N([x])$.

Proof. Suppose α is a zero of $f(x)$ in $[x]$. If $0 \in f'([x])$ then $N([x]) = [-\infty, \infty]$. Otherwise, by the mean value theorem

$$0 = f(\alpha) = f(m) + f'(\xi)(\alpha - m), \quad \xi \in \operatorname{int}[\alpha, m] \subseteq [x].$$

This implies that $\alpha = m - f(m)/f'(\xi) \subseteq N([x])$, which proves the first statement.

If $N([x]) \subseteq [x]$ then $f'([x]) \neq 0$ on $[x]$. Then by the mean value theory there are ξ_1 and ξ_2 in $[x]$ such that

$$\begin{aligned}(m - f(m)/f'(\xi_1)) - a &= -f(a)/f'(\xi_1),\\ b - (m - f(m)/f'(\xi_2)) &= f(b)/f'(\xi_2).\end{aligned}$$

Because $N([x]) \subseteq [a, b]$, the product of the left sides is positive. But since $f'(\xi_1)$ and $f'(\xi_2)$ have the same sign this means that $f(a)f(b) < 0$ and f has therefore a zero in $[x]$.

Finally, there cannot be two or more zeros in $[x]$, because then we would have $f'(c) = 0$ for some $c \in [x]$. □

In the interval Newton method, a starting interval $[x_0]$ is chosen, and we compute for $k = 0, 1, 2, \ldots$ a sequence of intervals $[x_{k+1}]$ given by

$$N([x_k]) = \operatorname{mid}([x_k]) - \frac{f(\operatorname{mid}[x_k])}{f'([x_k])}.$$

If $N([x_k]) \subset [x_k]$ we set $[x_{k+1}] = N([x_k]) \cap [x_k]$. Otherwise, if $N([x_k]) \cap [x_k]$ is empty, we know that $[x_k]$ does not contain a root and stop. In neither condition holds we stop, subdivide the initial interval and start again. It can be shown that if $[x_0]$ does not contain a root then after a finite number of steps the iteration will stop with an empty interval.

If we are close enough to a zero, then the length of the intervals $[x_k]$ will converge quadratically to zero, just as the standard Newton method.

Example 6.3.4.

Take $f(x) = x^2 - 2$ and $[x_0] = [1, 2]$. Using interval Newton method

$$N([x_k]) = \operatorname{mid}([x_k]) - \frac{(\operatorname{mid}[x_k])^2 - 2}{2\,[x_k]}, \quad [x_{k+1}] = N([x_k]) \cap [x_k].$$

we obtain the sequence of intervals

$$[x_1] = N([x_0]) = 1.5 - \frac{2.25 - 2}{2[1, 2]} = [22/16, 23/16] = [1.375, 1.4375],$$

$$[x_2] = N([x_1]) = \frac{45}{32} - \frac{(45/32)^2 - 2}{2[22/16, 23/16]} = \frac{45}{32} - \frac{(45)^2 - 2(32)^2}{128\,[22, 23]} \subset [1.41406, 1.41442].$$

The quadratic convergence of the radius of the intervals is evident:

$$0.5, 0.03125, 0.00036, \ldots.$$

The interval Newton method, is well suited to determine *all zeros in a given interval*. Divide the given interval into subintervals and for each subinterval $[x]$ check whether the condition $N([x]) \subseteq [x]$ in Theorem 6.3.6 holds. If this is the case, we continue the interval Newton iterations, and if we are close enough the iterations converge towards a root. If the condition is not satisfied but $N([x]) \cap [x]$ is empty then there is no zero in the subinterval and this can be discarded. If the condition fails but $N([x]) \cap [x]$ is not empty, then subdivide the interval and try again. The calculations can be organized so that we have a queue of intervals waiting to be precessed. Intervals may be added or removed form the queue. When the queue is empty we are done.

The above procedure may not always work. Its performance will depend among other things on the sharpness of the inclusion of the derivative $f'([x])$. Things will go wrong, e.g., in case of multiple roots where $N([x]) = [-\infty, \infty]$.

6.3.5 Higher Order Methods

Newton's method has quadratic convergence, which means that the number of significant digits approximately doubles in each iteration. Although there is rarely any practical need for methods of higher order of convergence such methods may be useful in special applications.

For the following discussion we assume that α is a simple zero of f and that f has a sufficient number of continuous derivatives in a neighborhood of α. We first briefly review some famous methods with cubic convergence for simple roots of $f(x) = 0$.

Newton's method was derived by approximating the function $f(x)$ with its linear Taylor approximation. Higher order iteration methods can be constructed by including more terms from the Taylor expansion. The quadratic Taylor approximation of the equation $f(x_n + h) = 0$ is

$$f(x_n) + hf'(x_n) + \frac{h^2}{2}f''(x_n) = 0, \quad h = x - x_n. \tag{6.3.15}$$

Assuming that $f'(x_n)^2 \geq 2f(x_n)f''(x_n)$, the solutions of this quadratic equation are real and equal to

$$h_n = -\frac{f'(x_n)}{f''(x_n)}\left(1 \pm \sqrt{1 - 2\frac{f(x_n)f''(x_n)}{(f'(x_n))^2}}\right).$$

Rearranging and taking the solution of smallest absolute value we get

$$x_{n+1} = x_n - u(x_n) \cdot \frac{2}{1 + \sqrt{1 - 2t(x_n)}}, \tag{6.3.16}$$

where we have introduced the notation

$$u(x) = \frac{f(x)}{f'(x)}, \qquad t(x) = u(x)\frac{f''(x)}{f'(x)}. \tag{6.3.17}$$

The iteration (6.3.16) is **Euler's iteration method**.

Assuming that $|t(x_n)| \ll 1$ and using the approximation

$$\frac{2}{1 + \sqrt{1 - 2t(x_n)}} \approx 1 + \tfrac{1}{2}t(x_n), \tag{6.3.18}$$

valid for $|t| \ll 1$, we obtain another third order iteration method usually also attributed to Euler

$$x_{n+1} = x_n - u(x_n)\big(1 + \tfrac{1}{2}t(x_n)\big). \tag{6.3.19}$$

A different method of cubic convergence is obtained by using a rational approximation of (6.3.18)

$$x_{n+1} = x_n - u(x_n) \cdot \frac{1}{1 - \frac{1}{2}t(x_n)} = x_n - \frac{f(x_k)}{f'(x_k) - \frac{f''(x_k)}{2f'(x_k)}f(x_k)}. \tag{6.3.20}$$

This is **Halley's**[5] iteration method [8], which has the distinction of being the most frequently rediscovered iteration method. Halley's method has a simple geometric interpretation. Consider a hyperbola

$$h(x) = b + \frac{a}{x - c},$$

where a, b, c are determined so that $h(x)$ is osculatory to f at $x = x_n$, i.e., is tangent to the curve at this point and has the same curvature there. Then x_{n+1} is the intersection of this hyperbola with the x-axis.

The methods (6.3.19) and (6.3.20) correspond to the (1,0) and (0,1) Padé approximations of Euler's method. We now show that both are of third order for simple zeros. Following Gander [6] we consider an iteration function of the general form

$$\phi(x) = x - u(x)H(t(x)), \tag{6.3.21}$$

where $u(x)$ and $t(x)$ are defined by (6.3.17). Differentiating (6.3.21) and using $u'(x) = 1 - t(x)$ we get

$$\phi'(x) = 1 - (1 - t(x))H(t) - u(x)H'(t)t'(x).$$

Since $u(\alpha) = t(\alpha) = 0$ it follows that $\phi'(\alpha) = 1 - H(0)$. Hence if $H(0) = 1$ then $\phi'(\alpha) = 0$ and the iteration function (6.3.21) is at least of second order. Differentiating once more and putting $x = \alpha$ we get

$$\phi''(\alpha) = t'(\alpha)H(0) - 2u'(\alpha)H'(0)t'(\alpha) = t'(\alpha)(H(0) - 2H'(0)).$$

Hence $\phi'(\alpha) = \phi''(\alpha) = 0$ and the method (6.3.21) at least of third order if the conditions

$$H(0) = 1, \qquad H'(0) = 1/2 \tag{6.3.22}$$

are satisfied. For Euler's and Halley's method we have

$$H(t) = 2\left(1 + \sqrt{1 - 2t}\right)^{-1}, \qquad H(t) = (1 - \tfrac{1}{2}t)^{-1},$$

respectively, and both these methods satisfy the conditions in (6.3.22). (Verify this!)

Example 6.3.5.

Using (6.3.20) a short calculation shows that Halley's method for solving the equation $f(x) = x^2 - c = 0$ can be written

$$x_{n+1} = x_n - 2x_n \frac{x_n - c/x_n}{3x_n + c/x_n}, \quad n = 0, 1, 2, \ldots. \tag{6.3.23}$$

(see Problem 10). For $c = 3/4$ we and using the initial approximation $x_0 = (\sqrt{2} + 2)/4$ obtained by linear interpolation in Example 6.3.2 we obtain the following result (correct digits in boldface)

$$x_0 = \mathbf{0.8}5355339059327, \quad x_1 = \mathbf{0.86602}74293290, \quad x_2 = \mathbf{0.86602540378444}.$$

[5] Edmund Halley (1656–1742), an English astronomer, who predicted the periodic reappearance (c:a 75 years) of a comet named after him.

Already two iterations give a result correct to 14 digits. Compared to Newton's method we have gained one iteration, but each iteration is more costly.

It is not difficult to construct iteration methods of arbitrarily high order for solving $f(x) = 0$. It can be shown ([25, Theorem 5.3]) that any iteration function of order p must depend explicitly on the first $p-1$ derivatives of f. Consider the Taylor expansion at x_n

$$0 = f(x_n + h) = f(x_n) + hf'(x_n) + \sum_{k=2}^{p-1} \frac{h^k}{k!} f^{(k)}(x_n) + O(h^p). \tag{6.3.24}$$

Neglecting the $O(h^p)$-term this is a polynomial equation of degree $p-1$ in h. Assuming that $f'(x_n) \neq 0$ we could solve this by the fixed point iteration $h_i = F(h_{i-1})$, where

$$F(h) = -\frac{f(x_n) + \sum_{k=2}^{p-1} h^k f^{(k)}(x_n)/k!}{f'(x_n)},$$

taking h_0 to be the Newton correction.

To get an explicit method of order p we write

$$\frac{f(x_n)}{f'(x_n)} \equiv u = -h - \sum_{k=2}^{p-1} a_k h^k, \tag{6.3.25}$$

where

$$a_k = \frac{f^{(k)}(x_n)}{k! f'(x_n)}, \quad k \geq 2. \tag{6.3.26}$$

This can be interpreted as a formal power series in h (cf. Sec. 3.1.5). Reversing this series we can express h as a formal power series in u

$$h = -u - \sum_{k=2}^{p-1} c_k u^k + \cdots, \tag{6.3.27}$$

$$\begin{aligned} &c_2 = a_2, \qquad c_3 = 2a_2^2 - a_3, \qquad c_4 = 5a_2^3 - 5a_2 a_3 + a_4, \\ &c_5 = 14a_2^4 - 21a_2^2 a_3 + 6a_2 a_4 + 3a_3^2 - a_5, \ldots. \end{aligned} \tag{6.3.28}$$

More coefficients can easily be determined; see Problem 3.1.12. This leads to the **Schröder family** of iteration methods

$$x_{n+1} = x_n - u(x_n) - \sum_{k=2}^{p-1} c_k(x_n) u^k(x_n), \tag{6.3.29}$$

(E. Schröder [22]). If f is analytic these can be shown to have convergence order p for simple roots. For a proof see Henrici [9, p. 529].

Setting $p = 2$ gives Newton's method and for $p = 3$ we get the third order method of Euler (6.3.19). The family (6.3.29) of high order methods makes use of polynomials in u. As for the case $p = 3$, variants of these using rational expressions

in u can be derived from Padé approximants of these polynomials; see Traub [25, Sec. 5.2]. There are indications that methods which use rational approximations with about equal degree of nominator and denominator are best. For example, for $p = 4$, the rational approximation

$$1 + c_2u + c_3u^2 = \frac{c_2 + (c_2^2 - c_3)u}{c_2 - c_3u} + \mathcal{O}(u^3)$$

can be used to derive an alternative method of order 4.

We now introduce a rational family of iteration methods of arbitrary order, which is very convenient to use. First note that Halley's method can also be derived as follows. Starting from (6.3.15) we get

$$h_n = -f(x_n)\Big/\Big(f'(x_n) + \frac{h_n}{2}f''(x_n)\Big).$$

Replacing h_n in the denominator by the Newton correction $-f(x_n)/f'(x_n)$, does not change the order of the method and leads to (6.3.20). We note that this can be written $x_{n+1} = x_n + B_2(x_n)$, where

$$B_2(x) = -f(x)\ \frac{f'(x)}{\det\begin{pmatrix} f'(x) & \frac{f''(x)}{2!} \\ f(x) & f'(x) \end{pmatrix}}.$$

This method belongs to a rational family of iteration function of arbitrary order, which we now give. Set $D_p(x) = \det(F_p)$, where

$$F_p(x) = \begin{pmatrix} f'(x) & \frac{f''(x)}{2!} & \dots & \frac{f^{(p-1)}(x)}{(p-1)!} & \frac{f^{(p)}(x)}{(p)!} \\ f(x) & f'(x) & \ddots & \ddots & \frac{f^{(p-1)}(x)}{(p-1)!} \\ 0 & f(x) & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \frac{f''(x)}{2!} \\ 0 & 0 & \dots & f(x) & f'(x) \end{pmatrix} \in \mathbf{R}^{p\times p}, \tag{6.3.30}$$

is a Toeplitz upper Hessenberg matrix defined with respect to the normalized derivatives of $f(x)$. (Recall that a square matrix is called Toeplitz if its elements are identical along each diagonal.) The iteration

$$x_{n+1} = x_n + B_p(x_n), \quad B_p(x) = -f(x)\frac{\det(F_{p-2}(x))}{\det(F_{p-1}(x))} \tag{6.3.31}$$

can be shown to be of order p for simple roots.

The determinant formula (6.3.31) is attractive since it leads to a simple implementation. To evaluate $\det(F_p(x))$ we use Gaussian elimination (without pivoting) to compute the LU factorization

$$F_p(x_n) = L_pU_p, \qquad \operatorname{diag}(U_p) = (u_{11}, u_{22}, \dots, u_{pp}).$$

where L_p is unit lower triangular and U_p upper triangular; see Sec. 1.3.4. Then $\det(F_p) = u_{11}u_{22}\dots u_{pp}$. Hence the ratio $\det(F_p(x))/\det(F_{p-1}(x))$ equals the *last diagonal element* u_{pp} in U_p. It follows that

$$x_{n+1} = x_n - f(x_n)/u_{kk}(x_n), \quad k = 1:p, \tag{6.3.32}$$

gives the result of a sequence of iteration formulas of order $2:p+1$.

Note that the Gaussian elimination is simplified by the fact that $F_p(x_n)$ is Hessenberg. Only the $p-1$ subdiagonal elements need to be eliminated, which requires $\frac{1}{2}p^2$ flops. Further, it is possible to implement the computation of $\operatorname{diag}(U_p)$ using only two row vectors; see Problem 14.

The main drawback to the use of higher order iteration methods is the computational cost of evaluating higher derivatives of f. However, if f satisfies a differential equation higher order derivatives can be calculated by differentiating the differential equation.

Example 6.3.6.

Methods using high order derivatives are useful in particular when seeking zeros of a function satisfying a (usually second order) differential equation. The Bessel functions of the first kind $\mathcal{J}_\nu(x)$ satisfies the differential equation

$$x^2y'' + xy' + (x^2 - \nu^2)y = 0.$$

The smallest zero ξ of $\mathcal{J}_0(x)$ is close to $x_0 = 2.40$, and

$$\mathcal{J}_0(x_0) = 0.00250\,76832\,9724, \qquad \mathcal{J}_0'(x_0) = -\mathcal{J}_1(x) = -0.52018\,52681\,8193.$$

Differentiating the differential equation for $\mathcal{J}_0(x)$ we get

$$xy^{(k+1)} + ky^{(k)} + xy^{(k-1)} + (k-1)y = 0, \quad k \geq 1,$$

which gives a recursion for computing higher derivatives. Taking $p = 5$

$$y''(x_0)/2! = 0.10711\,80892\,2261, \qquad y'''(x_0)/3! = 0.05676\,83752\,3951,$$
$$y^{iv}(x_0)/4! = -0.00860\,46362\,1903.$$

Forming the Toeplitz matrix F_4 and computing the diagonal elements of U in the LU factorization, we obtain using (6.3.32) the following sequence of approximations to ξ:

$$\mathbf{2.40482}\,07503 \quad \mathbf{2.40482\,55}406 \quad \mathbf{2.40482\,55576}\,7553, \quad \mathbf{2.40482\,55576\,9573}$$

(correct digits shown in boldface).

Review Questions

1. (a) Under what assumptions is convergence of Newton's method quadratic?

(b) Device an example where Newton's method diverges, even though the equation has real roots.

2. Describe an iteration for the division-free computation of the reciprocal of a positive number c. Determine the largest set of starting values x_0 such that the iterates converge to $1/c$.

3. The equation $f(x) = \sin x = 0$ has one trivial root $x = 0$ in the interval $(-\pi/2, \pi/2)$. Show that for an initial approximation x_0 chosen so that $\tan x_0 = 2x_0$ Newton's method cycles, and $x_{2k} = x_0$ for all $k \geq 0$!

4. (a) Assume that f is continuously differentiable in a neighborhood of a double root α of the equation $f(x) = 0$. Describe how the equation can be converted to one with a simple root α.

(b) Discuss the case when $f(x) = 0$ has two distinct roots which *nearly* coincide.

Problems and Computer Exercises

1. (a) Compute $\epsilon_{n+1}/\epsilon_n^2$ for $n = 0, 1, 2$, and the limit, as $n \to \infty$, in Example 6.3.1.

(b) Treat the equation in Example 6.3.1 using $f'(x_2)$ as a fixed approximation to $f'(x_n)$ for $n > 2$. Compare the convergence of this simplified method with the true Newton method..

2. The equation $x^3 - 2x - 5 = 0$ is of historical interest because it was the one used by Wallis[6] to present Newton's method to the French Academy. Determine the roots of this equation.

Hint: It has one real and two complex roots.

3. Use Newton's method to determine the positive root of the equation to six correct decimals: (a) $x = 1 - e^{-2x}$; (b) $x \ln x - 1 = 0$

4. Determine the unique positive real root of the equation $x^q - x - 1 = 0$ for $q = 2 : 8$.

5. (a) Consider the Newton iteration used in Example 6.3.2 for computing square root. Show that the iterations satisfy

$$x_{n+1} - \sqrt{c} = \frac{1}{2x_n}(x_n - \sqrt{c})^2.$$

Use this relation to show that, for all $x_0 > 0$, convergence is monotone $x_1 \geq x_2 \geq x_3 \geq \cdots \geq \sqrt{c}$ and that $\lim_{n\to\infty} = \sqrt{c}$ (compare Figure 1.2.5).

(b) In Example 6.3.2 Newton's method was used to compute $\sqrt{c}$ for $1/2 \leq c \leq 1$. Determine the maximum error of the linear initial approximation used there. Then use the expression for the error in (a) to determine the number of iterations that suffices to give $\sqrt{c}$ with an error less than 10^{-14} for *all* c in

[6] John Wallis (1616–1703) the most influential English mathematician before Newton.

$[1/2, 1]$ using this initial approximation. Show that the influence of rounding errors is negligible.

6. Determine p, q and r so that the order of the iterative method

$$x_{n+1} = px_n + qc/x_n^2 + rc^2/x_n^5$$

for computing $\sqrt[3]{c}$ becomes as high as possible. For this choice of p, q and r, give a relation between the error in x_{n+1} and the error in x_n.

7. (A. Ben Israel) The function $f(x) = xe^{-x}$ has a unique zero $\alpha = 0$. Show that for any $x_0 > 1$ the Newton iterates move away from the zero.

8. The Cartesian coordinates of a planet in elliptic orbit at time t are equal to $ea(\sin(x), \cos(x))$, where a is the semi-major axis, and e the eccentricity of the ellipse. Using Kepler's laws of planetary motion it can be shown that the angle x, called the eccentric anomaly, satisfies Kepler's equation

$$x - e\sin x = M, \quad 0 < |e| < 1,$$

where $M = 2\pi t/T$ is the mean anomaly and T the orbital period.

(a) Newton used his method to solve Kepler's equation. Show that for each e, M there is one unique real solution $x = \alpha$, such that $M - |e| \leq \alpha < M + |e|$.

(b) Show that the simple fixed-point iteration method

$$x_{n+1} = e\sin x_n + M, \quad x_0 = 0,$$

is convergent.

(c) Study the convergence of Newton's method

$$x_{n+1} = x_n + \frac{e\sin x_n - x_n + M}{1 - e\sin x_n}.$$

8. Determine the multiple root $\alpha = 1$ of the equation $p(x) = (1-x)^5 = 0$, when the function is evaluated using Horner's scheme, i.e.,

$$p(x) = (((((x-5)x+10)x-10)x+5)x-1 = 0.$$

(a) Use bisection (cf. Algorithm 6.1.1) with initial interval $(0.9, 1.1)$ and tolerance $\tau = 10^{-8}$. What final accuracy is achieved?

(b) Use Newton's method, starting from $x_0 = 1.1$ and evaluating $p'(x)$ using Horner's scheme. Terminate the iterations when for the first time $|x_{n+1} - 1| > |x_n - 1|$. How many iterations are performed before termination? Repeat with a couple of other starting values!

(c) Same as (b), but perform one step of the modified Newton's method (6.3.5) with $x_0 = 1.1$ and $q = 5$. How do you explain that the achieved accuracy is much better than predicted by (6.1.10)?

9. Show that if Newton's method applied to the equation $u(x) = 0$, where $u(x) = f(x)/f'(x)$, then

$$x_{n+1} = x_n - \frac{u(x_n)}{1 - t(x_n)}, \quad t(x_n) = \frac{f(x_n)f''(x_n)}{(f'(x_n))^2}. \tag{6.3.33}$$

This transformation is most useful if an *analytical simplification* can be done such that $u(x)$ can be evaluated accurately also in a neighborhood of α.

10. (a) Show that Halley's method can also be derived by applying Newton's method to the equation $f(x)(f'(x))^{-1/2} = 0$.

(b) What is the efficiency index of Newton's and Halley's method, respectively, if it is assumed that evaluating each of f, f' and f'' takes one unit of work.

(c) Show that Halley's method applied to $f(x) = x^2 - c = 0$, $c > 0$, gives rise to the iteration

$$x_{n+1} = x_n \frac{x_n^2 + 3c}{3x_n^2 + c} = x_n - \frac{2x_n(x_n^2 - c)}{3x_n^2 + c}.$$

Apply Halley's method to $f(x) = x^k - c = 0$, $c > 0$.

11. (A. Ben Israel) Consider the **quasi-Halley method**

$$x_{n+1} = x_n - \frac{f(x_k)}{f'(x_k) - \dfrac{f'(x_k) - f'(x_{k-1})}{2(x_k - x_{k-1})f'(x_k)} f(x_k)}$$

where the second derivative $f''(x_k)$ has been approximated by a divided difference. Show that if f'' is Lipschitz continuous near a root α then

$$|\alpha - x_{k+1}| = O(|\alpha - x_k|^\gamma,$$

where γ satisfies the quadratic equation $\gamma^2 - 2\gamma - 1 = 0$. Conclude that the order of this method is approximately 2.41 as compared to 3 for Halley's method.

12. (Bailey et al. [1]) In 1976 Brent and Salamin independently discovered the following iteration, which generates a sequence $\{p_k\}$ converging quadratically to π:
Set $a_0 = 1$, $b_0 = 1/\sqrt{2}$, and $s_0 = 1/2$. For $k = 1, 2, 3, \ldots$ compute

$$\begin{aligned} &a_k = (a_{k-1} + b_{k-1})/2, \quad b_k = \sqrt{a_{k-1}b_{k-1}}, \\ &c_k = a_k^2 - b_k^2, \quad s_k = s_{k-1} - 2^k c_k, \quad p_k = 2a_k^2/s_k. \end{aligned}$$

Perform this iteration in IEEE 754 double precision. Verify the quadratic convergence by listing the errors in $|p_k - \pi|$ in successive iterations. How many iterations can you do before the error starts to grow? What is the best accuracy achieved?

13. In Example 6.3.4 the first two steps in the interval Newton method for solving the equation $x^2 - 2 = 0$ are shown. Implement this method and carry out the iterations until convergence.

14. (a) Compute $\det(F_p(x))$ in (6.3.30) for $p = 3$ and write down the corresponding rational fourth order iteration method in terms of u, a_2, a_3 in (6.3.28).

(b) Implement in MATLAB the iteration method (6.3.31) for arbitrary order p. Input should be an approximation x_n to the root, $f(x_n)$ and the row vector of scaled derivatives

$$\left(f'(x),\ \frac{f''(x)}{2!},\ \ldots,\ \frac{f^{(p-1)}(x)}{(p-1)!},\ \frac{f^{(p)}(x)}{p!}\right)$$

evaluated at $x = x_n$. Output should be the diagonal elements of U_p in the LU factorization of $F_p(x_n)$ and the sequence of approximations $x_{n+1,k} = x_n + f(x_n)/u_{kk}(x_n)$, $k = 1 : p$. Try to economize on memory requirement.

15. Write a program that computes the inverse of the error function $\operatorname{erf}(x)$ by solving the equation $\operatorname{erf}(x) - y = 0$, $0 \le y < 1$. Use Newton's method and the series expansion given in Example 1.3.4 to compute values of $\operatorname{erf}(x)$ and its derivative. Note that for large values of x $\operatorname{erf}(x) \approx 1 - 1/(\sqrt{\pi}x)$.

16. (a) Given $\sigma_1 \ge \sigma_1 \ge \ldots \ge \sigma_n > 0$, $c_1, c_2, \ldots c_n$, and $\alpha > 0$, consider the **secular equation** $\phi(\lambda) = \alpha$, where

$$\phi^2(\lambda) = \sum_{i=1}^{n} \left(\frac{\sigma_i c_i}{\sigma_i^2 + \lambda}\right)^2.$$

Show that $\phi(\lambda)$ is a convex and strictly decreasing function of λ. Conclude that if $\phi(0) > \alpha$, this equation has a unique root $\lambda > 0$ of smallest magnitude.

(b) Newton's method for solving $\phi(\lambda) - \alpha = 0$ is

$$\lambda_{k+1} = \lambda_k - h_k, \quad h_k = \frac{\phi(\lambda_k) - \alpha}{\phi'(\lambda_k)}.$$

Show that with $\lambda_0 = 0$ this method produces a strictly increasing sequence λ_k converging to the solution. Derive an explicit expression for h_k.

(c) The Newton iteration in (b) often converges very slowly. A more efficient method is obtained by instead applying Newton's method to the equation $h(\lambda) = 1/\phi(\lambda) = 1/\alpha$. Show that this iteration can be written

$$\lambda_{k+1} = \lambda_k - h_k \frac{\phi(\lambda_k)}{\alpha},$$

where h_k is the Newton step in (b).

(d) Let $\sigma_i = 1/i^2$, $c_i = 1/i^2 + 0.001$, $i = 1, 2, \ldots, 20$. Plot the function $f(\lambda)$ for $\lambda \in (0, 0.0005)$. Solve the equation $\phi(\lambda) = \alpha = 2.5$, using $\lambda_0 = 0$, comparing the two methods in (b) and (c).

6.4 Local Minimum of a Scalar Function

In this section we consider the problem of finding the minimum (maximum) of a real-valued function

$$\min g(x), \quad x \in [a, b], \tag{6.4.1}$$

which is closely related to that of solving a scalar equation. Problem (6.4.1) occurs as an important subproblem in methods for minimizing a function $\phi(z)$ of n variables, $z \in \mathbf{R}^n$, and for solving systems of nonlinear equation. For example, if z_k is the current approximation to the optimal point, the next approximation is often found by minimizing a function

$$g(\lambda) = \phi(x_k + \lambda d_k),$$

where d_k is a search direction, and the steplength λ is to be determined.

If g is differentiable in $[a, b]$, a **necessary condition** for an interior point of $\tau \in I$ to be a local minimum is that $g'(\tau) = 0$. If g' does not change sign on I it is also possible that the minimum is at a or b. If this is checked for separately, then it is possible to reduce the problem to a zero-finding problem for g'. Since g' also vanishes at a point of maximum and inflection, it is however necessary to check if the point found really is a minimum.

Most algorithms for minimizing a nonlinear function of one (or more) variables find at best a *local* minimum. For a function with several local minima, there is no guarantee that the *global* (lowest local) minimum in $[a, b]$ will be found. One obvious remedy is to try several different starting points and hope that the lowest of the local minima found is also the global minimum. This approach is neither efficient or safe. In practice we have to be content with algorithms which nearly always give correct results in most practical applications.

6.4.1 Unimodal Functions

A condition which ensures that a function g has a unique global minimum τ in $[a, b]$ is that g(x) is strictly decreasing for $a \le x < \tau$ and strictly increasing for $\tau < x \le b$. Such a function is called **unimodal**.

Definition 6.4.1.

The function $g(x)$ is **unimodal** *on $[a, b]$ if there exists a unique $\tau \in [a, b]$ such that, given any $c, d \in [a, b]$ for which $c < d$*

$$d < \tau \Rightarrow g(c) > g(d); \qquad c > \tau \Rightarrow g(c) < g(d). \tag{6.4.2}$$

This condition does not assume that g is differentiable or even continuous on $[a, b]$. For example, $|x|$ is unimodal on $[-1, 1]$.

6.4.2 Golden Section Search

We now describe an **interval reduction method** for finding the local minimum of a unimodal function, which only uses function values of g. It is based on the following lemma.

Lemma 6.4.2. *Suppose that g is unimodal on $[a, b]$, and τ is the point in Definition 6.4.1. Let c and d be points such that $a \le c < d \le b$. If $g(c) \le g(d)$ then $\tau \le d$, and if $g(c) \ge g(d)$ then $\tau \ge c$.*

Proof. If $d < \tau$ then $g(c) > g(d)$. Thus, if $g(c) \leq g(d)$ then $\tau \leq d$. The other part follows similarly. □

Assume that g is unimodal in $[a, b]$. Then using Lemma 6.4.2 it is possible to find a reduced interval on which g is unimodal by evaluating $g(x)$ at two interior points c and d such that $c < d$. Setting

$$[a', b'] = \begin{cases} [c, b], & \text{if } g(c) > g(d); \\ [a, d], & \text{if } g(c) < g(d). \end{cases}$$

we can enclose x^* in an interval of length at most equal to $\max(b - c, d - a)$. (If $g(c) = g(d)$ then $\tau \in [c, d]$, but we ignore this possibility.) To minimize this length one should take c and d so that $b - c = d - a$. Hence $c + d = a + b$, and we can write

$$c = a + t(b - a), \qquad d = b - t(b - a), \quad 0 < t < 1/2.$$

Then $d - a = b - c = (1 - t)(b - a)$, and by choosing $t \approx 1/2$ we can almost reduce the length of the interval by a factor $1/2$. However, $d - c = (1 - 2t)(b - a)$ must not be too small for the available precision in evaluating $g(x)$.

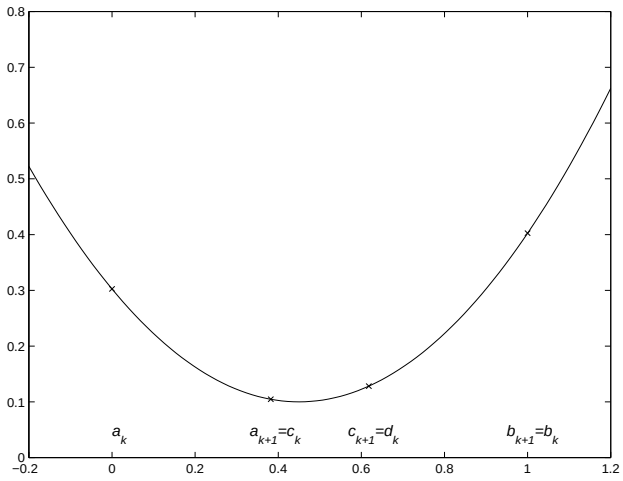


Figure 6.4.1. *One step of interval reduction, $g(c_k) \geq g(d_k)$.*

If we only consider one step the above choice would be optimal. Note that this step requires **two** function evaluations. A clever way to save function evaluations is to arrange it so that if $[c, b]$ is the new interval then d can be used as one of the points in the next step; similarly if $[a, d]$ is the new interval then c can be used at the next step. Suppose this can be achieved with a fixed value of t. Since $c + d = a + b$ the points lie symmetric with respect to the midpoint $\frac{1}{2}(a + b)$ and we need only consider the the first case. Then t must satisfy the following relation (cf. above and Figure 6.7.1)

$$d - c = (1 - 2t)(b - a) = (1 - t)t(b - a).$$

Hence t should equal the root in the interval $(0, 1/2)$ of the quadratic equation $1 - 3t + t^2 = 0$, which is $t = (3 - \sqrt{5})/2$. With this choice the length of the interval will be reduced by the factor

$$1 - t = 2/(\sqrt{5} + 1) = 0.618034\ldots$$

at each step, which is the **golden section** ratio. For example, 20 steps gives a reduction of the interval with a factor $(0.618034\ldots)^{20} \approx 0.661 \cdot 10^{-5}$.

Algorithm 6.4.1 *Golden Section Search.*

Let g be a given continuous function and $I = [a, b]$ an interval. The following algorithm computes an approximation $m \in I$ to a local minimum of $g(x)$, with an error less than a specified tolerance τ.

$xmin = goldsec(g, a, b, \tau);$
$t = 2/(3 + \sqrt{5});$
$c = a + t \cdot (b - a);$
$d = b - t \cdot (b - a);$
$gc = g(c); \quad gd = g(d);$
while $(d - c) > \tau \cdot \max(|c|, |d|)$
  if $gc \geq gd$ $\%Keep\ right\ endpoint\ b$
    $a = c; \quad c = d;$
    $d = b - t \cdot (b - a);$
    $gc = gd; \quad gd = g(d);$
  else $\%Keep\ left\ endpoint\ a$
    $b = d; \quad d = c;$
    $c = a + t \cdot (b - a);$
    $gd = gc; \quad gc = g(c);$
  end;
end;
$xmin = (c + d)/2;$

Rounding errors will interfere when determining the minimum of a scalar function $g(x)$. Because of rounding errors the computed approximation $fl(g(x))$ of a unimodal function $g(x)$ is not in general unimodal; cf. Figure 6.4.2. However, if we assume that in Definition 6.4.1 the points c and d satisfy $|c - d| > \tau$ for some small τ, the condition (6.4.2) will hold also for the computed function. For any method using only computed values of g there is a fundamental limitation in the accuracy of the computed location of the minimum point τ in $[a, b]$. The best we can hope for is to find $x_k \in [a, b]$ such that

$$g(x_k) \leq g(x^*) + \delta,$$

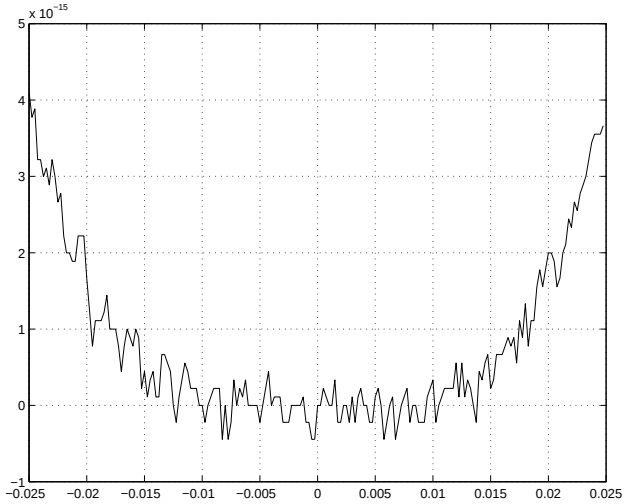


Figure 6.4.2. *The effect of rounding errors on minimizing a function.*

where δ is an upper bound of rounding and other errors in the computed function values $\bar{g}(x) = fl\,(g(x))$; If g is twice differentiable in a neighborhood of a minimum point τ then by Taylor's theorem

$$g(\tau + h) \approx g(\tau) + \tfrac{1}{2}h^2 g''(\tau).$$

This means that there is no difference in the floating point representation of $g(\tau+h)$ unless h is of the order of $\sqrt{u}$. Hence we can not expect τ to be determined with an error less than

$$\epsilon_\alpha = \sqrt{2\,\delta/|g''(x^*)|}, \tag{6.4.3}$$

unless we can also use values of g' or the function has some special form.

6.4.3 Minimization by Interpolation

For finding the minimum of a unimodal function g golden section search method has the advantage that linear convergence is guaranteed. In that respect it corresponds to the bisection method for finding a zero of a function. If the function is sufficiently smooth and we have a good initial approximation, then a process with superlinear convergence will be much faster. Such methods can be devised using interpolation by a polynomial or rational function, chosen so that its minimum is easy to determine. Since these methods do not always converge they should be combined with golden section search. There is a close analogy with robust methods for solving a nonlinear equation, where a combination of inverse interpolation and bisection can be used; see Section 6.2.4.

Since a linear function in general has no minimum the simplest choice is to use a second degree polynomial, i.e. a parabola. Suppose that at step n we have

three distinct points in u, v and w. The quadratic polynomial interpolating $g(x)$ at these points is (cf. (6.2.11))

$$p(x) = g(v) + (x - v)[u, v]g + (x - v)(x - u)[u, v, w]g.$$

Setting the derivative of $p(x)$ equal to zero gives

$$0 = [u, v]g + (v - u)[u, v, w]g + 2(x - v)[u, v, w]g.$$

and solving for x

$$x = v + d, \quad d = -\frac{[u, v]g + (v - u)[u, v, w]g}{2[u, v, w]g}. \tag{6.4.4}$$

This is a minimum point of $p(x)$ if $[u, v, w]g > 0$. We assume that of all the points where g has been evaluated v is the one with least function value. Therefore d should be small, so the effect of rounding errors in computing d is minimized. Initially we can take $u = a$, $w = b$, and if $g(c) < g(d)$ then $v = c$ otherwise $v = d$, where c and d are the two golden section points.

Multiplying the nominator and denominator of d by $(v - u)(w - v)(w - u)$, a short calculation shows that $d = -s_1/s_2$, where

$$\begin{aligned} r_1 &= (w - v)(g(v) - g(u)), \qquad r_2 = (v - u)(g(w) - g(v)), \\ s_1 &= (w - v)r_1 + (v - u)r_2, \qquad s_2 = 2(r_2 - r_1). \end{aligned} \tag{6.4.5}$$

Consider parabolic interpolation at the points x_{i-2}, x_{i-1}, x_i, $i = 2, 3, \ldots$ and let $\epsilon_i = x_i - \tau$. Assuming that $g(x)$ is sufficiently smooth in a neighborhood of τ it can be shown that asymptotically the relation

$$\epsilon_{i+1} \sim \frac{c_3}{2c_2}\epsilon_{i-1}\epsilon_{i-2}, \quad c_r = \frac{1}{k!}g^{(k)}(\zeta_r), \tag{6.4.6}$$

holds between successive errors. Hence the convergence order equals the real root $p = 1.3247\ldots$ of the equation $x^3 - x - 1 = 0$.

If two or more of the points u, v, w coincide, or if the parabola degenerates into a straight line, then $s_2 = 0$. The parabolic interpolation step is only taken if the following inequalities are true:

$$|d| < \tfrac{1}{2}|e|, \quad s_2 \neq 0, \quad v + d \in [a, b],$$

where e is the value of the second last cycle and, as before, tol is a combination of absolute and relative tolerance is used

$$\text{tol} = \epsilon|x| + \tau.$$

Otherwise a golden section step is taken, i.e.,

$$x = \begin{cases} (1 - t)v + ta, & \text{if } v \geq \frac{1}{2}(a + b); \\ (1 - t)v + tb, & \text{if } v < \frac{1}{2}(a + b), \end{cases}$$

where $1 - t = 2/(\sqrt{5} + 1)$.

The combination of inverse quadratic interpolation and golden section search has been suggested by Brent [2, 1973, Ch. 5],where the many delicate points to consider in an implementation are discussed. At a typical step there are six significant points a, b, u, v, w and x, not all distinct. The position of these points are updated at each step. Initially $[a, b]$ is an interval known to contain a local minimum point. At a later point in the algorithm they have the following significance: A local minimum lies in $[a, b]$; of all the points at which g has been evaluated v is the one with the least value of g; w is the point with the next lowest value of g; u is the previous value of w, and x is the last point at which g has been evaluated.

The Matlab function `fminbnd` is based on the Fortran implementation FMIN of Brent's algorithm given in Forsythe, Malcolm, and Moler [5, pp.184–187].

Review Questions

1. How many steps in needed in golden section search to reduce an initial interval $[a, b]$ by a factor of 10^{-6}?
2. Suppose the twice differentiable function $f(x)$ has a local minimum at a point x^*. What approximate limiting accuracy can you expect in a method for computing x^* which uses only function values?
3. The algorithm FMIN is a standard method for finding the minimum of a function. It uses a combination of two methods. Which?

Problems and Computer Exercises

1. Use the algorithm goldsec to find the minimum of the quadratic function $f(x) = (x - 1/2)^2$ starting from $a = 0.25$, $b = 1$. Plot the successive inclusion intervals.
2. Modify the algorithm goldsec to use parabolic interpolation instead of golden section if this gives a point within the interval.
3. (a) Plot the function

$$g(x) = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04},$$

and show that it has a local minimum in each of the intervals $[0.2, 0.4]$ and $[0.8, 1.0]$.

(b) Use your algorithm from Problem 2 to determine the location of the two minima of $g(x)$ in (a).

(c) MATLAB includes a function `fminbnd`, that also uses a combination of golden section search and parabolic interpolation to find a local minimum. Compare the result using this function with the result from (b).

4. (Brent [2, Sec. 5.6]) The function

$$g(x) = \sum_{i=1}^{20} \left(\frac{2i-5}{x-i^2} \right)^2,$$

has poles at $x = 1^2, 2^2, \ldots, 20^2$. Restricted to the open interval $(i^2, (i+1)^2)$, $i = 1:19$, it is unimodal. Determine the minimum points in these intervals and the corresponding values of $g(x)$.

6.5 Zeros of Polynomials

6.5.1 Introduction

The problem of solving a polynomial equation

$$p(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_n = 0, \quad (a_0 \neq 0), \tag{6.5.1}$$

has played a major role in the development of mathematical concepts for many centuries. Even the "high school formula" for solving a quadratic equations requires the introduction of irrational and complex numbers. There is a long history of investigations into algebraic expressions for the zeros of equations of higher degree. In the 16th century Cardano published formulas for the roots of a cubic equation (see Problem 2.3.8). Formulas for the roots when $n = 4$ are also known. In 1826 Abel proved that it is not possible to find algebraic expressions for the roots for the class of polynomial equations of degree $n > 4$. However, even the existing formulas for $n \leq 4$ are not in general suitable for numerical evaluation of the roots. In Section 2.3.2, it was shown that care must be taken to ensure numerical stability also for the simple formulas in the quadratic case (see also Problem 2)!

Despite the absence of closed solution formulas the **fundamental theorem of algebra** states for any algebraic equation $p(z) = 0$ of degree $n > 0$, there exists at least one complex number z_1 such that $p(z_1) = 0$. Hence we can write $p(z) = p_0(z) = (z - z_1)p_1(z)$, where $p_1(z)$ is a polynomial of degree $n-1$. Now $p_1(z)$ must have at least one root z_2, and we write $p_1(z) = (z - z_2)p_2(z)$, where $p_2(z)$ has degree $n-2$. Continuing this reasoning $p_{n-1}(z) = (z - z_n)p_n$, where $p_n = a_n$ is a constant. It follows that, counting multiplicities, the equation (6.5.1) has exactly n (real or complex) roots $z_1, z_2, \ldots, z_n$, and

$$p(z) = a_0(z - z_1)(z - z_2)\cdots(z - z_n). \tag{6.5.2}$$

By this representation it also follows that if the coefficients a_0, $a_1, \ldots, a_n$ are real, then eventual complex roots must occur in conjugate pairs.

Solving polynomial equations of high degree does not play a central role in scientific computing. Usually the applications involve only equations of moderate degree, say 10–20, for which acceptable subroutines exist. Polynomial equations of high degree play a major role also as a computational task in the area of computer algebra, where one need to solve (6.5.1) for $n > 100$. This in general requires high

multiple precision computations and algorithm for such problems are still a subject of research.

It should be emphasized that although a problem may be given in the form (6.5.1), it could be that the coefficients of $p(z)$ are not the original data. Then it may be better to avoid computing them. An important example of this is when the polynomial is the **characteristic polynomial**

$$p_(z) = \det(zI - A)$$

of a matrix $A \in \mathbf{A}^{n\times n}$. Then it is an eigenvalue problem in disguise, and the n roots of $p_A(z) = 0$ are the eigenvalues of A. Here the original data are the elements of the matrix A and numerical values of $p(z)$ can then in general be evaluated much more accurately directly from the matrix elements, see Chapter 9. Even when the eigenvalues are well determined by the elements of A and appear to be well separated, they can be *extraordinary sensitive to to small relative perturbations in the coefficients coefficients of* $p_A(z)$. In the following we discuss a famous example, due to Wilkinson [28, 1984]. This paper[7] contains an extensive discussion of numerical problems in determining roots of polynomial equations.

Example 6.5.1.

Consider the Wilkinson polynomial

$$p(z) = (z-1)(z-2)\cdots(z-20) = z^{20} - 210z^{19} + \ldots + 20!,$$

with zeros $1, 2, \ldots, 20$. Let $\bar{p}(z)$ be the polynomial which is obtained when the coefficient $a_1 = 210$ in $p(z)$ is replaced by

$$-(210 + 2^{-23}) = -210.000000119\ldots,$$

while the rest of the coefficients remain unchanged. Even though the relative perturbation in a_1 is of order 10^{-10}, many of the zeros of the perturbed polynomial $\bar{p}(z)$ deviate greatly from those of $p(z)$. In fact, correct to nine decimal places, the perturbed zeroes are

1.000000000	$10.095266145 \pm 0.643500904i$
2.000000000	
3.000000000	$11.793633881 \pm 1.652329728i$
4.000000000	
4.999999928	$13.992358137 \pm 2.518830070i$
6.000006944	
6.999697234	$16.730737466 \pm 2.812624894i$
8.007267603	
8.917250249	$19.502439400 \pm 1.940330347i$
20.846908101	

For example, the two zeros $16, 17$ have not only changed substantially, but have become a complex pair. It should be emphasized that this behavior is quite typical

[7] Wilkinson received the Chauvenet Prize of the Mathematical Association of America for this exposition of the ill-conditioning of polynomial zeros.

of polynomials with real coefficients and real roots. Indeed, many polynomials which arise in practice behave much worse than this.

If we assume that the coefficients a_i of a polynomial are given with full machine accuracy, then the error δ in computed values of $p(x)$ (for real x) is bounded by

$$\delta < 1.06u \sum_{i=0}^{n} |(2i+1)a_{n-i}x^i| < \gamma_{2n+1} \sum_{i=0}^{n} |a_{n-i}||x|^i,$$

see Section 2.4. Hence by (6.1.7) the attainable accuracy of a zero α is equal to

$$\epsilon_\alpha = \frac{\delta}{|p'(\alpha)|} = \frac{\sum_{i=0}^{n} |(2i+1)a_{n-i}\alpha^i|}{|p'(\alpha)|}.$$

In particular for the root $\alpha = 14$ in the above example we get $\epsilon_\alpha = 1.89 \cdot 10^{16}$. However, the changes in this example are so large that this linearized perturbation theory does not apply!

As the above example emphasizes, *computing the characteristic polynomial is not, as is sometimes thought, a simplification of the eigenvalue problem.* Eigenvalue problems should be solved with one of the highly developed modern eigenvalue algorithms; consult Chapter 9, Volume II, and references therein! Note also that if the coefficients of the characteristic polynomial $\det(A - zI)$ are required, these are best computed by first computing the eigenvalues λ_i of A are computed and then forming

$$p(z) = \prod_{i=1}^{n} (z - \lambda_i). \tag{6.5.3}$$

The **companion matrix** of the polynomial $p(z)$ in (6.5.1), normalized so that $a_0 = 1$, is defined as

$$C = \begin{pmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}. \tag{6.5.4}$$

(Sometimes the companion matrix is defined slightly differently, e.g., with the coefficients of the polynomial in the last row or in the last column.) Using the definition (1.6.4) it can be verified that the characteristic polynomial of C equals

$$p_C(z) = \det(zI - C) = z^n + a_1 z^{n-1} + \cdots + a_{n-1}z + a_n.$$

Hence the the eigenvalues of C are the roots of $p(z) = 0$ and turning the tables, algorithms for solving eigenvalue problems can be used for solving polynomial equations. In MATLAB the function `roots(p)` computes the roots of a polynomial $p(z)$ using the QR algorithm to solve the eigenvalue problem for the companion matrix. Although the operation count for this QR algorithm is $\mathcal{O}(n^3)$ and the storage requirement $1.5n^2$ experiments suggest that for small and moderate values of n it is as

as fast as competing algorithms and can be more accurate. Further problems with overflow or underflow are avoided. However, it seems less suitable for applications where n is large and the roots are wanted to multiple precision accuracy.

If the coefficients are known (and stored) exactly, then by using multiple-precision arithmetic the accuracy in the zeros can be increased. It is generally true that the solution of polynomial equations of high degree requires the use of multiple precision floating-point arithmetic in order to achieve high accuracy.

Example 6.5.2.

The largest positive root of the equation

$$p(x) = (x+2)(x^2-1)^6 - 3 \cdot 10^{-6} \cdot x^{11} = 0$$

is to be computed. Here $p(z)$ is a polynomial of degree 13. If the coefficients are computed using decimal floating point arithmetic with seven digits, then the coefficient of x^{11} which is $(12 - 3 \cdot 10^{-6})$ will be rounded to 12.00000. Thus the machine will treat the equation $(x+2)(x^2-1)^6 = 0$, whose exact positive root is 1.

This is a poor result. One can get the root $\alpha = 1.053416973823$ to full accuracy for example by writing the equation in the form

$$x = \phi(x), \qquad \phi(x) = 1 + \frac{0.1}{x+1}\left(\frac{3x^{11}}{x+2}\right)^{1/6},$$

and solving this by the iteration $x_0 = 1$, $x_{k+1} = \phi(x_k)$. Hence the relative error in the previous result is greater then 5%.

6.5.2 Some Basic Formulas

Comparing the coefficients of z^{n-k} in the representations (6.5.1) and (6.5.2) of $p(z)$ we find that $(-1)^k a_k/a_0$ is the sum of the $\binom{n}{k}$ products of the roots taken k at a time. Thus we obtain the following relations between the coefficients and zeros of a polynomial

$$\sum_i z_i = -a_1/a_0, \quad \sum_{i<j} z_i z_j = a_2/a_0, \quad \sum_{i<j<k} z_i z_j z_k = a_3/a_0, \cdots$$
$$\cdots z_1 z_2 \cdots z_n = (-1)^n a_n/a_0. \tag{6.5.5}$$

The functions on the left side are called **elementary symmetric functions** of the variables $z_1, z_2, \ldots, z_n$, since interchanging any of the variable will not change the functions.

If $a_n \neq 0$ then the **reciprocal polynomial** is

$$q(y) = y^n p(1/y) = a_n y^n + \cdots + a_1 y + a_0. \tag{6.5.6}$$

The zeros of the reciprocal polynomial are $1/z_1, 1/z_2, \ldots, 1/z_n$ and from (6.5.5) we have the relations,

$$\sum_i 1/z_i = -a_{n-1}/a_n, \quad \text{etc..}$$

Function values of the polynomial $p(z)$ in (6.5.1) at a (real or complex) point w can conveniently be computed by repeated **synthetic division** of $p(z)$ with $z-w$; cf. Section 1.3.1. Let

$$\begin{aligned} p(z) &= (z-w)q(z) + b_n, \\ q(z) &= b_0 z^{n-1} + b_1 z^{n-2} + \ldots + b_{n-1}. \end{aligned} \tag{6.5.7}$$

Then the sequence $\{b_i\}_{i=0}^n$ is defined by the recursion

$$b_0 = a_0, \quad b_i = b_{i-1}w + a_i, \quad i = 1:n. \tag{6.5.8}$$

Setting $z = w$ we we see that $p(w) = b_n$ is the remainder and $q(z)$ is the quotient polynomial when dividing $p(z)$ with $(z-w)$. Differentiating (6.5.7) we get

$$p'(z) = (z-w)q'(z) + q(z), \tag{6.5.9}$$

and setting $z = w$ we find that $p'(w) = q(w)$. We can evaluate $q(w)$ by synthetic division of $q(z)$ with $(z-w)$,

$$\begin{aligned} q(z) &= (z-w)r(z) + c_{n-1}, \\ r(z) &= c_0 z^{n-2} + c_1 z^{n-3} + \ldots + c_{n-2}. \end{aligned}$$

Now $p'(w) = q(w) = c_{n-1}$, where

$$c_0 = b_0, \qquad c_i = c_{i-1}w + b_i, \quad i = 1:n-1.$$

Higher derivatives can be computed in the same fashion. Differentiating once more gives

$$p''(z) = (z-w)q''(z) + 2q'(z),$$

and so $\frac{1}{2}p''(w) = q'(w) = d_{n-2}$, where

$$d_0 = c_0, \qquad d_i = d_{i-1}w + c_i, \quad i = 1:n-2.$$

To compute $p^{(i)}(w)$ using these formulas requires $n-i$ additions and multiplications.

In the important special case where all the coefficients $a_0, a_1, \ldots, a_n$ are real, the above formulas are somewhat inefficient, and one can save operations by performing synthetic division with the quadratic factor

$$(z-w)(z-\bar{w}) = z^2 - 2zRe(w) + |w|^2,$$

which has real coefficients (see Problem 1).

Synthetic division can also be used to shift the origin of a polynomial $p(z)$. Given $a_0, a_1, \ldots, a_n$ and s, we then want to find coefficients $c_0, c_1, \ldots, c_n$ so that

$$p(w+s) = q(s) = c_0 s^n + c_1 s^{n-1} + \cdots + c_n. \tag{6.5.10}$$

Clearly this is the Taylor expansion of $p(z)$ at $z = w$. It follows that

$$c_n = p(w), \quad c_{n-1} = p'(w), \quad c_{n-1} = \frac{1}{2}p''(w), \quad \ldots, \quad c_0 = \frac{1}{n!}p^{(n)}(w),$$

and the coefficients c_i can be computed by repeated synthetic division of $p(z)$ by $(z-w)$ as described above in about $n^2/2$ multiplications.

It is often desirable to obtain some preliminary information as to where the zeros of a polynomial $p(z)$ are located. Some information about the location of real roots can be obtained from a simple examination of the sign of the coefficients of the polynomial. A simple observation is that if $a_i > 0$, $i = 1:n$, then $p(x)$ can have no real positive zero. A generalization of this result, known as **Descartes' rule of sign**, [8] states that the number of positive roots is either given by the number of variations in sign in the sequence $a_0, a_1, \ldots, a_n$, or is *less* than that by an even number. (Multiple roots are counted with their multiplicity.) By considering the sign variations for the polynomial $p(-z)$ we similarly get an upper bound on the number of negative roots. By shifting the origin, we can get bounds on the number of roots larger and smaller than a given number. In Sec. 6.5.3 we give a method to obtain precise information about the number of real roots in any given interval.

Many classical results are known about the number of real or complex roots in a disk or half plane. It is outside the scope of this presentation and we refer to surveys in the literature.

6.5.3 Sturm Sequences

Precise information about the number of real roots in an interval can be obtained from a **Sturm sequence**[9] for $p(x)$.

Definition 6.5.1.

A sequence of real polynomials $p_0(x), p_1(x), \ldots, p_m(x)$ is a strict Sturm sequence for $p(x) = p_0(x)$ on the interval $[a, b]$ if the following conditions hold:

(i) *No two consecutive polynomials in the sequence vanish simultaneously on the interval $[a, b]$.*

(ii) *If $p_j(r) = 0$ for $j < m$, then $p_{j-1}(r)p_{j+1}(r) < 0$.*

(iii) *Throughout the interval $[a, b]$, $p_m(x) \neq 0$.*

(iv) *If $p_0(r) = 0$, then $p_0'(r)p_1(r) > 0$.*

Given a polynomial $p_1(x)$ of degree not greater than that of $p_0(x)$ a Sturm sequence can be constructed by the Euclidean algorithm as follows. Let $q_1(x)$ be the quotient polynomial and $-p_2(x)$ the remainder in the quotient $p_0(x)/p_1(x)$, i.e. $p_0(x) = q_1(x)p_1(x) - p_2(x)$, where the degree of $p_2(x)$ is strictly less than that of $p_1(x)$. Continuing in this way, we compute $p_2(x), \ldots, p_m(x)$ by

$$p_{k+1}(x) = q_k(x)p_k(x) - p_{k-1}(x), \quad k = 1:m-1, \tag{6.5.11}$$

[8] René Descartes (1596–1650) French philosopher and mathematician.

[9] J. C. F. Sturm (1803–1855) a Swiss mathematician best known for his theorem on Sturm sequences, discovered in 1829 and his theory of Sturm–Liouville differential equations. He succeeded Poisson in the chair of mechanics at the École Polytechnique in Paris 1839.

where $q_k(x)$ is the quotient and $-p_{k+1}$ the remainder in the quotient $p_{k-1}(x)/p_k(x)$. We stop when $p_m(x)$ nowhere vanishes on the interval $[a, b]$. Clearly, if $p_i(r) = 0$, then $p_{j+1}(r) = -p_{j-1}(r) < 0$, so condition (ii) is satisfied.

Let $V(x)$ denote the number of variations in sign in the Sturm sequence at x. If $p_0(x)$ and $p_1(x)$ have only simple zeros that separate each other, then it can be shown that the number of zeros of $p_0(x)$ on $[a, b]$ is equal to $|V(a) - V(b)|$.

Theorem 6.5.2.

Take $p_1(x) = p_0'(x)$ and define $p_2(x), \ldots, p_m(x)$ by (6.5.11), where $p_m(x)$ has a fixed sign on the interval $[a, b]$ ($p_0(a) \neq 0$ and $p_0(b) \neq 0$). Let V(r) denote the number of variations of sign in the sequence of values $p_0(r), p_1(r), \ldots, p_m(r)$, vanishing terms not being counted. Then the number of roots of $p_0(x)$ in $[a, b]$, each multiple root being counted once, is exactly equal to $|V(a) - V(b)|$.

Note that if all real zeros of $p_0(x)$ are simple and $p_1(x) = p_0'(x)$, then (6.5.11) generates a Sturm sequence. If $p_0(x)$ has multiple zeros, then $p_0(x)$ and $p_0'(x)$ have a common divisor, which divides every $p_i(x)$ in the sequence, and this will not affect $V(r)$.

Example 6.5.3.

The equation $p(x) = p_0 = x^5 - 3x - 1 = 0$ has three real roots $z_1 = -1.21465$, $z_2 = -0.33473$, and $z_3 = 1.38879$ and two complex roots. The derivative equals $p'(x) = p_1 = 5x^4 - 3$, and the rest of the Sturm chain is given by

$$p_2 = \frac{12}{5}x + 1, \quad p_3 = \frac{59083}{20736}.$$

Here p_2 is a polynomial of degree one and the Sturm chain ends with $s = 3 < n$.

We denote by $[l_k, u_k]$ an interval containing the zero x_k. Evaluating the sign changes of the Sturm sequence at $x = -2$ and $x = 2$ shows that there are $3 - 0 = 3$ roots x_k, $k = 1, 2, 3$, in the interval $[-2, 2]$ Counting the number of sign changes at the midpoint $x = 0$ allows us to deduce that $u_k = 0$, $k = 1, 2$ and $l_3 = 0$; see Table 6.5.1. The interval $[-2, 0]$ contains two roots so we determine next the number of sign changes at the midpoint $x = -1$.

At this point we have determined three disjoint intervals $[-2, -1]$, $[-1, 0]$, and $[0, 2]$, which each contain one root. We continue bisecting each of these intervals, which can be performed in parallel.

Methods based on Sturm sequences can can be competitive, when only a relatively small number of real roots in an given interval are of interest. Consider a real symmetric tridiagonal matrix,

$$A = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix},$$

Table 6.5.1. *Left: Sign variations in the Sturm sequence. Right: Intervals* $[l_k, u_k]$ *containing the zero* x_k*.*

x	p_0	p_1	p_2	p_3	δ
-2	$-$	$+$	$-$	$+$	3
$+2$	$+$	$+$	$+$	$+$	0
0	$-$	$-$	$+$	$+$	1
-1	$+$	$+$	$-$	$+$	2
1	$-$	$+$	$+$	$+$	1

l_1	u_1	l_2	u_2	l_3	u_3
-2	2	-2	2	-2	2
	0		0	0	
	-1	-1			
				1	

such that $\beta_k \neq 0$, $k = 2 : n$ has only simple eigenvalues. Let $p_k(\lambda)$ be the characteristic polynomial of the kth leading principal minor of $(A - \lambda I)$. Define $p_0(\lambda) = 1$, and $p_k(\lambda)$ by the three-term recursion

$$p_1(\lambda) = \alpha_1 - \lambda, \quad p_k(\lambda) = (\alpha_k - \lambda)p_{k-1}(\lambda) - \beta_k^2 p_{k-2}(\lambda), \quad k = 2 : n. \tag{6.5.12}$$

Then the sequence

$$1, p_0(\lambda), \ldots, p_n(\lambda) = \det(A - \lambda I)$$

is known to form a Sturm sequence. Combined with the bisection method, this recursion can be used to develop an efficient numerical method for determining the eigenvalues of a symmetric tridiagonal matrix A in a given interval $[a, b]$ without reference to any of the others; see Vol. II, Sec. 9.6.6. It can also be used for determining the singular values of a bidiagonal matrix in a given interval; see Vol. II, Sec. 9.7.7.

The Sturm sequence algorithm only works when $f(x)$ is a real function of a real variable. To determine complex zeros an algorithm that performs a search and exclusion of the complex plane can be used. The **quadtree** exclusion algorithm, due to H. Weyl [27], and illustrated in Figure 6.5.1, is such a "two-dimensional bisection algorithm".[10] It was one of the first algorithms with guaranteed convergence to all n zeros of a polynomial of degree n. The algorithm is based on a exclusion test applied to squares in the complex plane. For example, assuming that $f(z)$ is analytic in K then if $|f'(z)| \leq M$ for all $z \in K$ and $|f(z_0)| > \eta M$ there is no zero of $f(z)$ in K. Any square that does not pass the test and thus may contain a root is called suspect. (Note that it is not required that a suspect square actually contains a root.)

The computations begin with an initial suspect square S containing all the zeros of $p(x)$. This square can be found from an upper bound on the absolute value of the zeros of $p(x)$. In the algorithm, as soon as we have a suspect square, this is partitioned into four congruent subsquares. At the center of each of them a test estimating the distance to the closest zero of $p(x)$ is performed. (A relative error within, say, 40% will suffice.) If the test guarantees that this distance exceeds half of the length of the diagonal of the square then the square cannot contain any

[10] In general a quadtree is a tree where each node is split along d dimensions giving 2^d children.

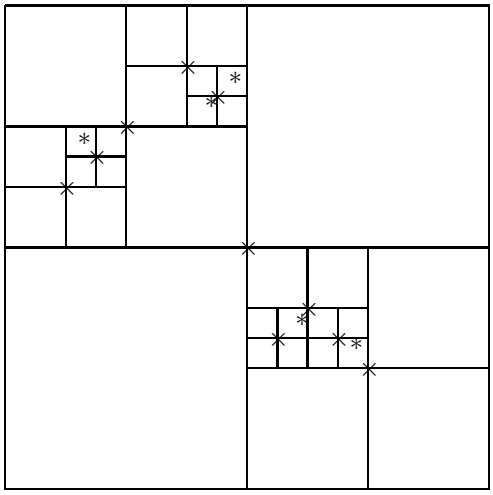


Figure 6.5.1. *Suspect squares computed by Weyl's quadtree method. Their centers (marked by ×) approximate the five zeros marked by ∗.*

zero and is discarded. Each remaining suspect square undergoes the same recursive partitioning into four subsquares and the test. The zeros lying in a suspect square are approximated by its center with errors bounded by half the length of its diagonal. Each iteration step decreases the diagonal of the remaining squares by a factor of two so the errors will decrease by a factor of 1/2.

6.5.4 Deflation and Zero Suppression

Suppose we have found a root α to the equation $p(z) = 0$. Then taking $z_k = \alpha$ in (6.5.8)–(6.5.7) we have $b_n = p(\alpha) = 0$ and the remaining roots of $p(z)$ are also roots of the polynomial equation

$$q(z) = \frac{p(z)}{z - \alpha} = 0$$

Hence we can continue the iterations with the quotient polynomial $q(z)$ of degree $n - 1$. This process is called **deflation** and can be repeated; as soon as a root has been found it is factored out. Proceeding like this, all roots are eventually found. Since we work with polynomials of lower and lower degree, deflation saves arithmetic operations. More important is that it prevents the iterations to converge to the same simple root more than once.

So far we have ignored that roots which are factored out are only known with finite accuracy. Also rounding errors occur in the computation of the coefficients of the quotient polynomial $q(x)$. Clearly there is a risk that both these types of errors can have the effect that the zeros of the successive quotient polynomials deviate more and more from those of $p(z)$. Indeed, deflation is not unconditionally a stable numerical process. A closer analysis performed by Wilkinson [28, 1963][11] shows

[11] James Hardy Wilkinson, English mathematician 1919–1986. From 1946 Wilkinson he worked

that if the coefficients of the quotient polynomials are computed by the recursion (6.5.8), then errors resulting from deflation are negligible provided that:

1. the roots are determined in order of *increasing* magnitude;

2. each root is determined to its limiting accuracy.

Note that if the above procedure is applied to the reciprocal polynomial $z^n p(1/z)$ we obtain the zeros of $p(z)$ in order of *decreasing* magnitude.

With Laguerre's method it is quite probable that we get convergence to the root of smallest magnitude from the initial value $z_0 = 0$. However, this cannot be guaranteed and therefore one often proceeds in two steps. First, all n roots are determined using deflation in the process. Next, each root found in the first step is refined by doing one or several Newton iterations using the *original* polynomial $p(z)$.

Deflation can be avoided by using a **zero suppression** technique suggested by Maehly [1954]. He notes that the derivative of the reduced polynomial $q(z) = p(z)/(z - \xi_1)$ can be expressed as

$$q'(z) = \frac{p'(z)}{z - \xi_1} - \frac{p(z)}{(z - \xi_1)^2}.$$

More generally, assume that we have determined approximations $\xi_1, \ldots, \xi_j$ to j roots of $p(z) = 0$. Then the the first derivative of the reduced polynomial $q_j(z) = p(z)/[(z - \xi_1) \cdots (z - \xi_j)]$ can be expressed as

$$q_j'(z) = \frac{p'(z)}{(z - \xi_1) \cdots (z - \xi_j)} - \frac{p(z)}{(z - \xi_1) \cdots (z - \xi_j)} \sum_{i=1}^{j} \frac{1}{z - \xi_i}.$$

Hence Newton's method applied to $q_j(z)$ can be written

$$z_{k+1} = z_k - \frac{p(z_k)}{p'(z_k) - \sum_{i=1}^{j} p(z_k)/(z_k - \xi_i)}, \tag{6.5.13}$$

which is the **Newton–Maehly method**. This iteration has the advantage that it is not sensitive to the accuracy in the approximations to the previous roots $\xi_1, \ldots, \xi_j$. Indeed, the iteration (6.5.13) is locally quadratically convergent to simple zeros of $p(z)$ for *arbitrary* values of $\xi_1, \ldots, \xi_j$.

6.5.5 Simultaneous Determination of Roots

For the removal of a linear factor by deflation it is necessary that the zero has been computed to full working accuracy, since otherwise the remaining approximative zeros can be meaningless. This is a disadvantage if only low accuracy is required.

on the group that built the Pilot ACE computer at National Physical Laboratory, first as Turing's assistant and later as manager of the group. He later contributed greatly to the development of reliable software for matrix computations.

An alternative to deflation is to use an iterative methods that, under appropriate separation assumptions, allows for the *simultaneous* determination of all the roots of a polynomial equation. Suppose that the numbers $\xi_i^{(k)}$, $i = 1 : n$ are a set of n distinct approximations to the of $p(z)$. A new set of approximations are then computed from

$$\xi_i^{(k+1)} = \xi_i^{(k)} - p(\xi_i^{(k)}) \Big/ \Big[a_0 \prod_{\substack{j=1 \\ j \neq i}}^{n} (\xi_i^{(k)} - \xi_j^{(k)}) \Big], \quad i = 1 : n. \tag{6.5.14}$$

This is **Weierstrass' method**,[12] introduced in 1891 in connection with a new constructive proof of the fundamental theorem of algebra. The method was rediscovered and analyzed in the 1960s by Durand and is also known as the **Durand–Kerner method**.

With $q(z) = (z - \xi_1^{(k)})(z - \xi_2^{(k)}) \cdots (z - \xi_n^{(k)})$ the formula may also be written

$$\xi_i^{(k+1)} = \xi_i^{(k)} - p(\xi_i^{(k)})/q'(\xi_i^{(k)}),$$

which shows that to first approximation the method is identical to Newton's method. This relation can be used to prove that for simple (real or complex) zeros the asymptotic order of convergence of the Weierstrass method equals 2. (For multiple zeros the method will only converge linearly.) The relation

$$\sum_{i=1}^{n} \xi_i^{(k)} = \sum_{i=1}^{n} \alpha_i = -a_1, \quad k \geq 1,$$

which holds independent of the initial approximations, can be used as a control; see Kjellberg [12].

It is possible to accelerate Weierstrass method by using the new approximations of the roots in (6.5.14) as they become available. This leads to the iteration

$$\xi_i^{(k+1)} = \xi_i^{(k)} - p(\xi_i^{(k)}) \Big/ \Big[a_0 \prod_{j<i} (\xi_i^{(k)} - \xi_j^{(k+1)}) \prod_{j>i} (\xi_i^{(k)} - \xi_j^{(k)}) \Big], \quad i = 1 : n.$$

This *serial* version of the Weierstrass method can be shown to have an order of convergence at least $1 + \sigma_n$, where $1 < \sigma_n < 2$ is the unique positive root to $\sigma^n - \sigma - 1 = 0$.

If no a priori information about the roots is available then the initial approximations $\xi_i^{(0)}$ can be chosen equidistantly on a circle $|z| = \rho$, centered at the origin, which encloses all the zeros of $p(z)$. Such a circle can be found, e.g., by using the result that all the roots of the polynomial $p(z)$ lie in the disk $|z| \leq \rho$, where

$$\rho = \max_{1 \leq k \leq n} 2 \left(\frac{|a_k|}{|a_0|} \right)^{1/k}.$$

Note that this is (6.5.15) applied to the reciprocal polynomial.

[12] Karl Theodor Wilhelm Weierstrass (1815–1897) influential German mathematician, often said to be the father of modern analysis.

6.5.6 A Modified Newton Method

Most of the iteration methods described in earlier sections can be applied to polynomial root finding. Note that if $p(z)$ has real coefficients, then $p(z)$ and $p'(z)$ are real for real values of z. This means that Newton's method cannot converge to a *complex* root from a *real* initial approximation. The same holds for the secant method and its variants. The Muller–Traub method (see Section 6.2.3) can also converge to complex roots from real approximations. It requires only one evaluation of $p(z)$ per step, and has therefore become popular.

If we have sufficiently good initial approximations to a (real or complex) zero of $p(z)$, this can be computed by Newton's method. It is desirable to find the zeros in roughly increasing order of magnitude, since this leads to stable deflation; see Sec. 6.5.4. We now describe a modified Newton method due to Madsen [14], which has been shown to be very competitive. By including a one-dimensional search along the Newton direction this method achieves good global convergence properties and is effective also for multiple roots.

To initialize let $z_0 = 0$,

$$\delta z_0 = \begin{cases} -p(0)/p'(0) = -a_n/a_{n-1}, & \text{if } a_{n-1} \neq 0 \\ 1 & \text{otherwise,} \end{cases}$$

and take

$$z_1 = \frac{1}{2\rho}\frac{\delta z_0}{|\delta z_0|}, \qquad \rho = \max_{1\le k\le n}\left(\frac{|a_{n-k}|}{|a_n|}\right)^{1/k}. \tag{6.5.15}$$

This assures that $|z_1|$ is less than the modulus of any zero of $p(z)$ (see [10, Exercise 2.2.11]). Further, if $p'(0) \neq 0$, it is in the direction of steepest descent of $|p(z)|$ from the origin (see Sec. 6.3.2). This choice makes it likely that convergence will take place to a root of near minimal modulus.

The general idea of the algorithm is that given z_k, a tentative step h_k is computed by Newton's method. The next iterate is found by taking the best point (in terms of minimizing $|f(z)|$) found by a short search along the line through z_k and $z_k + h_k$. When the search yields no better value than at z_k we take $z_{k+1} = z_k$ and make sure that the next search is shorter and in a different direction. Since the line searches will be wasteful if we are near a simple root, we then switch to the standard Newton's method.

In the first stage of the algorithm, when searches are being performed, new iterates z_{k+1} are computed as follows:

1. If the last iteration was successful ($z_k \neq z_{k-1}$) then the Newton correction

$$h_k = -p(z_k)/p'(z_k), \tag{6.5.16}$$

is computed and the next tentative step is taken as

$$\delta z_k = \begin{cases} h_k, & \text{if } |h_k| \le 3|z_k - z_{k-1}|; \\ 3|z_k - z_{k-1}|e^{i\theta}h_k/|h_k| & \text{otherwise.} \end{cases}$$

Here θ is chosen rather arbitrarily as $\arctan(3/4)$. This change of direction is included because if a saddle point is being approached, the direction h_k may be a bad choice.

2. If the last step was unsuccessful ($z_k = z_{k-1}$) we change the search direction and reduce the step size. In this case the tentative step is chosen to be

$$\delta z_k = -\tfrac{1}{2} e^{i\theta} \delta z_{k-1}.$$

Repeated use of this is sure to yield a good search direction.

3. Once the tentative step δz_k has been found we test the inequality $|p(z_k + \delta z_k)| < |p(z_k)|$. If this is satisfied we calculate the numbers

$$|p(z_k + p\,\delta z_k)|, \quad p = 1, 2, \ldots, n,$$

as long as these are strictly decreasing. Note that, if we are close to a multiple root of multiplicity m, then we will find the estimate $z_k + mh_k$, which gives quadratic convergence to this root. A similar situation will hold if we are at a fair distance from a cluster of m zeros and other zeros are further away.

If $|p(z_k + \delta z_k)| \geq |p(z_k)|$, we calculate the numbers

$$|p(z_k + 2^{-p}\delta z_k)|, \quad p = 0, 1, 2,$$

again continuing until the sequence ceases to decrease.

A switch to standard Newton is made if in the previous iteration a standard Newton step $z_{k+1} = z_k + h_k$ was taken and Theorem 6.3.2 ensures the convergence of Newton's method with initial value z_{k+1}, i.e., when $f(z_k)f'(z_k) \neq 0$ and

$$2\,|f(z_k)| \max_{z \in K_k} |f''(z)| \leq |f'(z_k)|^2, \quad K_k : |z - z_k| \leq |h_k|,$$

is satisfied, cf. (6.3.12). This inequality can be approximated using already computed quantities by

$$2\,|f(z_k)||f'(z_k) - f'(z_{k-1})| \leq |f'(z_k)|^2 |z_{k-1} - z_k|. \tag{6.5.17}$$

The iterations are terminated and z_{k+1} accepted as a root whenever $z_{k+1} \neq z_k$ and

$$|z_{k+1} - z_k| < u|z_k|,$$

holds, where u is the unit roundoff. The iterations are also terminated if

$$|p(z_{k+1})| = |p(z_k)| < 16nu|a_n|,$$

where the right hand side is a generous overestimate of the final roundoff made in computing $p(z)$ at the root of the smallest magnitude. The polynomial is then deflated as described in the previous section.

More details about this algorithm and methods for computing error bounds can be found in [14] and [15].

6.5.7 Laguerre's Method

In **Laguerre's method**[13] the polynomial $p(z)$ of degree n is approximated in the neighborhood of the point z_k by a special polynomial of the form

$$r(z) = a(z - w_1)(z - w_2)^{n-1},$$

where the parameters a, w_1 and w_2 are determined so that

$$p(z_k) = r(z_k), \qquad p'(z_k) = r'(z_k), \qquad p''(z_k) = r''(z_k). \tag{6.5.18}$$

If z_k is an approximation to a simple zero, α then the simple zero w_1 of $r(z)$ is taken as the new approximation z_{k+1} of α. Laguerre's method has very good global convergence properties for polynomial equations, and with *cubic* convergence for simple roots (real or complex). For multiple roots convergence is only linear.

In order to derive Laguerre's method we note that the logarithmic derivative of $p(z) = (z - \alpha_1) \cdots (z - \alpha_n)$ is

$$S_1(z) = \frac{p'(z)}{p(z)} = \sum_{i=1}^{n} \frac{1}{z - \alpha_i}.$$

Taking the derivative of this expression we obtain

$$-\frac{dS_1(z)}{dz} = S_2(z) = \left(\frac{p'(z)}{p(z)}\right)^2 - \frac{p''(z)}{p(z)} = \sum_{i=1}^{n} \frac{1}{(z - \alpha_i)^2}.$$

Using (6.5.18) to determine the parameters of the approximating polynomial r(z) we obtain the equations

$$S_1(z_k) = \frac{1}{z_k - w_1} + \frac{(n-1)}{z_k - w_2}, \qquad S_2(z_k) = \frac{1}{(z_k - w_1)^2} + \frac{(n-1)}{(z_k - w_2)^2}.$$

Eliminating $z_k - w_2$ gives a quadratic equation for the correction $z_k - w_1 = z_k - z_{k+1}$. After some algebra we obtain (check this!)

$$z_{k+1} = z_k - \frac{np(z_k)}{p'(z_k) \pm \sqrt{H(z_k)}}, \tag{6.5.19}$$

where

$$H(z_k) = (n-1)^2[p'(z_k)]^2 - n(n-1)p(z_k)p''(z_k).$$

The sign in the denominator in (6.5.19) should be chosen so that the magnitude of the correction $|z_{k+1} - z_k|$ becomes as small as possible.

For polynomial equations with only real roots, Laguerre's method is globally convergent, i.e., it converges for *every* choice of real initial estimate z_0. Suppose the roots are ordered such that $\alpha_1 \le \alpha_2 \le \cdots \le \alpha_n$. If $z_0 \in (\alpha_{j-1}, \alpha_j)$, $j = 2 : n$, then

[13] Edmund Nicolas Laguerre, 1834–1886, French mathematician at École Polytechnique, Paris and best known for his work on orthogonal polynomials.

Laguerre's method converges to one of the roots α_{j-1}, α_j; if $z_0 < \alpha_1$ or $z_0 > \alpha_n$ then convergence is to α_1 or α_n respectively.

For polynomial equations with complex roots, Laguerre's method no longer converges for every choice of initial estimate. However, experience has shown that the global convergence properties are good also in this case. In particular, if we take $z_0 = 0$, then Laguerre's method will usually converge to the root of smallest modulus. We finally remark that, as might be expected, for multiple roots convergence of Laguerre's method is only linear.

Consider the polynomial equation $p(z) = 0$ and assume that $a_n \neq 0$ so that $\alpha = 0$ is not a root. Now suppose that $a_{n-2}a_{n-1} \neq 0$, and take $z_0 = 0$ in Laguerre's method. A simple calculation gives

$$z_1 = \frac{-na_n}{a_{n-1} \pm \sqrt{H(z_0)}}, \qquad H(z_0) = (n-1)^2 a_{n-1}^2 - 2n(n-1)a_n a_{n-2}, \quad (6.5.20)$$

where the sign is to be chosen so the the $|z_1|$ is minimized. In particular, for $n = 2$, $H(z_0)$ is the discriminant of $p(z)$ and z_1 is the root of smallest modulus.

Example 6.5.4.

If there are complex roots, then there may be several distinct roots of smallest modulus. For example, the equation

$$p(z) = z^3 - 2z^2 + z - 2,$$

has roots $\pm i$ and 2. Using the above formula (6.5.20) for z_1 with $n = 3$, we get

$$z_1 = \frac{6}{1 \pm 2i\sqrt{11}} = \frac{2}{15} \pm i\frac{4\sqrt{11}}{15} = 0.06666666667 \pm 0.88443327743i.$$

Continuing the iterations with Newton's method we get convergence to one of the two roots $\pm i$,

$$z_2 = -0.00849761051 + 1.01435422762i,\ z_3 = -0.00011503062 + 1.00018804502i$$
$$z_4 = -0.00000002143 + 1.00000003279i,\ z_5 = -0.00000000000 + 1.00000000000i$$

Review Questions

1. Describe the method of iterated successive synthetic division for computing function values and derivatives of a polynomial.
2. Consider the polynomial $p(z) = z^4 - 2z^3 - 4z^2 + z + 1$. Using Descartes' rule of sign what can you deduce about the number of real positive roots?
3. Suppose that all roots of a polynomial equation are to be determined. Describe two methods to avoid the problem of repeatedly converging to roots already found.

4. Discuss the ill-conditioning of roots of polynomial equations. What famous polynomial did J. H. Wilkinson use as an example?

5. (a) What is the companion matrix of a polynomial $p(x) = x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$?

(b) One approach to computing the eigenvalues of a matrix A is to find the coefficients of the characteristic polynomial $p_A(\lambda) = \det(\lambda I - A)$, and then solve the algebraic equation $p_A(\lambda) = 0$. Why should such a method usually be avoided?

6. What properties are satisfied by a Sturm sequence of real polynomials $p_0(x)$, $p_1(x), \ldots, p_m(x)$? Describe one way of generating a Sturm sequence using the Euclidian algorithm.

Problems and Computer Exercises

1. Apply Newton's method to determine one of the complex roots of the equation $z^2 + 1 = 0$. Start with $z_0 = 1 + i$.

2. Consider a polynomial with real coefficients

$$p(z) = a_0z^n + a_1z^{n-1} + \cdots + a_n, \quad a_i \neq 0, \quad i = 0 : n.$$

(a) Count the number of (real) additions and multiplications needed to compute a value $p(z_0)$ by synthetic division of $p(z)$ by $(z - z_0)$, when z_0 is a real and complex number, respectively.

(b) For a complex number $z_0 = x_0 + iy_0$, $p(z_0)$ can also be computed by performing the synthetic division of p(z) with the real quadratic factor

$$d(z) = (z - z_0)(z - \bar{z}_0) = z^2 - 2x_0z + (x_0^2 + y_0^2).$$

Derive a recursion for computing the quotient polynomial $q(z)$ and $p(z_0) = b_{n-1}z_0 + b_n$, where

$$q(z) = b_0z^{n-2} + b_1z^{n-3} + \ldots + b_{n-2},$$
$$p(z) = q(z)d(z) + b_{n-1}z + b_n.$$

Count the number of real additions and multiplications needed to compute $p(z_0)$ and also show how to compute $p'(z_0)$.

3. (a) Using the Cardano–Tartaglia formula the real root α to the equation $x^3 = x + 4$ can be written in the form

$$\alpha = \sqrt[3]{2 + \frac{1}{9}\sqrt{321}} + \sqrt[3]{2 - \frac{1}{9}\sqrt{321}}.$$

Use this expression to compute α. Discuss the loss of accuracy due to cancellation.

(b) Compute α to the same accuracy by Newton's method using the initial approximation $x_0 = 2$.

4. A method due to D. Bernouilli for obtaining roots of the algebraic equation

$$p(x) = x^n + a_1 x^{n-1} + \cdots + a_{n-1}x + a_n = 0$$

is based on the related linear difference equation

$$\mu_k + a_1\mu_{k-1} + \cdots + a_{n-1}\mu_{k-n+1} + a_n\mu_{k-n} = 0,$$

having the same coefficients as the algebraic equation.

(a) Show that the general solution to the difference equation is

$$\mu_k = c_1\alpha_1^k + c_2\alpha_2^k + \cdots + c_n\alpha_n^k,$$

where α_k, $k = 1 : n$, are the roots of the algebraic equation.

(b) Assume that the roots are ordered after decreasing magnitude and that $|\alpha_1| > |\alpha_2|$. Show that if α_1 is real, then for almost all choices of initial values $\mu_0, \ldots, \mu_{n-1}$, it holds that

$$\lim_{k\to\infty} \frac{\mu_k}{\mu_{k-1}} = \alpha_1.$$

(c) Let C be the companion matrix of $p(x)$. Show that the sequence μ_k can be generated by forming successive matrix-vector products

$$m_k = Cm_{k-1}, \quad m_k = (\,\mu_{k+n-1} \quad \ldots \quad \mu_{k+1} \quad \mu_k\,)^T.$$

Show that $m_k = C^k m_0$.

Comment: When applied to a general matrix C this is known as the power method for computing eigenvalues.

5. In Graeffe's root-squaring method one separates even and odd powers of the polynomial $p(z)$ and squares the equation as follows

$$(a_0 z^n + a_2 z^{n-2} + a_4 z^{n-4} + \cdots)^2 = (a_1 z^{n-1} + a_3 z^{n-3} + a_5 z^{n-5} + \cdots)^2.$$

Putting $u = z^2$ the resulting equation becomes

$$q(u) = p(-z)p(z) = b_0 u^n + b_1 u^{n-1} + \cdots + b_n = 0.$$

This has roots equal to the squares of the roots of the original equation.

(a) Show that the coefficients b_k of $q(u)$ can be computed from

$$b_0 = a_0^2, \qquad (-1)^k b_k = a_k^2 + \sum_{j=1}^{k} (-1)^j 2a_{k-j}a_{k+j}, \quad k = 1, 2, \ldots, n.$$

(b) After squaring m times we obtain (after normalizing $A_0 = 1$) an equation in $u = z^{2m}$

$$u^n + A_1 u^{n-1} + A_2 u^{n-2} + \cdots + A_n = 0,$$

with roots $\beta_k = \alpha_k^{2m}$. Assume that the roots α_k of the original equation $p(z) = 0$ are real and distinct. Use the relations between coefficients and roots of an algebraic equation to show that for m large enough we have

$$\beta_1 \approx -A_1, \qquad \beta_2 \approx -A_2/A_1, \qquad \beta_3 \approx -A_3/A_2, \ldots.$$

(c) Square the polynomial $z^3 - 8z^2 + 17z - 10$ three times, and then use the relations in (b) to compute approximations to its three real roots.

6. Consider the iteration $z_{n+1} = z_n^2 + c$, where $c = p + iq$ is a fixed complex number. For a given z_0 the sequence of iterates $z_n = x_n + iy_n$, $n = 0, 1, 2, \ldots$ may either converge to one of the two roots of the quadratic equation $z^2 - z + c = 0$ or diverge to infinity. Consider z_0 chosen, e.g., in the unit square of the complex plane. The boundary separating the region of convergence from other points in the plane is a very complex **fractal curve** know as the **Julia set**. The **Mandelbrot set** is obtained by fixing $z_0 = 0$ and sweeping over values of c in a region of the complex plane.

(a) Picture the Julia set as follows. Set $c = 0.27334 + 0.000742i$. Sweep over points of z_0 in the region $-1 \le \Re z_0 \le 1$, $-1.3 \le \Im z_0 \le 1.3$. If $|z_N| < R$, for $N = 100$ and $R = 10$ color the point z_0 black. otherwise color the point from hot (red) to cool (blue) according to how fast the iteration is diverging, i.e. according to how fast the inequality $|z_n| > R$ becomes satisfied.

(b) Picture the Mandelbrot set in a similar way. Sweep over values of c in the region $-2.25 \le \Re c \le 0.75$, $-1.5 \le \Im c \le 1.5$.

Notes and References

An interesting historical account of Newton's method is given in Ypma [30]. Newton's method is contained in his book "Method of Fluxions" written 1671, but not published until 1736. Joseph Raphson was allowed to see Newton's work and Newton's method first was first published in a book by Raphson 1690. This is why the method in English literature is often called the Newton–Raphson method. The first to give a modern description of Newton's method using derivatives seems to have been Thomas Simpson 1740. Edmund Halley was contemporary with Isaac Newton and his third order method was published more than 300 years ago [8].

Halley's method has been rediscovered by J. H. Lambert [13] and numerous other people; see Traub [25, Sec. 5.22]. A nice exposition of this and other third order methods is given by Gander [6]. Families of iteration methods of arbitrary order ar studied in a remarkable paper by Schröder [22]. An English translation of this paper is given by G. W. Stewart [23]. The determinant family of iteration functions $B_p(x)$ is a special case of a parametrized family of iteration functions for polynomials given by Traub [26]; see also [10, Sec. 4.4]. This family was derived independently by Kalantari et al. [11].

One of the best algorithms to combine bisection and interpolation was developed by van Wijngaarden and Dekker at Mathematical Center in Amsterdam in the 1960s; see [3]. It was taken up and improved by Brent [2]; see also [5], Section 7.2. Brent's new algorithm, in contrast to Dekker's, never converges much more slowly

than bisection. Fortran and C versions of some of the zero-finding and minimization routines given in the book are available from Netlib.

Several comprehensive monographs dealing with methods for solving scalar nonlinear equations are available. Traub [25, 1964] gives an exhaustive enumeration of iteration methods with and without memory, with their order of convergence and efficiency index. Much classical material is also found in Ostrowski [19, 1973]. The elegant treatment by Householder [10, 1970] also deserves special mention.

The recently reprinted book by Brent [2] deals exclusively with methods which only uses function values for finding zeros and minima of functions of a single variable. It is unique in the careful treatment of algorithmic details that are crucial when developing reliable computer codes.

There is is a vast literature on methods for solving algebraic equations. An excellent introduction is given by Householder [10] and detailed surveys found, e.g., in Durand [4] and Sendov et al. [24]. The modified Newton method is due to Madsen [14] Another much used method for computing polynomial roots is the Jenkins–Traub method, which is included in the IMSL library. A good discussion of this rather complex method is found in Ralston and Rabinowitz [21, Sec. 8.11]. An evaluation of the speed and accuracy of the QR algorithm, used in MATLAB has been given by [7].

The theory of Sturm sequences are treated in [10, Sec. 2.5]. The quadtree method was used by Weyl [27] to give a constructive proof of the fundamental theorem of algebra. An interesting analysis of the efficient implementation of this method is given by Pan [20], who also gives a brief account of the history of algorithms for solving polynomial equations.

[1] David H. Bailey, Jon M. Borwein, Peter B. Borwein, and Simon Plouffe. The quest for pi. *Notes Amer. Math. Soc.*, 19:1:50–57, 1975.

[2] Richard P. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall, Englewood Cliffs, NJ, 1973. Reprinted by Dover Publications, Mineola, New York, 2002.

[3] J. C. P. Bus and T. J. Dekker. Two efficient algorithms with guaranteed convergence for finding a zero of a function. *ACM Trans. Math. Software*, 1:330–345, 1975.

[4] E. Durand. *Solutions numériques des équations algébriques. Tom I: Équations du type $F(x) = 0$, racines d'un polynôme.* Masson, Paris, 1960.

[5] George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler. *Computer Methods for Mathematical Computations.* Prentice-Hall, Englewood Cliffs, NJ, 1977.

[6] Walter Gander. On Halley's iteration method. *American Mathematics Monthly*, 92:131–134, 1985.

[7] S. Goedecker. Remark on algorithms to find roots of polynomials. *SIAM J. Sci. Comput.*, 15:1059–1063, 1994.

[8] Edmond Halley. A new and general method of finding the roots of equations. *Philos. Trans. Roy. Soc. London*, 18:136, 1694.

[9] Peter Henrici. *Applied and Computational Complex Analysis. Volume 1 Power Series, Integration, Conformal Mapping, Location of Zeros.* Wiley Classics Library, New York, 1988. Reprint of the 1974 original.

[10] Alston S. Householder. *The Numerical Treatment of a Single Nonlinear Equation.* McGraw-Hill, New York, 1970.

[11] Bahman Kalantari, Iraj Kalantari, and Rahim Zaare-Nahandi. A basic family of iteration functions for polynomial root finding and its characterizations. *J. Comput. Appl. Math.*, 80:209–226, 1997.

[12] Göran Kjellberg. Two observations on Durand–Kerner's root finding method. *BIT*, 24:556–559, 1973.

[13] Johann H. Lambert. *Beiträge zum Gebrauche der Mathematik und deren Anwendungen. Zweiter Theil.* 1770.

[14] Kaj Madsen. A root-finding algorithm based on Newton's method. *BIT*, 13:71–75, 1973.

[15] Kaj Madsen and John K. Reid. Fortran subroutines for finding polynomial zeros. Tech. Report A.E.R.E. R.7986, Computer Science and Systems Division, A.E.R.E. Harwell, 1975.

[16] Cleve B. Moler. *Numerical Computing with Matlab.* SIAM, Philadelphia, PA, 2004.

[17] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[18] James M. Ortega and Werner C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York, 1970.

[19] A. M. Ostrowski. *Solution of Equations in Euclidian and Banach Spaces.* Academic Press, New York, third edition, 1973.

[20] V. Y. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Review*, 39:187–220, 1997.

[21] Anthony Ralston and Philip Rabinowitz. *A First Course in Numerical Analysis.* McGraw-Hill, New York, second edition, 1978.

[22] Ernst Schröder. Über unendlich viele Algorithmen zur Auflösung der Gleichungen. *Mathematische Annalen*, 2:317–365, 1870.

[23] Ernst Schröder. On infinitely many algorithms for solving equations (translated by G. W. Stewart). Tech. Report TR-92-121, Department of Computer Science, University of Maryland, College Park, MD, 1992.

[24] Bl. Sendov, A. Andreev, and N. Kjurkchiev. Numerical solution of polynomial equations. In P. G. Ciarlet and J. L. Lions, editors, *Handbook of Numerical Analysis*, volume III, pages 629–778. Elsevier Science, Cambridge, UK, 1994.

[25] J. F. Traub. *Iterative Methods for the Solution of Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1964.

[26] J. F. Traub. A class of globally convergent iteration functions for the solution of polynomial equations. *Math. Comp.*, 20:113–138, 1966.

[27] Hermann Weyl. Randbemerkungen zu Hauptproblemen der Mathematik, II, Fundamentalsatz der Algebra und Grundlagen der Mathematik. *Math. Z.*, 20:131–151, 1924.

[28] James H Wilkinson. The perfidious polynomial. In G. H. Golub, editor, *Studies in Numerical Analysis*, pages 1–28. American Mathematical Society, Providence, RI, 1984.

[29] L. Yao and Adi Ben-Israel. The Newton and Halley methods for complex roots. *Amer. Math. Monthly*, 105:806–818, 1998.

[30] Tjalling J. Ypma. Development of the Newton–Raphson method. *SIAM Review*, 37:531–551, 1995.

Index

algorithm
    bisection method, 5
    golden section search, 49
attainable accuracy
    for multiple root, 8
    simple root, 7

bisection method, 2–6

companion matrix, 55
contraction mapping, 11
convergence
    cubic, 38
    linear, 14
    order of, 14
    sublinear, 14, 18
    superlinear, 14
cubic convergence
    methods of, 38–41

deflation, 61–62
Descartes' rule of sign, 58
domain of uncertainty, 7

efficiency index, 15, 23
elementary symmetric functions, 56
Euler's method, 38

false position method, 18–19
fixed point, 10
fixed point iteration, 10
    with memory, 20
fractal curve, 70

golden section
    ratio, 49
    search, 49

Halley's method, 38

ill-conditioned
    multiple roots, 8
    polynomial roots, 54–55
Illinois method, 27
intermediate-value theorem, 2
interval Newton's method, 36
interval reduction method, 47
inverse interpolation, 25
iteration method
    Euler's, 38
    Halley's, 38, 45
    Laguerre's, 66–67
    Muller–Traub's, 64
    Newton's, 28
    Newton–Maehly's, 62
    Newton–Raphson's, 28
    Schröder's, 40

Julia set, 70

Laguerre's method, 66–67
line search, 47–52

Mandelbrot set, 70
minimization
    one-dimensional, 47–52
Muller–Traub's method, 24, 64
multi-section method, 6
multiplicity of root, 8

Newton's method, 28–36
    complex case, 34
    convergence of, 29–36
    interval, 36
    modified, 64
Newton–Maehly's method, 62

Newton–Raphson's method, 28

point of attraction, 11
polynomial, 53–63
    reciprocal, 56, 62
    shift of origin, 57
polynomial roots
    simultaneous determination, 62–63

quadtree algorithm, 60

regula falsi, *see* false-position method

Schröder methods, 40
secant method, 20–23
    modified, 27
    rate of convergence, 22
    waltz rhythm, 23
secular equation, 46
simple root, 7
square root
    algorithm for, 32
Sturm sequences, 58–61
sublinear convergence, 14, 18
superlinear convergence, 14
synthetic division, 57
    quadratic factor, 57

termination criteria, 9

unimodal function, 47

Weierstrass' method, 63

zero suppression, 62

Contents

7 Direct Methods for Solving Linear System **1**

7.1 Linear Algebra and Matrix Analysis 2
7.1.1 Linear Vector Spaces 2
7.1.2 Matrix and Vector Algebra 4
7.1.3 Determinants and Permutations 8
7.1.4 Partitioning and Block Matrices 10
7.1.5 Modified Linear Systems 12
7.1.6 The Singular Value Decomposition 14
7.1.7 Norms of Vectors and Matrices 17
7.1.8 Conditioning of Linear Systems 22
Review Questions 27
Problems 27
7.2 Elimination Methods 29
7.2.1 Triangular Matrices 29
7.2.2 Gaussian Elimination 31
7.2.3 Elementary Elimination Matrices 39
7.2.4 Pivoting Strategies 43
7.2.5 Computational Variants 48
7.2.6 Computing the Inverse 52
Review Questions 55
Problems 55
7.3 Symmetric Matrices 56
7.3.1 Symmetric Positive Definite Matrices 56
7.3.2 Cholesky Factorization 61
7.3.3 Inertia of Symmetric Matrices 65
7.3.4 Symmetric Indefinite Matrices 66
Review Questions 70
Problems 70
7.4 Banded Linear Systems 71
7.4.1 Banded Matrices 71
7.4.2 LU Factorization of Banded Matrices 73
7.4.3 Tridiagonal Linear Systems 77
7.4.4 Inverses of Banded Matrices 81
Review Questions 82

Problems . . . 82
7.5 Perturbation Theory and Condition Estimation . . . 84
7.5.1 Component-Wise Perturbation Analysis . . . 84
7.5.2 Backward Error Bounds . . . 87
7.5.3 Estimating Condition Numbers . . . 89
Review Questions . . . 92
Problems . . . 92
7.6 Rounding Error Analysis . . . 93
7.6.1 Floating Point Arithmetic . . . 93
7.6.2 Error Analysis of Gaussian Elimination . . . 95
7.6.3 Scaling of Linear Systems . . . 100
7.6.4 Iterative Refinement of Solutions . . . 103
7.6.5 Interval Matrix Computations . . . 106
Review Questions . . . 110
Problems . . . 110
7.7 Block Algorithms for Gaussian Elimination . . . 110
7.7.1 Block and Blocked Algorithms . . . 110
7.7.2 Recursive Algorithms . . . 116
7.7.3 Kronecker Systems . . . 117
7.7.4 Linear Algebra Software . . . 119
Review Questions . . . 121
Problems . . . 121
7.8 Sparse Linear Systems . . . 122
7.8.1 Introduction . . . 122
7.8.2 Storage Schemes for Sparse Matrices . . . 123
7.8.3 Graph representation of sparse matrices. . . . 126
7.8.4 Nonzero Diagonal and Block Triangular Form . . . 128
7.8.5 LU Factorization of Sparse Matrices . . . 130
7.8.6 Cholesky Factorization of Sparse Matrices . . . 132
Review Questions . . . 136
Problems . . . 137
7.9 Structured Systems . . . 138
7.9.1 Toeplitz and Hankel Matrices . . . 138
7.9.2 Cauchy-Like Matrices . . . 139
7.9.3 Vandermonde systems . . . 140

Bibliography **145**

Index **150**

Chapter 7

Direct Methods for Solving Linear System

The problem treated in this chapter is the numerical solution of a system $Ax = b$ of m linear equations in n variables. Systems of linear equations enters at some stage in almost every scientific computing problem. Often their solution is the dominating part of the work to solve the problem. Also the solution of a *nonlinear* problem is usually accomplished by solving a *sequence* of linear systems obtained, e.g., by Newton's method. Since algorithms for solving linear systems are perhaps the most widely used in scientific computing, it is of great importance that they are efficient and reliable.

The linear system $Ax = b$ has a unique solution for all vectors b only if the matrix A has full row and column rank. If $\text{rank}\,(A) < n$ the system either has many solutions (is underdetermined) or no solution (is overdetermined). Note that due to inaccuracy of the elements of A the rank may not be well defined. Under- and overdetermined systems will be treated in Chapter 8.

Two quite different classes of methods for solving systems of linear equations are of interest: **direct** methods and **iterative** methods. In a direct method the system is transformed by a sequence of elementary transformed into a system of simpler form, e.g., triangular or diagonal form, which can be solved in an elementary way. The most important direct method is Gaussian elimination, which is the method of choice when the matrix A is of full rank and has no special structure.

Disregarding rounding errors, direct methods give the exact solution after a finite number of arithmetic operations. Iterative methods, on the other hand, compute a sequence of approximate solutions, which (assuming exact arithmetic) in the limit converges to the exact solution x. Iterative methods have the advantage that in general they only require a subroutine for computing the matrix-vector product Ax for any given vector x. Hence they may be much more efficient than direct methods when the matrix A is large and matrix-vector multiplication cheap. The distinction is not sharp since iterative methods are usually applied to a so called preconditioned version of the system that may involve the solution of a sequence of simpler auxiliary systems by a direct method. Iterative methods and preconditioning techniques are treated in Chapter 11.

Many applications give rise to linear systems where the matrix has some special property that can be used to achieve savings in work and storage. An important case is when A is symmetric positive definite, when about half the work and storage can be saved; see Section 7.3. If only a small fraction of the elements in A are nonzero the linear system $Ax = b$ is called **sparse**. The simplest case is when A has a banded structure, but also more general sparsity patterns can be taken advantage of; see Section 7.8. Indeed, without the exploitation of sparsity many important problems would be intractable!

There are also some classes of structured matrices, which although not sparse, have a structure, which can be used to develop fast solution methods. One example is Vandermonde matrices, which are related to polynomial interpolation. Other important examples of structured matrices are Toeplitz and Hankel matrices. In all these instances the n^2 elements in the matrix are derived from only $(n-1)$ quantities.

Numerical methods for linear systems are a good illustration of the difference between classical mathematics and practical numerical analysis. Even though the mathematical theory is simple and the algorithms have been known for centuries, decisive progress in the development of algorithms has been made during the last few decades. It is important to note that methods, which are perfectly acceptable for theoretical use, may be useless for the numerical solution. For example, the explicit determinant formula (Cramer's rule) for the inverse matrix and for the solution of linear systems of equations is extremely uneconomical except for matrices of order two or three, and matrices of very special structure.

Since critical details in the algorithms can influence the efficiency and accuracy in a way the beginner can hardly expect the reader is strongly advised to use the efficient and well-tested software available in the public domain; see Section 7.7.3.

The emphasis in this chapter will be on algorithms for *real* linear systems, since (with the exception for Hermitian systems) these occur most commonly in applications. However, all algorithms given can readily be generalized to the complex case.

7.1 Linear Algebra and Matrix Analysis

7.1.1 Linear Vector Spaces

We denote the field of real numbers by $\mathbf{R}$ and $\mathbf{R}^n$ is the vector space of n-tuples of real numbers. The operation addition and scalar multiplication are defined for all $v \in \mathbf{R}^n$, and have the following properties:

1. the following distributive properties hold:

$$\alpha(v+w) = \alpha v + \alpha w, \qquad (\alpha+\beta)v = \alpha v + \beta v,$$

   for all $\alpha, \beta \in \mathbf{K}$ and v, $w \in \mathbf{W}$.

2. there is an element $0 \in \mathbf{W}$ called the **null vector** such that $v + 0 = v$ for all $v \in \mathbf{R}^n$;

3. for each vector v there exists a vector $-v$ such that $v + (-v) = 0$;

4. $0 \cdot v = 0$ and $1 \cdot v = v$ where 0 and 1 are the zero and unity in $\mathbf{K}$.

Similar properties hold for the vector space $\mathbf{C}^n$ of n-tuples of elements of the field of complex numbers by $\mathbf{C}$.

If $\mathbf{W} \subset \mathbf{V}$ is a vector space then $\mathbf{W}$ is called a **vector subspace** of $\mathbf{V}$. The set of all linear combinations of $v_1, \ldots, v_k \in \mathbf{V}$ form a vector subspace denoted by

$$\text{span}\,\{v_1, \ldots, v_k\} = \sum_{i=1}^{k} \alpha_i v_i, \quad \alpha_i \in \mathbf{K}, \quad i = 1 : k.$$

If $\mathbf{S_1}, \ldots, \mathbf{S_k}$ are vector subspaces of $\mathbf{V}$ then their sum defined by

$$S = \{v_1 + \cdots + v_k |\ v_i \in \mathbf{S_i},\ i = 1 : k\}$$

is also a vector subspace. The intersection T of a set of vector subspaces is also a subspace,

$$T = \mathbf{S_1} \cap \mathbf{S_2} \cdots \cap \mathbf{S_k}.$$

(The union of vector spaces is generally no vector space.) If the intersection of the subspaces are empty, $\mathbf{S_i} \cap \mathbf{S_j} = 0$, $i \neq j$, then the sum of the subspaces is called their **direct sum** and denoted by

$$\mathbf{S} = \mathbf{S_1} \oplus \mathbf{S_2} \cdots \oplus \mathbf{S_k}.$$

A set of vectors $\{v_1, v_2, \ldots, v_k\}$ in $\mathbf{V}$ is said to be **linearly independent** if

$$\sum_{i=1}^{k} c_i v_i = 0, \quad \Rightarrow \quad c_1 = c_2 = \cdots = c_k = 0.$$

Otherwise, if a nontrivial linear combination of $v_1, \ldots, v_k$ is zero, the vectors are said to be linearly dependent. Then at least one vector v_i will be a linear combination of the rest.

A **basis** in $\mathbf{V}$ is any set of linearly independent vectors $v_1, v_2, \ldots, v_n \in \mathbf{V}$ such that all vectors $v \in \mathbf{V}$ can be uniquely decomposed as

$$v = \sum_{i=1}^{n} \xi_i v_i.$$

The scalars ξ_i are called the components or coordinates of v with respect to the basis $\{v_i\}$.

If the vector space $\mathbf{V}$ has a basis of k vectors, then every system of linearly independent vectors of $\mathbf{V}$ has at most k elements and any other basis of $\mathbf{V}$ has the same number k of elements. The number k is called the **dimension** of $\mathbf{V}$ and denoted by $\dim(\mathbf{V})$.

The **standard basis** for $\mathbf{C}^n$ is the set of unit vectors $e_1, e_2, ..., e_n$, where the jth component of e_i equals 1 if $j = i$, and 0 otherwise. We shall use the same name for a vector as for its coordinate representation by a column vector, with respect to the standard basis. For the the vector space $\mathcal{P}_n$ of polynomials of degree less than n monomials $1, x, \ldots, x^{n-1}$ form a basis.

7.1.2 Matrix and Vector Algebra

A **matrix** A is a collection of $m \times n$ numbers ordered in m rows and n columns

$$A = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}.$$

We write $A \in \mathbf{R}^{m\times n}$, where $\mathbf{R}^{m\times n}$ denotes the set of all real $m \times n$ matrices. If $m = n$, then the matrix A is said to be square and of order n. If $m \neq n$, then A is said to be rectangular. The empty matrix is a matrix of dimension 0×0 with no columns and no rows. Empty matrices are convenient to use as place holders.

A **column vector** is a matrix consisting of just one column and we write $x \in \mathbf{R}^m$ instead of $x \in \mathbf{R}^{m\times 1}$. Similarly a **row vector** is a matrix consisting of just one row.

A **linear map** from the vector space $\mathbf{C}^n$ to $\mathbf{C}^m$ is a function f such that

$$f(\alpha v + \beta w) = \alpha f(u) + \beta f(v)$$

for all $\alpha, \beta \in \mathbf{K}$ and $u, v \in \mathbf{C}^n$. Let x and y be the column vectors representing the vectors v and $f(v)$, respectively , using the standard basis of the two spaces. Then there is a unique matrix $A \in \mathbf{C}^{m\times n}$ representing this map such that

$$y = Ax.$$

This gives a link between linear maps and matrices.

We will follow a convention introduced by Householder[1] and use capital letters (e.g. A, B) to denote matrices. The corresponding lower case letters with subscripts ij then refer to the (i, j) component of the matrix (e.g. a_{ij}, b_{ij}). Greek letters $\alpha, \beta, \dots$ are usually used to denote scalars. Column vectors are usually denoted by lower case letters (e.g. x, y).

Two matrices in $\mathbf{R}^{m\times n}$ are said to be **equal**, $A = B$, if

$$a_{ij} = b_{ij}, \quad i = 1 : m, \quad j = 1 : n.$$

The basic operations with matrices are defined as follows. The product of a matrix A with a scalar α is

$$B = \alpha A, \qquad b_{ij} = \alpha a_{ij}.$$

The **sum** of two matrices A and B in $\mathbf{R}^{m\times n}$ is

$$C = A + B, \qquad c_{ij} = a_{ij} + b_{ij}. \tag{7.1.1}$$

As a special case of the multiplication rule, if $A \in \mathbf{R}^{m\times n}$, $x \in \mathbf{R}^n$, then

$$y = Ax \in \mathbf{R}^m, \qquad y_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1 : m.$$

[1] A. S. Householder 1904–1993, American mathematician at Oak Ridge National Laboratory and University of Tennessee. He pioneered the use of matrix factorization and orthogonal transformations in numerical linear algebra.

The **product** of two matrices A and B is defined if and only if the number of columns in A equals the number of rows in B. If $A \in \mathbf{R}^{m\times n}$ and $B \in \mathbf{R}^{n\times p}$ then

$$C = AB \in \mathbf{R}^{m\times p}, \qquad c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}, \tag{7.1.2}$$

and can be computed with mnp multiplications.

Matrix multiplication is associative and distributive,

$$A(BC) = (AB)C, \qquad A(B + C) = AB + AC,$$

but not *not commutative*. The product BA is not even defined unless $p = m$. Then the matrices $AB \in \mathbf{R}^{m\times m}$ and $BA \in \mathbf{R}^{n\times n}$ are both square, but if $m \neq n$ of different orders. In general, $AB \neq BA$ even when $m = n$. If $AB = BA$ the matrices are said to **commute**.

Example 7.1.1.

If $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$ and $C \in \mathbf{R}^{p\times q}$, then the product $M = ABC \in \mathbf{R}^{m\times q}$ is defined. computing M as $(AB)C$ requires $mp(n + q)$ operations, whereas using $A(BC)$ requires $nq(m + p)$ operations. These numbers can be very different! For example, if A and B are square $n \times n$ matrices and x a column vector of length n then computing the product ABx as $(AB)x$ requires $n^3 + n^2$ operations whereas $A(Bx)$ only requires $2n^2$ operations. When $n \gg 1$ this makes a great difference!

It is useful to define also **array operations**, which are carried out element-by-element on vectors and matrices. Following the convention in MATLAB we denote array multiplication and division by $.*$ and $./$, respectively. If A and B have the same dimensions $A\,.*\,B$ is the matrix with elements equal to $a_{ij} \cdot b_{ij}$ and $A./B$ has elements a_{ij}/b_{ij}. (Note that for $+, -$ array operations coincides with matrix operations so no distinction is necessary.)

The **transpose** A^T of a matrix $A = (a_{ij})$ is the matrix whose rows are the columns of A, i.e., if $C = A^T$ then $c_{ij} = a_{ji}$. For a complex matrix we denote by A^H the complex conjugate transpose of A

$$A = (a_{ij}), \qquad A^H = (\bar{a}_{ji}),$$

and it holds that $(AB)^H = B^H A^H$.

Row vectors are obtained by transposing column vectors (e.g. x^T, y^T). For the transpose of a product we have

$$(AB)^T = B^T A^T,$$

i.e., the product of the transposed matrices in *reverse order*.

We recall that $r = \operatorname{rank}(A)$ is the number of linearly independent columns which is the same as the number of linearly independent rows of A. A square matrix A of order n is said to **nonsingular** if $\operatorname{rank}(A) = n$. It is left as an exercise to show that the rank of a sum of two matrices satisfies

$$\operatorname{rank}(A + B) \leq \operatorname{rank}(A) + \operatorname{rank}(B), \tag{7.1.3}$$

and the rank of a product of two matrices satisfies

$$\operatorname{rank}(AB) \le \min\{\operatorname{rank}(A), \operatorname{rank}(B)\}. \tag{7.1.4}$$

If A is square and nonsingular there exists an **inverse matrix** denoted by A^{-1} with the property that

$$A^{-1}A = AA^{-1} = I.$$

By A^{-T} we will denote the matrix $(A^{-1})^T = (A^T)^{-1}$. For the inverse of a product of two matrices we have

$$(AB)^{-1} = B^{-1}A^{-1},$$

where the product of the inverse matrices are taken in reverse order.

The absolute value of a matrix A and vector b is defined by

$$|A|_{ij} = (|a_{ij}|), \qquad |b|_i = (|b_i|).$$

We also introduce the partial ordering "$\le$" for matrices A, B and vectors x, y, which is to be interpreted component-wise[2]

$$A \le B \iff a_{ij} \le b_{ij}, \qquad x \le y \iff x_i \le y_i.$$

Further, it is easy to show that if $C = AB$, then

$$|c_{ij}| \le \sum_{k=1}^{n} |a_{ik}|\,|b_{kj}|,$$

and hence $|C| \le |A|\,|B|$. A similar rule holds for matrix-vector multiplication.

The Euclidean **inner product** of two vectors x and y in $\mathbf{R}^n$ is given by

$$x^T y = \sum_{i=1}^{n} x_i y_i = y^T x, \tag{7.1.5}$$

and the **Euclidian length** of the vector x is

$$\|x\|_2 = (x^T x)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}. \tag{7.1.6}$$

The **outer product** of $x \in \mathbf{R}^m$ and $y \in \mathbf{R}^n$ is the matrix

$$xy^T = \begin{pmatrix} x_1 y_1 & \dots & x_1 y_n \\ \vdots & & \vdots \\ x_m y_1 & \dots & x_m y_n \end{pmatrix} \in \mathbf{R}^{m \times n}. \tag{7.1.7}$$

For some problems it is more relevant and convenient to work with complex vectors and matrices. We denote by $\mathbf{C}^{n \times m}$ the vector space of all complex $n \times m$

[2]Note that $A \le B$ in other contexts means that $B - A$ is positive semidefinite.

matrices whose components are complex numbers.[3] Most concepts introduced here carry over to complex matrices. Addition and multiplication of vectors and matrices follow the same rules as before. The **Hermitian** inner product of two vectors x and y in $\mathbf{C}^n$ is defined by

$$x^H y = \sum_{k=1}^{n} \bar{x}_k y_k, \tag{7.1.8}$$

where $x^H = (\bar{x}_1, \ldots, \bar{x}_n)$ and $\bar{x}_k$ denotes the complex conjugate of x_k. Hence $x^H y = \overline{y^H x}$ and $x^H x$ is a real number.

Any matrix D for which $d_{ij} = 0$ if $i \neq j$ is called a **diagonal matrix**. If $x \in \mathbf{R}^n$ is a vector then $D = \operatorname{diag}(x) \in \mathbf{R}^{n\times n}$ is the diagonal matrix formed by the elements of x. For a matrix $A \in \mathbf{R}^{n\times n}$ the elements a_{ii}, $i = 1 : n$, form the **main diagonal** of A, and we write

$$\operatorname{diag}(A) = \operatorname{diag}(a_{11}, a_{22}, \ldots, a_{nn}).$$

For $k = 1 : n-1$ the elements $a_{i,i+k}$ $(a_{i+k,i})$, $i = 1 : n-k$ form the kth **superdiagonal** (**subdiagonal**) of A. The elements $a_{i,n-i+1}$, $i = 1 : n$ form the (main) **antidiagonal** of A.

The **unit matrix** $I_n \in \mathbf{R}^{n\times n}$ is defined by

$$I_n = \operatorname{diag}(1, 1, \ldots, 1) = (e_1, e_2, \ldots, e_n),$$

and the k-th column of I_n is denoted by e_k. We have that $I_n = (\delta_{ij})$, where δ_{ij} is the **Kronecker symbol** $\delta_{ij} = 0, i \neq j$, and $\delta_{ij} = 1, i = j$. For all square matrices of order n it holds $AI_n = I_n A = A$. If the size of the unit matrix is obvious we delete the subscript and just write I.

Definition 7.1.1.

A matrix A is said to have **upper bandwidth** *r and* **lower bandwidth** *s if*

$$a_{ij} = 0, \quad j > i + r, \qquad a_{ij} = 0, \quad i > j + s,$$

respectively. This means that the number of non-zero diagonals above and below the main diagonal are r and s respectively. The maximum number of nonzero elements in any row is then $w = r + s + 1$, which is the **bandwidth** *of A.*

For a matrix $A \in \mathbf{R}^{m\times n}$ which is not square we define the bandwidth as

$$w = \max_{1\le i\le m} \{j - k + 1 \mid a_{ij} a_{ik} \neq 0\}.$$

Note that the bandwidth of a matrix depends on the ordering of its rows and columns. An important, but hard, problem is to find an optimal ordering of columns that minimize the bandwidth. However, there are good heuristic algorithms that can be used in practice and give almost optimal results; see Section 7.6.3.

[3] In MATLAB the only data type used is a matrix with either real or complex elements.

Several classes of band matrices that occur frequently have special names. Thus, a matrix for which $r = s = 1$ is called **tridiagonal**, if $r = 0$, $s = 1$ ($r = 1$, $s = 0$) it is called lower (upper) **bidiagonal** etc. A matrix with $s = 1$ ($r = 1$) is called an upper (lower) **Hessenberg** matrix.

A matrix A is called **symmetric** if its elements are symmetric about its main diagonal, i.e. $a_{ij} = a_{ji}$, $1 \le i < j \le n$, or equivalently $A^T = A$. A complex matrix A is called **Hermitian** if $A^H = A$ and **skew-Hermitian** if $A^H = -A$. The product of two Hermitian matrices is symmetric if and only if A and B commute, that is, $AB = BA$. If $A^T = -A$, then A is called **skew-symmetric**.

A square matrix A is called **persymmetric** if it is symmetric about its antidiagonal, i.e., $a_{ij} = a_{n-j+1,n-i+1}$.

7.1.3 Determinants and Permutations

The classical definition of the determinant requires some elementary facts about permutations, which we now state. Let $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ be a permutation of the integers $\{1, 2, \ldots, n\}$. The pair α_r, α_s, $r < s$ is said to form an inversion in the permutation if $\alpha_r > \alpha_s$. For example, in the permutation $\{2, \ldots, n, 1\}$ there are $(n-1)$ inversions $(2,1), (3,1), \ldots, (n,1)$. A permutation α is said to be even and $\operatorname{sign}(\alpha) = 1$ if it contains an even number of inversions; otherwise the permutation is odd and $\operatorname{sign}(\alpha) = -1$. The product of two permutations σ and τ is the composition $\sigma\tau$ defined by

$$\sigma\tau(i) = \sigma[\tau(i)], \quad i = 1 : n.$$

A **transposition** τ is a permutation which only interchanges two elements. Any permutation can be decomposed into a sequence of transpositions, but this decomposition is not unique.

Lemma 7.1.2.

A transposition τ of a permutation will change the number of inversions in the permutation by an odd number and thus $\operatorname{sign}(\tau) = -1$.

Proof. If τ interchanges two adjacent elements α_r and α_{r+1} in the permutation $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, this will not affect inversions in other elements. Hence the number of inversions increases by 1 if $\alpha_r < \alpha_{r+1}$ and decreases by 1 otherwise. Suppose now that τ interchanges α_r and α_{r+q}. This can be achieved by first successively interchanging α_r with α_{r+1}, then with α_{r+2}, and finally with α_{r+q}. This takes q steps. Next the element α_{r+q} is moved in $q-1$ steps to the position which α_r previously had. In all it takes an *odd number* $2q-1$ of transpositions of adjacent elements, in each of which the sign of the permutation changes. □

Definition 7.1.3.

The **determinant** *of a square matrix $A \in \mathbf{R}^{n\times n}$ is the scalar*

$$\det(A) = \sum_{\alpha \in S_n} \operatorname{sign}(\alpha)\, a_{1,\alpha_1} a_{2,\alpha_2} \cdots a_{n,\alpha_n}, \tag{7.1.9}$$

where the sum is over all permutations of the set $\{1,\dots,n\}$ *and* $\operatorname{sign}(\alpha) = \pm 1$ *according to whether* α *is an even or odd permutation.*

Note that there are $n!$ terms in (7.1.9) and each term contains exactly one factor from each row and each column in A. It follows easily that $\det(\alpha A) = \alpha^n \det(A)$ and $\det(A^T) = \det(A)$. The matrix A is nonsingular if and only if $\det(A) \neq 0$.

Theorem 7.1.4.

Let the matrix A *be nonsingular. Then the solution of the linear system* $Ax = b$ *can be expressed as*

$$x_i = \det(A_j)/\det(A), \quad i = 1:n, \tag{7.1.10}$$

where A_j *is the matrix* A *where the* j*th column has been replaced by the right hand side* b*.*

The expression (7.1.10) is known as **Cramer's rule**.[4] Although elegant, it is both computationally expensive and numerically instable. It should not be used for numerical computation except in very special cases.

The direct use of the definition (7.1.9) to evaluate $\det(A)$ would require about $nn!$ operations, which rapidly becomes infeasible as n increases. A much more efficient way to compute $\det(A)$ is by repeatedly using the following properties:

Theorem 7.1.5.

(i) *The value of the* $\det(A)$ *is unchanged if a row (column) in* A *multiplied by a scalar is added to another row (column).*

(ii) *The determinant of a triangular matrix equals the product of the elements in the main diagonal, i.e., if* U *is upper triangular*

$$\det(U) = u_{11}u_{22}\cdots u_{nn}.$$

(iii) *If two rows (columns) in* A *are interchanged the value of* $\det(A)$ *is multiplied by* (-1)*.*

(iv) *The product rule* $\det(AB) = \det(A)\det(B)$*.*

For a linear system $Ax = b$ there are three possibilities: it may have no solution, one unique solution, or an infinite set of solutions. If $b \in \mathcal{R}(A)$, or equivalently $\operatorname{rank}(A,\ b) = \operatorname{rank}(A)$, the system is said to be **consistent**. If $r = m$ then $\mathcal{R}(A)$ equals $\mathbf{R}^m$ and the system is consistent for all b. Clearly a consistent linear system always has *at least one solution* x.

The corresponding homogeneous linear system $Ax = 0$ is satisfied by any $x \in \mathcal{N}(A)$ and thus has $(n - r)$ linearly independent solutions. It follows that if a solution to an inhomogeneous system $Ax = b$ exists, it is unique only if $r = n$, whence $\mathcal{N}(A) = \{0\}$.

[4] Named after the Swiss mathematician Gabriel Cramer 1704–1752.

7.1.4 Partitioning and Block Matrices

A matrix formed by the elements at the intersection of a set of rows and columns of a matrix A is called a **submatrix**. For example, the matrices

$$\begin{pmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{pmatrix}, \qquad \begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix},$$

are submatrices of A. The second submatrix is called a contiguous submatrix since it is formed by contiguous elements of A.

Definition 7.1.6.

A **submatrix** *of $A = (a_{ij}) \in \mathbf{R}^{m\times n}$, is a matrix $B \in \mathbf{R}^{p\times q}$ formed by selecting p rows and q columns of A,*

$$B = \begin{pmatrix} a_{i_1 j_1} & a_{i_1 j_2} & \cdots & a_{i_1 j_q} \\ a_{i_2 j_1} & a_{i_2 j_2} & \cdots & a_{i_2 j_q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_p j_1} & a_{i_p j_2} & \cdots & a_{i_p j_q} \end{pmatrix},$$

where

$$1 \le i_1 \le i_2 \le \cdots \le i_p \le m, \quad 1 \le j_1 \le j_2 \le \cdots \le j_q \le n.$$

If $p = q$ and $i_k = j_k$, $k = 1 : p$, then B is a **principal submatrix** *of A. If in addition, $i_k = j_k = k$, $k = 1 : p$, then B is a* **leading** *principal submatrix of A.*

It is often convenient to think of a matrix (vector) as being built up of contiguous submatrices (subvectors) of lower dimensions. This can be achieved by **partitioning** the matrix or vector into blocks. We write, e.g.,

$$A = \begin{array}{c} \\ p_1\{ \\ p_2\{ \\ \vdots \\ p_M\{ \end{array} \begin{array}{c} \begin{array}{cccc} q_1 & q_2 & \cdots & q_N \end{array} \\ \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \cdots & A_{MN} \end{pmatrix} \end{array}, \qquad x = \begin{array}{c} p_1\{ \\ p_2\{ \\ \vdots \\ p_M\{ \end{array} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{pmatrix} \tag{7.1.11}$$

where A_{IJ} is a matrix of dimension $p_I \times q_J$. We call such a matrix a **block matrix**. The partitioning can be carried out in many ways, and is often suggested by the structure of the underlying problem. For square matrices the most important case is when $M = N$, and $p_I = q_I$, $I = 1 : N$. Then the diagonal blocks A_{II}, $I = 1 : N$, are square matrices.

The great convenience of block matrices lies in the fact that the operations of addition and multiplication can be performed by treating the blocks A_{IJ} as *non-commuting scalars* and applying the definitions (7.1.1) and (7.1.2). Therefore many algorithms defined for matrices with scalar elements have another simple generalization to partitioned matrices. Of course the dimensions of the blocks must correspond in such a way that the operations can be performed. When this is the case, the matrices are said to be partitioned **conformally**.

Let $A = (A_{IK})$ and $B = (B_{KJ})$ be two block matrices of block dimensions $M \times N$ and $N \times P$ respectively, where the partitioning corresponding to the index K is the same for each matrix. Then we have $C = AB = (C_{IJ})$, where

$$C_{IJ} = \sum_{K=1}^{N} A_{IK} B_{KJ}, \quad 1 \le I \le M, \quad 1 \le J \le P.$$

Often it is convenient to partition a matrix into rows or columns. Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$. Then the matrix product $C = AB \in \mathbf{R}^{m\times p}$ can be written

$$C = AB = (a_1\ a_2\ \cdots\ a_n) \begin{pmatrix} b_1^T \\ b_2^T \\ \vdots \\ b_n^T \end{pmatrix} = \sum_{k=1}^{n} a_k b_k^T, \tag{7.1.12}$$

where $a_k \in \mathbf{R}^m$ are the columns of A and $b_k^T \in \mathbf{R}^p$ the rows in B. Note that each term in the sum of (7.1.12) is an *outer product*. The more common inner product formula is obtained from the partitioning

$$C = AB = \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{pmatrix} (b_1\ b_2\ \cdots\ b_p) = (c_{ij}), \qquad c_{ij} = a_i^T b_j. \tag{7.1.13}$$

with $a_i, b_j \in \mathbf{R}^n$. Note that when the matrices A and B only have relatively few nonzero elements *the outer product formula* (7.1.12) *is a more efficient way to compute* AB! Further, if A and x are as in (7.1.11) then the product $z = Ax$ is a block vector with blocks

$$z_I = \sum_{K=1}^{N} A_{IK} x_K, \quad I = 1 : M.$$

Example 7.1.2.

Assume that the matrices A and B are conformally partitioned into 2×2 block form. Then

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}. \tag{7.1.14}$$

Be careful to note that since matrix multiplication is not commutative the *order* of the factors in the products cannot be changed! In the special case of block upper triangular matrices this reduces to

$$\begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} = \begin{pmatrix} R_{11}S_{11} & R_{11}S_{12} + R_{12}S_{22} \\ 0 & R_{22}S_{22} \end{pmatrix}.$$

Note that the product is again block upper triangular and its block diagonal simply equals the products of the diagonal blocks of the factors.

Let

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}, \quad U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}, \tag{7.1.15}$$

be 2×2 block lower and upper triangular matrices, respectively. For an upper block triangular matrix with square diagonal blocks $U_{II}, I = 1 : N$ we have

$$\det(U) = \det(U_{11}) \det(U_{22}) \cdots \det(U_{NN}), \tag{7.1.16}$$

Hence U is nonsingular if and only if all its diagonal blocks are nonsingular. Since $\det(L) = \det(L^T)$, a similar result holds for a lower block triangular matrix.

If L and U in (7.1.15) are nonsingular with square diagonal blocks, then their inverses are given by

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} L_{21} L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}, \qquad U^{-1} = \begin{pmatrix} U_{11}^{-1} & -U_{11}^{-1} U_{12} U_{22}^{-1} \\ 0 & U_{22}^{-1} \end{pmatrix}. \tag{7.1.17}$$

This can be verified by forming the products $L^{-1}L$ and $U^{-1}U$ using the rule for multiplying partitioned matrices.

7.1.5 Modified Linear Systems

Consider the block 2×2 linear system

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix}.$$

where A and D are square matrices, and A nonsingular. We can eliminate the x variables by premultiplying the first block by CA^{-1} and subtracting from the second block equations, giving

$$Sy = c - CA^{-1}b, \quad S = D - CA^{-1}B. \tag{7.1.18}$$

The matrix S is called the **Schur complement**of A.[5]

The elimination can be expressed in matrix form as

$$\begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} A & B \\ 0 & S \end{pmatrix}, \tag{7.1.19}$$

Inverting the block lower triangular matrix on the left hand side using (7.1.17) we obtain the block LU factorization

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I & 0 \\ CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ 0 & S \end{pmatrix}. \tag{7.1.20}$$

[5]Issai Schur (1875–1941) was born in Russia but studied at the University of Berlin, where he became full professor in 1919. Schur is mainly known for his fundamental work on the theory of groups but he also worked in the field of matrices.

From $M^{-1} = (LU)^{-1} = U^{-1}L^{-1}$ using the formulas (7.1.17) for the inverses of 2×2 block triangular matrices we get the **Schur–Banachiewicz** inverse formula[6]

$$M^{-1} = \begin{pmatrix} A^{-1} & -A^{-1}BS^{-1} \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{pmatrix}. \tag{7.1.21}$$

Similarly, assuming that D is nonsingular, we can factor M into a product of a block upper and a block lower triangular matrix

$$M = \begin{pmatrix} I & BD^{-1} \\ 0 & I \end{pmatrix} \begin{pmatrix} T & 0 \\ C & D \end{pmatrix}, \qquad T = A - BD^{-1}C, \tag{7.1.22}$$

where T is the Schur complement of D in M. (This is equivalent to block Gaussian elimination in reverse order.) From this factorization an alternative expression of M^{-1} can be derived,

$$M^{-1} = \begin{pmatrix} T^{-1} & -T^{-1}BD^{-1} \\ -D^{-1}CT^{-1} & D^{-1} + D^{-1}CT^{-1}BD^{-1} \end{pmatrix}. \tag{7.1.23}$$

If A and D are nonsingular the two triangular factorizations (7.1.20) and (7.1.22) both exist. Then, using (7.1.16), it follows that

$$\det(M) = \det(A - BD^{-1}C)\det(D) = \det(A)\det(D - CA^{-1}B).$$

In the special case that $D^{-1} = \lambda$, $B = x$, and $B = y$, this gives

$$\det(A - \lambda xy^T) = \det(A)(1 - \lambda y^T A^{-1}x). \tag{7.1.24}$$

This shows that $\det(A - \lambda xy^T) = 0$ if $\lambda = 1/y^T A^{-1}x$, a fact which is useful for the solution of eigenvalue problems.

The following formula gives an expression for the inverse of a matrix A after it is modified by a matrix of rank p, and is very useful in situations where $p \ll n$.

Theorem 7.1.7. *[Max A. Woodbury [69]]*

Let A and D be square nonsingular matrices and let B and C be matrices of appropriate dimensions such that $(A - BD^{-1}C)$ exists and is nonsingular. Then

$$(A - BD^{-1}C)^{-1} = A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1}, \tag{7.1.25}$$

which is the **Woodbury formula**.

Proof. The result follows directly by equating the $(1,1)$ blocks in the inverse M^{-1} in (7.1.21) and (7.1.23). □

[6] Tadeusz Banachiewicz (1882–1954) Polish astronomer and mathematician. In 1919 he became the director Cracow Observatory. He developed in 1925 a special kind of matrix algebra for "cracovians", which brought him international recognition.

If we specialize the Woodbury formula to the case where D is a scalar we get the well known **Sherman–Morrison formula**

$$(A - \sigma bc^T)^{-1} = A^{-1} + \alpha A^{-1}bc^T A^{-1}, \quad \alpha = 1/(\sigma^{-1} - c^T A^{-1}b). \tag{7.1.26}$$

It follows that $(A - \sigma bc^T)$ is nonsingular if and only if $\sigma \neq 1/c^T A^{-1}b$. This formula can be used to cheaply compute the new inverse when a matrix A is modified by a matrix of rank one. Such formulas are called updating formulas and are widely used in many contexts.

Frequently it is required to solve a linear problem, where the matrix has been modified by a correction of low rank. Consider first a linear system $Ax = b$, where $A \in \mathbf{R}^{n\times n}$ is modified by a correction of rank one,

$$(A - \sigma uv^T)\hat{x} = b. \tag{7.1.27}$$

Using the Sherman–Morrison formula the solution can be written

$$(A - \sigma uv^T)^{-1}b = A^{-1}b + \alpha A^{-1}u(v^T A^{-1}b), \quad \alpha = 1/(\sigma^{-1} - v^T A^{-1}u), \tag{7.1.28}$$

Here $x = A^{-1}b$ is the solution to the original system and $v^T A^{-1}b = v^T x$ is a scalar. Hence

$$\hat{x} = x + \beta w, \qquad \beta = v^T x/(\sigma^{-1} - v^T w), \quad w = A^{-1}u, \tag{7.1.29}$$

which shows that the solution $\hat{x}$ can be obtained from x by solving the system $Aw = u$. Note that *computing A^{-1} can be avoided.*

More generally, consider a linear system where the matrix has been modified with a matrix of rank $p > 1$,

$$(A + U\Sigma V^T)\hat{x} = b, \quad U, V \in \mathbf{R}^{n\times p}, \tag{7.1.30}$$

with $\Sigma \in \mathbf{R}^{p\times p}$ nonsingular. Using now the Woodbury formula, we can write

$$(A - U\Sigma V^T)^{-1}b = x + A^{-1}U(\Sigma^{-1} - V^T A^{-1}U)^{-1}V^T x. \tag{7.1.31}$$

This formula first requires the solution of the linear systems $AW = U$ with p right hand sides. The correction is then obtained by solving the linear system of size $p \times p$

$$(\Sigma^{-1} - V^T W)z = V^T x,$$

and forming Wz. If $p \ll n$ and the solution $x = A^{-1}b$ has been computed by a direct method it is this scheme is very efficient.

We end this with a note of caution that the updating methods given here can not be expected to be numerically stable in all cases. In particular, problems will arise when the initial problem is more illconditioned than the modified one.

7.1.6 The Singular Value Decomposition

The **singular value decomposition** (SVD) of a matrix $A \in \mathbf{R}^{m\times n}$ is of great theoretical and practical importance. Although its history goes back more than a century its use in numerical computations is much more recent.

Theorem 7.1.8. (Singular Value Decomposition.)
Every matrix $A \in \mathbf{R}^{m\times n}$ of rank r can be written

$$A = U\Sigma V^T, \qquad \Sigma = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \in \mathbf{R}^{m\times n}, \tag{7.1.32}$$

where $U \in \mathbf{R}^{m\times m}$ and $V \in \mathbf{R}^{n\times n}$ are orthogonal, $\Sigma_1 = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$, and

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0.$$

(Note that if $r = n$ and/or $r = m$, some of the zero submatrices in Σ disappear.) The σ_i are called the **singular values** *of A and if we write*

$$U = (u_1, \ldots, u_m), \qquad V = (v_1, \ldots, v_n),$$

the u_i, $i = 1, \ldots, m$, and v_i, $i = 1, \ldots, n$, are left and right **singular vectors** *, respectively.*

Proof. Let $f(x) = \|Ax\|_2 = (x^T A^T Ax)^{1/2}$, the Euclidian length of the vector $y = Ax$, and consider the problem

$$\sigma_1 := \max_x \{f(x) \mid x \in \mathbf{R}^n,\ \|x\|_2 \leq 1\}.$$

Here $f(x)$ is a convex function[7] defined on a convex, compact set. It is well known (see, e.g., Ciarlet [13, Sec. 7.4]) that the maximum σ_1 is then attained on an extreme point of the set. Let v_1 be such a point with $\sigma_1 = \|Av_1\|$, $\|v_1\|_2 = 1$. If $\sigma_1 = 0$ then $A = 0$, and (7.1.32) holds with $\Sigma = 0$, and U and V arbitrary orthogonal matrices. Therefore, assume that $\sigma_1 > 0$, and set $u_1 = (1/\sigma_1)Av_1 \in \mathbf{R}^m$, $\|u_1\|_2 = 1$. Let the matrices

$$V = (v_1, V_1) \in \mathbf{R}^{n\times n}, \qquad U = (u_1, U_1) \in \mathbf{R}^{m\times m}$$

be orthogonal. (Recall that it is always possible to extend an orthogonal set of vectors to an orthonormal basis for the whole space.) Since $U_1^T Av_1 = \sigma_1 U_1^T u_1 = 0$ it follows that $U^T AV$ has the following structure:

$$A_1 \equiv U^T AV = \begin{pmatrix} \sigma_1 & w^T \\ 0 & B \end{pmatrix},$$

where $w^T = u_1^T AV_1$ and $B = U_1^T AV_1 \in \mathbf{R}^{(m-1)\times(n-1)}$.

$$\left\| A_1 \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} \sigma_1^2 + w^T w \\ Bw \end{pmatrix} \right\|_2 \geq \sigma_1^2 + w^T w.$$

We have $UA_1 y = AVy = Ax$ and, since U and V are orthogonal, it follows that

$$\sigma_1 = \max_{\|x\|_2=1} \|Ax\|_2 = \max_{\|y\|_2=1} \|A_1 y\|_2,$$

[7] A function $f(x)$ is convex on a convex set S if for any x_1 and x_2 in S and any λ with $0 < \lambda < 1$, we have $f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$.

and hence,

$$\sigma_1(\sigma_1^2 + w^T w)^{1/2} \geq \left\| A_1 \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2 .$$

Combining these two inequalities gives $\sigma_1 \geq (\sigma_1^2 + w^T w)^{1/2}$, and it follows that $w = 0$. The proof can now be completed by an induction argument on the smallest dimension $\min(m, n)$. □

The geometrical significance of this theorem is as follows. The rectangular matrix A represents a mapping from $\mathbf{R}^n$ to $\mathbf{R}^m$. The theorem shows that there is an orthogonal basis in each of these two spaces, with respect to which this mapping is represented by a generalized diagonal matrix Σ. We remark that a singular value decomposition $A = U\Sigma V^H$, with U and V unitary, and Σ real diagonal, holds for any **complex** matrix $A \in \mathbf{C}^{m\times n}$.

The singular values of A are uniquely determined. The singular vector v_j, $j \leq r$, is unique (up to a factor ± 1) if σ_j^2 is a *simple* eigenvalue of $A^T A$. For multiple singular values, the corresponding singular vectors can be chosen as any orthonormal basis for the unique subspace that they span. Once the singular vectors v_j, $1 \leq j \leq r$ have been chosen, the vectors u_j, $1 \leq j \leq r$ are uniquely determined, and vice versa, using

$$Av_j = \sigma_j u_j, \qquad A^T u_j = \sigma_j v_j, \qquad j = 1, \ldots, r. \tag{7.1.33}$$

If U and V are partitioned according to

$$U = (U_1,\ U_2), \quad U_1 \in \mathbf{R}^{m\times r}, \quad V = (V_1,\ V_2), \quad V_1 \in \mathbf{R}^{n\times r}. \tag{7.1.34}$$

then the SVD can be written in the compact form

$$A = U_1 \Sigma_1 V_1^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T. \tag{7.1.35}$$

The last expression expresses A as a sum of matrices of rank one.

From (6.2.20) it follows that

$$A^T A = V \Sigma^T \Sigma V^T, \qquad AA^T = U \Sigma \Sigma^T U^T.$$

Thus σ_j^2, $j = 1, \ldots, r$ are the nonzero eigenvalues of the symmetric and positive semidefinite matrices $A^T A$ and AA^T, and v_j and u_j are the corresponding eigenvectors. Hence in principle the SVD can be reduced to the eigenvalue problem for symmetric matrices. For a proof of the SVD using this relationship see Stewart [1973, p. 319]. *However, this does not lead to a numerically stable way to compute the SVD*, since the singular values are square roots of the eigenvalues.

Definition 7.1.9.

The **range** *of the matrix $A \in \mathbf{R}m \times n$, denoted by $\mathcal{R}(A)$, is the subspace of $\mathbf{R}^m$ of dimension $r = \operatorname{rank}(A)$*

$$\mathcal{R}(A) = \{y \in \mathbf{R}^m |\ y = Ax,\ x \in \mathbf{R}^n\}. \tag{7.1.36}$$

The **null space** $\mathcal{N}(A)$ *of* A *is a subspace of* $\mathbf{R}^n$ *of dimension* $n - r$*:*

$$\mathcal{N}(A) = \{x \in \mathbf{R}^n | \ Ax = 0\}. \tag{7.1.37}$$

The SVD gives complete information about the four fundamental subspaces associated with A and A^T. It is easy to verify that the range of A and nullspace of A^T are given by

$$\mathcal{R}(A) = \mathcal{R}(U_1) \qquad \mathcal{N}(A^T) = \mathcal{R}(U_2) \tag{7.1.38}$$

Since $A^T = V\Sigma^T U^T$ it follows that also

$$\mathcal{R}(A^T) = \mathcal{R}(V_1) \qquad \mathcal{N}(A) = \mathcal{R}(V_2). \tag{7.1.39}$$

We immediately find the well-known relations

$$\mathcal{R}(A)^\perp = \mathcal{N}(A^T), \qquad \mathcal{N}(A)^\perp = \mathcal{R}(A^T),$$

7.1.7 Norms of Vectors and Matrices

In perturbation theory as well as in the analysis of errors in matrix computation it is useful to have a measure of the size of a vector or a matrix. Such measures are provided by vector and matrix norms, which can be regarded as generalizations of the absolute value function on $\mathbf{R}$.

Definition 7.1.10.

A **norm** *on a vector space* $\mathbf{V} \in \mathbf{C}^n$ *is a function* $\mathbf{V} \to \mathbf{R}$ *denoted by* $\|\cdot\|$ *that satisfies the following three conditions:*

1. $\|x\| > 0, \quad \forall x \in \mathbf{V}, \quad x \neq 0$ *(definiteness)*
2. $\|\alpha x\| = |\alpha| \, \|x\|, \quad \forall \alpha \in \mathbf{C}, \quad x \in \mathbf{C}^n$ *(homogeneity)*
3. $\|x + y\| \leq \|x\| + \|y\| \quad \forall x, y \in \mathbf{V}$ *(triangle inequality)*

The triangle inequality is often used in the form (see Problem 11)

$$\|x \pm y\| \geq \big| \, \|x\| - \|y\| \, \big|.$$

The most common vector norms are special cases of the family of **Hölder** norms or p-norms

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}, \qquad 1 \leq p < \infty. \tag{7.1.40}$$

The three most important particular cases are $p = 1, 2$ and the limit when $p \to \infty$:

$$\begin{aligned} \|x\|_1 &= |x_1| + \cdots + |x_n|, \\ \|x\|_2 &= (|x_1|^2 + \cdots + |x_n|^2)^{1/2} = (x^H x)^{1/2}, \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i|. \end{aligned} \tag{7.1.41}$$

A vector norm$\|\cdot\|$ is called **absolute** if $\|x\| = \|\,|x|\,\|$, and **monotone** if $|x| \le |y| \Rightarrow \|x\| \le \|y\|$. It can be shown that a vector norm is monotone if and only if it is absolute; see Stewart and Sun [61, Theorem II.1.3]. Clearly the vector p-norms are absolute for all $1 \le p < \infty$.

The vector 2-norm is also called the Euclidean norm. It is invariant under unitary (orthogonal) transformations since

$$\|Qx\|_2^2 = x^H Q^H Q x = x^H x = \|x\|_2^2$$

if Q is orthogonal.

The proof that the triangle inequality is satisfied for the p-norms depends on the following inequality. Let $p > 1$ and q satisfy $1/p + 1/q = 1$. Then it holds that

$$\alpha\beta \le \frac{\alpha^p}{p} + \frac{\beta^p}{q}.$$

Indeed, let x and y be any real number and λ satisfy $0 < \lambda < 1$. Then by the convexity of the exponential function it holds that

$$e^{\lambda x + (1-\lambda)y} \le \lambda e^x + (1-\lambda)e^y.$$

We obqatin the desired result by setting $\lambda = 1/p$, $x = p\log\alpha$ and $x = q\log\beta$.

Another important property of the p-norms is the **Hölder inequality**

$$|x^H y| \le \|x\|_p \|y\|_q, \quad \frac{1}{p} + \frac{1}{q} = 1, \quad p \ge 1. \tag{7.1.42}$$

For $p = q = 2$ this becomes the well known **Cauchy–Schwarz inequality**

$$|x^H y| \le \|x\|_2 \|y\|_2.$$

Another special case is $p = 1$ for which we have

$$|x^H y| = \left| \sum_{i=1}^n x_i^H y_i \right| \le \sum_{i=1}^n |x_i^H y_i| \le \max_i |y_i| \sum_{i=1}^n |x_i| = \|x\|_1 \|y\|_\infty. \tag{7.1.43}$$

Definition 7.1.11.

For any given vector norm $\|\cdot\|$ on $\mathbf{C}^n$ the **dual norm** *$\|\cdot\|_D$ is defined by*

$$\|x\|_D = \max_{y \ne 0} |x^H y| / \|y\|. \tag{7.1.44}$$

The vectors in the set

$$\{y \in \mathbf{C}^n \mid \|y\|_D \|x\| = y^H x = 1\}. \tag{7.1.45}$$

are said to be **dual vectors** *to x with respect to $\|\cdot\|$.*

It can be shown that the dual of the dual norm is the original norm (see [61, Theorem II.1.12]). It follows from the Hölder inequality that the dual of the p-norm is the q-norm, where

$$1/p + 1/q = 1.$$

The dual of the 2-norm can be seen to be itself. It can be shown to be the only norm with this property (see [42, Theorem 5.4.16]).

The vector 2-norm can be generalized by taking

$$\|x\|_{2,G}^2 = (x, x) = x^H G x, \tag{7.1.46}$$

where G is a Hermitian positive definite matrix. It can be shown that the unit ball $\{x : \|x\| \le 1\}$ corresponding to this norm is an ellipsoid, and hence they are also called **elliptic norms**. Other generalized norms are the **scaled** p-norms defined by

$$\|x\|_{p,D} = \|Dx\|_p, \quad D = \operatorname{diag}(d_1, \ldots, d_n), \quad d_i \neq 0, \quad i = 1 : n. \tag{7.1.47}$$

All norms on $\mathbf{C}^n$ are equivalent in the following sense: For each pair of norms $\|\cdot\|$ and $\|\cdot\|'$ there are positive constants c and c' such that

$$\frac{1}{c}\|x\|' \le \|x\| \le c'\|x\|', \quad \forall x \in \mathbf{C}^n. \tag{7.1.48}$$

In particular, it can be shown that for the p-norms we have

$$\|x\|_q \le \|x\|_p \le n^{(1/p-1/q)}\|x\|_q, \quad 1 \le p \le q \le \infty. \tag{7.1.49}$$

For example, setting $p = 2$ and $q = \infty$ we obtain

$$\|x\|_\infty \le \|x\|_2 \le \sqrt{n}\|x\|_\infty,$$

We now consider **matrix norms**. Given any vector norm, we can construct a matrix norm by defining

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\|. \tag{7.1.50}$$

This norm is called the **operator norm**, or the matrix norm **subordinate** to the vector norm. From the definition it follows directly that

$$\|Ax\| \le \|A\|\ \|x\|, \quad \forall x \in \mathbf{C}^n. \tag{7.1.51}$$

Whenever this inequality holds, we say that the matrix norm is **consistent** with the vector norm.

It is an easy exercise to show that operator norms are **submultiplicative**, i.e., whenever the product AB is defined it satisfies the condition

4. $N(AB) \le N(A)N(B)$

Explicit expressions for the matrix norms subordinate to the vector p-norms are known only for $p = 1, 2, \infty$:

Theorem 7.1.12.
For $p = 1, 2\ \infty$ the matrix subordinate p-norm are given by

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|, \tag{7.1.52}$$

$$\|A\|_2 = \max_{\|x\|=1} (x^H A^H A x)^{1/2} = \sigma_1(A), \tag{7.1.53}$$

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|. \tag{7.1.54}$$

Proof. To prove the result for $p = 1$ we partition $A = (a_1, \dots, a_n)$ by columns For any $x = (x_1, \dots, x_n)^T \neq 0$ we have

$$\|Ax\|_1 = \left\| \sum_{j=1}^{n} x_j a_j \right\|_1 \le \sum_{j=1}^{n} |x_j| \|a_j\|_1 \le \max_{1\le j\le n} \|a_j\|_1 \|x\|_1.$$

It follows that $\|Ax\|_1 \le \max_{1\le j\le n} \|a_j\|_1 = \|a_k\|_1$, for some $1 \le k \le n$. But then

$$\|Ae_k\|_1 = \|a_k\|_1 = \max_{1\le j\le n} \|a_j\|_1,$$

and hence $\|A\|_1 \ge \max_{1\le j\le n} \|a_j\|_1$. This implies (7.1.52). The formula (7.1.54) for the matrix ∞-norm is proved in a similar fashion. The expression for the 2-norm follows from the extremal property

$$\max_{\|x\|=1} \|Ax\|_2 = \max_{\|x\|=1} \|U\Sigma V^T x\|_2 = \sigma_1(A)$$

of the singular vector $x = v_1$. □

For $p = 1$ and $p = \infty$ the matrix subordinate norms are easily computable. Note that the 1-norm is the maximal column sum and the ∞-norm is the maximal row sum of the magnitude of the elements. It follows that $\|A\|_1 = \|A^H\|_\infty$.

The 2-norm, also called the **spectral norm**, equals the largest singular value $\sigma_1(A)$ of A. It has the drawback that it is expensive to compute. but is a useful analytical tool. Since the nonzero eigenvalues of $A^H A$ and AA^H are the same it follows that $\|A\|_2 = \|A^H\|_2$. An upper bound for the matrix 2-norm is

$$\|A\|_2 \le (\|A\|_1 \|A\|_\infty)^{1/2}. \tag{7.1.55}$$

The proof of this bound is given as an exercise in Problem 16.

Another way to proceed in defining norms for matrices is to regard $\mathbf{C}^{m\times n}$ as an mn-dimensional vector space and apply a vector norm over that space.

Definition 7.1.13.
The **Frobenius norm**[8] *is derived from the vector* 2*-norm*

$$\|A\|_F = \Big(\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2\Big)^{1/2} \tag{7.1.56}$$

The Frobenius norm is submultiplicative, but is often larger than necessary, e.g., $\|I_n\|_F = n^{1/2}$. This tends to make bounds derived in terms of the Frobenius norm not as sharp as they might be. From (7.1.58) it follows that

$$\frac{1}{\sqrt{n}}\|A\|_F \le \|A\|_2 \le \|A\|_F, \quad k = \min(m,n). \tag{7.1.57}$$

Note that $\|A^H\|_F = \|A\|_F$. Useful alternative characterizations of the Frobenius norm are

$$\|A\|_F^2 = \operatorname{trace}(A^H A) = \sum_{i=1}^{k}\sigma_i^2(A), \quad k = \min(m,n), \tag{7.1.58}$$

where $\sigma_i(A)$ are the nonzero singular values of A. Of the matrix norms the 1-,∞- and the Frobenius norm are absolute, but for the 2-norm the best result is

$$\| \, |A| \, \|_2 \le n^{1/2}\|A\|_2.$$

Table 7.1.1. *Numbers* γ_{pq} *such that* $\|A\|_p \le \gamma_{pq}\|A\|_q$*, where* $A \in \mathbf{C}^{m\times n}$ *and* $\operatorname{rank}(A) = r$*.*

$p\backslash q$	1	2	∞	F
1	1	$\sqrt{m}$	m	$\sqrt{m}$
2	$\sqrt{n}$	1	$\sqrt{m}$	$\sqrt{mn}$
∞	n	$\sqrt{n}$	1	$\sqrt{n}$
F	$\sqrt{n}$	$\sqrt{r}$	$\sqrt{m}$	1

The Frobenius norm shares with the 2-norm the property of being invariant with respect to unitary (orthogonal) transformations

$$\|QAP\| = \|A\|. \tag{7.1.59}$$

Such norms are called **unitarily invariant** and have an interesting history; see Stewart and Sun [61, Sec. II.3].

[8] Ferdinand George Frobenius (1849–1917) German mathematician, professor at ETH Zürich (1875–1892) before he succeeded Weierstrass at Berlin University.

Theorem 7.1.14.

Let $\|\cdot\|$ be a unitarily invariant norm. Then $\|A\|$ is a symmetric function of the singular values

$$\|A\| = \Phi(\sigma_1, \ldots, \sigma_n).$$

Proof. Let the singular value decomposition of A be $A = U\Sigma V^T$. Then the invariance implies that $\|A\| = \|\Sigma\|$, which shows that $\Phi(A)$ only depends on Σ. Since the ordering of the singular values in Σ is arbitrary Φ must be symmetric in σ_i, $i = 1 : n$. □

Unitarily invariant norms were characterized by John von Neumann [52], who showed that the converse of Theorem 7.1.14 is true: A function $\Phi(\sigma_1, \ldots, \sigma_n)$ which is symmetric in its arguments and satisfies the three properties in the Definition 7.1.10 of a vector norm defines a unitarily invariant matrix norm. In this connection such functions are called **symmetric gauge functions**. Two examples are

$$\|A\|_2 = \max_i \sigma_i, \qquad \|A\|_F = \left(\sum_{i=1}^{n} \sigma_i^2\right)^{1/2}.$$

One use of norms is the study of *limits of sequences of vectors and matrices* (see Sec. 9.2.4). Consider an infinite sequence $x_1, x_2, \ldots$ of elements of a vector space $\mathbf{V}$ and let $\|\cdot\|$ be a norm on $\mathbf{V}$. The sequence is said to converge (strongly if V is infinite dimensional) to a limit $x \in \mathbf{V}$, and we write $\lim_{k\to\infty} x_k = x$ if

$$\lim_{k\to\infty} \|x_k - x\| = 0,$$

For a finite dimensional vector space it follows from the equivalence of norms that convergence is independent of the choice of norm. The particular choice $\|\cdot\|_\infty$ shows that convergence of vectors in $\mathbf{C}^n$ is equivalent to convergence of the n sequences of scalars formed by the components of the vectors. By considering matrices in $\mathbf{C}^{m\times n}$ as vectors in $\mathbf{C}^{mn}$ the same conclusion holds for matrices.

7.1.8 Conditioning of Linear Systems

Consider a linear system $Ax = b$ where A is nonsingular and $b \neq 0$. The sensitivity of the solution x and the inverse A^{-1} to perturbations in A and b is of practical importance, since the matrix A and vector b are rarely known exactly. They may be subject to observational errors, or given by formulas which involve roundoff errors in their evaluation. (Even if they were known exactly, they may not be represented exactly as floating-point numbers in the computer.)

We start with deriving some results that are needed in the analysis.

Lemma 7.1.15.

Let $E \in \mathbf{R}^{n\times n}$ be a matrix for which $\|E\| < 1$. Then the matrix $(I - E)$ is nonsingular and for its inverse we have the estimate

$$\|(I - E)^{-1}\| \leq 1/(1 - \|E\|). \tag{7.1.60}$$

Proof. If $(I - E)$ is singular there exists a vector $x \neq 0$ such that $(I - E)x = 0$. Then $x = Ex$ and $\|x\| = \|Ex\| \leq \|E\| \, \|x\| < \|x\|$, which is a contradiction since $\|x\| \neq 0$. Hence $(I - E)$ is nonsingular.

Next consider the identity $(I - E)(I - E)^{-1} = I$ or

$$(I - E)^{-1} = I + E(I - E)^{-1}.$$

Taking norms we get

$$\|(I - E)^{-1}\| \leq 1 + \|E\| \, \|(I - E)^{-1}\|,$$

and (7.1.60) follows. (For another proof, see hint in Problem 7.2.19.) □

Corollary 7.1.16.

Assume that $\|B - A\|\|B^{-1}\| = \eta < 1$. *Then it holds that*

$$\|A^{-1}\| \leq \frac{1}{1-\eta}\|B^{-1}\|, \qquad \|A^{-1} - B^{-1}\| \leq \frac{\eta}{1-\eta}\|B^{-1}\|.$$

Proof. We have $\|A^{-1}\| = \|A^{-1}BB^{-1}\| \leq \|A^{-1}B\| \, \|B^{-1}\|$. The first inequality then follows by taking $E = B^{-1}(B - A) = I - B^{-1}A$ in Lemma 7.1.15. From the identity

$$A^{-1} - B^{-1} = A^{-1}(B - A)B^{-1} \tag{7.1.61}$$

we have $\|A^{-1} - B^{-1}\| \leq \|A^{-1}\| \, \|B - A\| \, \|B^{-1}\|$. The second inequality now follows from the first. □

Let x be the solution x to a system of linear equations $Ax = b$, and let $x + \delta x$ satisfy the perturbed system

$$(A + \delta A)(x + \delta x) = b + \delta b,$$

where δA and δb are perturbations in A and b. Subtracting out $Ax = b$ we get

$$(A + \delta A)\delta x = -\delta A x + \delta b.$$

Assuming that A and $A + \delta A$ are nonsingular, we can multiply by A^{-1} and solve for δx. This yields

$$\delta x = (I + A^{-1}\delta A)^{-1}A^{-1}(-\delta A x + \delta b), \tag{7.1.62}$$

which is the basic identity for the perturbation analysis.

In the simple case that $\delta A = 0$, we have $\delta x = A^{-1}\delta b$, which implies that $|\delta x| = |A^{-1}| \, |\delta b|$, Taking norms

$$\|\delta x\| \leq \|A^{-1}\| \, \|\delta b\|.$$

Usually it is more appropriate to consider *relative* perturbations,

$$\frac{\|\delta x\|}{\|x\|} \leq \kappa(A,x)\frac{\|\delta b\|}{\|b\|}, \quad \kappa(A,x) := \frac{\|Ax\|}{\|x\|}\|A^{-1}\|. \tag{7.1.63}$$

Here $\kappa(A,x)$ is the condition number with respect to perturbations in b. It is important to note that this implies that the size of the residual vector $r = b - A\bar{x}$ gives no direct indication of the *error* in an approximate solution $\bar{x}$. For this we need information about A^{-1} or the condition number $\kappa(A,x)$.

The inequality (7.1.63) is sharp in the sense that for any matrix norm and for any A and b there exists a perturbation δb, such that equality holds. From $\|b\| = \|Ax\| \leq \|A\|\,\|x\|$ it follows that

$$\kappa(A,x) \leq \|A\|\,\|A^{-1}\|, \tag{7.1.64}$$

but here *equality will hold only for rather special right hand sides* b. Equation (7.1.64) motivates the following definition:

Definition 7.1.17. *For a square nonsingular matrix* A *the* **condition number** *is*

$$\kappa = \kappa(A) = \|A\|\,\|A^{-1}\|. \tag{7.1.65}$$

where $\|\cdot\|$ *denotes any matrix norm.*

Clearly $\kappa(A)$ depends on the chosen matrix norm. If we want to indicate that a particular norm is used, then we write, e.g., $\kappa_\infty(A)$ etc. For the 2-norm we have using the SVD that $\|A\|_2 = \sigma_1$ and $\|A^{-1}\| = 1/\sigma_n$, where σ_1 and σ_n are the largest and smallest singular values of A. Hence

$$\kappa_2(A) = \sigma_1/\sigma_n. \tag{7.1.66}$$

Note that $\kappa(\alpha A) = \kappa(A)$, i.e., the condition number is invariant under multiplication of A by a scalar. From the definition it also follows easily that

$$\kappa(AB) \leq \kappa(A)\kappa(B).$$

Further, for all p-norms it follows from the identity $AA^{-1} = I$ that

$$\kappa_p(A) = \|A\|_p\|A^{-1}\|_p \geq \|I\|_p = 1,$$

that is, the condition number is always greater or equal to one.

We now show that $\kappa(A)$ also is the condition number with respect to perturbations in A.

Theorem 7.1.18.

Consider the linear system $Ax = b$*, where the matrix* $A \in \mathbf{R}^{n\times n}$ *is nonsingular. Let* $(A + \delta A)(x + \delta x) = b + \delta b$ *be a perturbed system and assume that*

$$\eta = \|A^{-1}\|\,\|\delta A\| < 1.$$

Then $(A + \delta A)$ is nonsingular and the norm of the perturbation δx is bounded by

$$\|\delta x\| \le \frac{\|A^{-1}\|}{1-\eta}\left(\|\delta A\|\,\|x\| + \|\delta b\|\right). \tag{7.1.67}$$

Proof. Taking norms in equation (7.1.62) gives

$$\|\delta x\| \le \|(I + A^{-1}\delta A)^{-1}\|\,\|A^{-1}\|\big(\|\delta A\|\,\|x\| + \|\delta b\|\big).$$

By assumption $\|A^{-1}\delta A\| \le \|A^{-1}\|\,\|\delta A\| = \eta < 1$. Using Lemma 7.1.15 it follows that $(I + A^{-1}\delta A)$ is nonsingular and

$$\|(I + A^{-1}\delta A)^{-1}\| \le 1/(1-\eta),$$

which proves the result. □

In most practical situations it holds that $\eta \ll 1$ and therefore $1/(1-\eta] \approx 1$. Therefore, if upper bounds

$$\|\delta A\| \le \epsilon_A\|A\|, \qquad \|\delta b\| \le \epsilon_b\|b\|, \tag{7.1.68}$$

for $\|\delta A\|$ and $\|\delta b\|$ are known, then for the normwise relative perturbation it holds that

$$\frac{\|\delta x\|}{\|x\|} \lessapprox \kappa(A)\left(\epsilon_A + \epsilon_b\frac{\|b\|}{\|A\|\|x\|}\right).$$

Substituting $b = I$, $\delta b = 0$ and $x = A^{-1}$ in (7.1.67) and proceeding similarly from $(A + \delta A)(X + \delta X) = I$, we obtain the perturbation bound for $X = A^{-1}$

$$\frac{\|\delta X\|}{\|X\|} \le \frac{\kappa(A)}{1-\eta}\frac{\|\delta A\|}{\|A\|}. \tag{7.1.69}$$

This shows that $\kappa(A)$ is indeed the condition number of A with respect to inversion.

The relative distance of a matrix A to the set of singular matrices in some norm is defined as

$$\operatorname{dist}(A) := \min\left\{\frac{\|\delta A\|}{\|A\|} \mid (A + \delta A) \text{ singular}\right\}. \tag{7.1.70}$$

The following theorem shows that the reciprocal of the condition number $\kappa(A)$ can be interpreted as a measure of the nearness to singularity of a matrix A.

Theorem 7.1.19 (Kahan [47]).
Let $A \in \mathbf{C}^{n\times n}$ be a nonsingular matrix and $\kappa(A) = \|A\|\|A^{-1}\|$ the condition number with respect to a norm $\|\cdot\|$ subordinate to some vector norm. Then

$$\operatorname{dist}(A) = \kappa^{-1}(A). \tag{7.1.71}$$

Proof. If $(A+\delta A)$ is singular, then there is a vector $x \neq 0$ such that $(A+\delta A)x = 0$. Then, setting $y = Ax$, it follows that

$$\|\delta A\| \geq \frac{\|\delta A\, x\|}{\|x\|} = \frac{\|Ax\|}{\|x\|} = \frac{\|y\|}{\|A^{-1}y\|} \geq \frac{1}{\|A^{-1}\|} = \frac{\|A\|}{\kappa(A)},$$

or $\|\delta A\|/\|A\| \geq 1/\kappa(A)$.

Now let x be a vector with $\|x\| = 1$ such that $\|A^{-1}x\| = \|A^{-1}\|$. Set $y = A^{-1}x/\|A^{-1}\|$ so that $\|y\| = 1$ and $Ay = x/\|A^{-1}\|$. Let z be a dual vector to y so that (see Definition 7.1.11) $\|z\|_D\|y\| = z^H y = 1$, where $\|\cdot\|_D$ is the dual norm. Then $\|z\|_D = 1$, and if we take

$$\delta A = -xz^H/\|A^{-1}\|,$$

it follows that

$$(A + \delta A)y = Ay - xz^H y/\|A^{-1}\| = (x - x)/\|A^{-1}\| = 0.$$

Hence $(A + \delta A)$ is singular. Further

$$\|\delta A\|\|A^{-1}\| = \|xz^H\| = \max_{\|v\|=1} \|(xz^H)v\| = \|x\| \max_{\|v\|=1} |z^H v| = \|z\|_D = 1,$$

and thus $\|\delta A\| = 1/\|A^{-1}\|$, which proves the theorem. □

The result in Theorem 7.1.19 can be used to get a *lower bound* for the condition number $\kappa(A)$, see, Problem 21. For the 2-norm the result follows directly from the SVD $A = U\Sigma V^H$. The closest singular matrix then equals $A + \delta A$, where

$$\delta A = -\sigma_n u_n v_n^H, \quad \|\delta A\|_2 = \sigma_n = 1/\|A^{-1}\|_2. \tag{7.1.72}$$

Matrices with small condition numbers are said to be **well-conditioned**. For any real, orthogonal matrix Q we have $\kappa_2(Q) = \|Q\|_2\|Q^{-1}\|_2 = 1$, so Q is perfectly conditioned in the 2-norm. Furthermore, for any orthogonal P and Q, we have $\kappa_2(PAQ) = \kappa_2(A)$, i.e., $\kappa_2(A)$ is invariant under orthogonal transformations.

When a linear system is ill-conditioned, i.e. $\kappa(A) \gg 1$, roundoff errors will in general cause a computed solution to have a large error. It is often possible to show that a small **backward error** in the following sense:

Definition 7.1.20.

An algorithm for solving a linear system $Ax = b$ is said to be (normwise) **backward stable** *if, for any data $A \in \mathbf{R}^{n\times n}$ and $b \in \mathbf{R}^n$, there exist perturbation matrices and vectors δA and δb, such that the solution $\bar{x}$ computed by the algorithm is the* exact *solution to a neighbouring system*

$$(A + \delta A)\bar{x} = (b + \delta b), \tag{7.1.73}$$

where

$$\|\delta A\| \leq c_1(n)u\|A\|, \qquad \|\delta b\| \leq c_2(n)u\|b\|.$$

A computed solution $\bar{x}$ is called a (normwise) stable solution if it satisfies (7.1.73).

Since the data A and b usually are subject to errors and not exact, it is reasonable to be satisfied with the computed solution $\bar{x}$ if the backward errors δA and δb are small in comparison to the uncertainties in A and b. As seen from (7.1.64), this does not mean that $\bar{x}$ is close to the exact solution x.

Review Questions

1. Define the concepts:
(i) Real symmetric matrix. (ii) Real orthogonal matrix.
(iii) Real skew-symmetric matrix. (iv) Triangular matrix.
(v) Hessenberg matrix.

2. (a) Given a vector norm define the matrix subordinate norm.
(b) Give explicit expressions for the matrix p norms for $p = 1, 2, \infty$.

3. Define the p norm of a vector x. Show that

$$\frac{1}{n}\|x\|_1 \leq \frac{1}{\sqrt{n}}\|x\|_2 \leq \|x\|_\infty,$$

which are special cases of (7.1.49).

4. Verify the formulas (7.1.21) for the inverse of a 2×2 block triangular matrices.

Problems

1. Show that if $R \in \mathbf{R}^{n\times n}$ is strictly upper triangular, then $R^n = 0$.

2. If A and B are square upper triangular matrices show that AB is upper triangular, and that A^{-1} is upper triangular if it exists. Is the same true for lower triangular matrices?

3. To solve a linear system $Ax = b$, where $A \in \mathbf{R}^n$, by Cramer's rule (see Equation (7.1.10) requires the evaluation of $n + 1$ determinants of order n. Estimate the number of multiplications needed for $n = 50$ if the determinants are evaluated in the naive way. Estimate the time it will take on a computer performing 10^9 floating point operations per second!

4. Show that if the complex matrix $U = Q_1 + iQ_2$ is unitary, then the real matrix

$$\tilde{U} = \begin{pmatrix} Q_1 & -Q_2 \\ Q_2 & Q_1 \end{pmatrix}$$

is orthogonal.

5. Let $A \in \mathbf{R}^{n\times n}$ be a given matrix. Show that if $Ax = y$ has *at least one* solution for any $y \in \mathbf{R}^n$, then it has *exactly one* solution for any $y \in \mathbf{R}^n$. (This is a useful formulation for showing uniqueness of approximation formulas.)

6. Let $A \in \mathbf{R}^{m\times n}$ have rows a_i^T, i.e., $A^T = (a_1, \ldots, a_m)$. Show that

$$A^T A = \sum_{i=1}^{m} a_i a_i^T.$$

What is the corresponding expression for $A^T A$ if A is instead partitioned into columns?

7. Let $S \in \mathbf{R}^{n\times n}$ be skew-Hermitian, i.e. $S^H = -S$.

(a) Show that $I - S$ is nonsingular.

(b) Show that the matrix $Q = (I-S)^{-1}(I+S)$, known as the **Cayley transform**[9] is unitary, i.e., $Q^H Q = I$.

8. Show that for $x \in \mathbf{R}^n$,

$$\lim_{p\to\infty} \|x\|_p = \max_{1\le i\le n} |x_i|.$$

9. Prove that the following inequalities are valid and best possible:

$$\|x\|_2 \le \|x\|_1 \le n^{1/2}\|x\|_2, \qquad \|x\|_\infty \le \|x\|_1 \le n\|x\|_\infty.$$

Derive similar inequalities for the comparison of the operator norms $\|A\|_1$, $\|A\|_2$, and $\|A\|_\infty$.

10. Show that any vector norm is uniformly continuous by proving the inequality

$$|\,\|x\| - \|y\|\,| \le \|x-y\|, \qquad x, y \in \mathbf{R}^n.$$

11. Show that for any matrix norm there exists a consistent vector norm.

Hint: Take $\|x\| = \|xy^T\|$ for any vector $y \in \mathbf{R}^n$, $y \neq 0$.

12. Derive the formula for $\|A\|_\infty$ given in Theorem 7.1.12.

13. Make a table corresponding to Table 7.1.1 for the vector norms $p = 1, 2, \infty$.

14. Prove that for any subordinate matrix norm

$$\|A+B\| \le \|A\| + \|B\|, \qquad \|AB\| \le \|A\|\|B\|.$$

15. Show that $\|A\|_2 = \|PAQ\|_2$ if P and Q are orthogonal matrices.

16. Use the result $\|A\|_2^2 = \rho(A^T A) \le \|A^T A\|$, valid for any matrix operator norm $\|\cdot\|$, where $\rho(A^T A)$ denotes the spectral radius of $A^T A$, to deduce the upper bound in (7.1.55).

17. Prove the expression (7.1.54) for the matrix norm subordinate to the vector ∞-norm.

18. (a) Let T be a nonsingular matrix, and let $\|\cdot\|$ be a given vector norm. Show that the function $N(x) = \|Tx\|$ is a vector norm.

(b) What is the matrix norm subordinate to $N(x)$?

(c) If $N(x) = \max_i |k_i x_i|$, what is the subordinate matrix norm?

[9] Arthur Cayley (1821–1895), English mathematician, is credited with developing the algebra of matrices. Although he worked as a lawyer for many years before he became Sadlerian professor at Cambridge in 1863, he has authored more than 900 papers.

19. Consider an upper block triangular matrix

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$

and suppose that R_{11}^{-1} and R_{22}^{-1} exists. Show that R^{-1} exists.

20. Use the Woodbury formula to prove the identity

$$(I - AB)^{-1} = I + A(I - BA)^{-1}B.$$

21. (a) Let A^{-1} be known and let B be a matrix coinciding with A except in one row. Show that if B is nonsingular then B^{-1} can be computed by about $2n^2$ multiplications using the Sherman–Morrison formula (7.1.26).

(b) Use the Sherman–Morrison formula to compute B^{-1} if

$$A = \begin{pmatrix} 1 & 0 & -2 & 0 \\ -5 & 1 & 11 & -1 \\ 287 & -67 & -630 & 65 \\ -416 & 97 & 913 & -94 \end{pmatrix}, \qquad A^{-1} = \begin{pmatrix} 13 & 14 & 6 & 4 \\ 8 & -1 & 13 & 9 \\ 6 & 7 & 3 & 2 \\ 9 & 5 & 16 & 11 \end{pmatrix},$$

and B equals A except that the element 913 has been changed to 913.01.

22. Use the result in Theorem 7.1.19 to obtain the lower bound $\kappa_\infty(A) \geq 3/(2|\epsilon|) = 1.5|\epsilon|^{-1}$ for the matrix

$$A = \begin{pmatrix} 1 & -1 & 1 \\ -1 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon \end{pmatrix}, \qquad 0 < |\epsilon| < 1.$$

(The true value is $\kappa_\infty(A) = 1.5(1 + |\epsilon|^{-1})$.)

7.2 Elimination Methods

7.2.1 Triangular Matrices

An **upper triangular** matrix is a matrix U for which $u_{ij} = 0$ whenever $i > j$. A square upper triangular matrix has form

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ & & \dots & u_{nn} \end{pmatrix}.$$

If also $u_{ij} = 0$ when $i = j$ then U is **strictly** upper triangular. Similarly a matrix L is **lower triangular** if $l_{ij} = 0$, $i < j$, and strictly lower triangular if $l_{ij} = 0$, $i \leq j$. Clearly the transpose of an upper triangular matrix is lower triangular and vice versa.

Triangular matrices have several nice properties. It is easy to verify that sums, products and inverses of square upper (lower) triangular matrices are again triangular matrices of the same type. The diagonal elements of the product $U = U_1U_2$ of two triangular matrices are just the product of the diagonal elements in U_1 and U_2. From this it follows that the diagonal elements in U^{-1} are the inverse of the diagonal elements in U.

Triangular linear systems are easy to solve. In an upper triangular linear system $Ux = b$, the unknowns can be computed recursively from

$$x_n = b_n/u_{nn} \qquad x_i = \Big(b_i - \sum_{k=i+1}^{n} u_{ik}x_k\Big)\Big/u_{ii}, \quad i = n-1:1. \tag{7.2.1}$$

Since the unknowns are solved for in *backward* order, this is called **back-substitution**. Similarly, in a lower triangular linear system $Ly = c$, the unknowns can be computed by **forward-substitution**.

$$y_1 = c_1/u_{11} \qquad y_i = \Big(c_i - \sum_{k=1}^{i-1} l_{ik}y_k\Big)\Big/l_{ii}, \quad i = 2:n. \tag{7.2.2}$$

When implementing a matrix algorithm such as (7.2.1) or (7.2.2) on a computer, the *order of operations* in matrix algorithms may be important. One reason for this is the economizing of storage, since even matrices of moderate dimensions have a large number of elements. When the initial data is not needed for future use, computed quantities may overwrite data. To resolve such ambiguities in the description of matrix algorithms it is important to be able to describe computations in a more precise form. For this purpose we will use an informal programming language, which is sufficiently precise for our purpose but allows the suppression of cumbersome details. These concepts are illustrated for the back-substitution (7.2.1). in the following program where the solution vector x overwrites the data vector b.

Algorithm 7.2.1 Back-substitution.
Given an upper triangular matrix $U \in \mathbf{R}^{n\times n}$ and a vector $b \in \mathbf{R}^n$, the following algorithm computes $x \in \mathbf{R}^n$ such that $Ux = b$:

for $i = n:-1:1$
$\quad s := \sum_{k=i+1}^{n} u_{ik}b_k;$
$\quad b_i := (b_i - s)/u_{ii};$
end

Here := is the assignment symbol and $x := y$ means that the value of y is assigned to x. Note that in order to minimize round-off errors b_i is added *last* to the sum; compare the error bound (2.4.3).

Another possible sequencing of the operations is:

```
for k = n : −1 : 1
    b_k := b_k/u_kk;
    for i = k − 1 : −1 : 1
        b_i := b_i − u_ik b_k;
    end
end
```

Here the elements in U are accessed column-wise instead of row-wise as in the previous algorithm. Such differences can influence the efficiency when implementing matrix algorithms. For example, if U is stored column-wise as is the convention in Fortran, the second version is to be preferred.

We will often use the concept of a **flop**, to mean roughly the amount of work associated with the computation

$$s := s + a_{ik}b_{kj},$$

i.e., one floating point addition and multiplication and some related subscript computation.[10] With this notation solving a triangular system requires $\frac{1}{2}n^2$ flam.

7.2.2 Gaussian Elimination

Consider a linear system $Ax = b$, where the matrix $A \in \mathbf{R}^{m\times n}$, and vector $b \in \mathbf{R}^m$ are given and the vector $x \in \mathbf{R}^n$ is to be determined, i.e.,

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}. \tag{7.2.3}$$

A fundamental observation is that the following elementary operation can be performed on the system without changing the set of solutions:

- Adding a multiple of the ith equation to the jth equation.
- Interchange two equations.

These correspond in an obvious way to row operations on the augmented matrix (A, b). It is also possible to interchange two columns in A provided we make the corresponding interchanges in the components of the solution vector x.

The idea behind **Gaussian elimination**[11] is to use such elementary operations to eliminate the unknowns in the system $Ax = b$ in a systematic way, so that

[10] Note that in older textbooks this was called a flop. However, in more recent books (e.g., Higham [41, 2002]) a flop is instead defined as a floating point add *or* multiply.

[11] Named after Carl Friedrich Gauss (1777–1855), but known already in China as early as the first century BC.

at the end an equivalent upper triangular system is produced, which is then solved by back-substitution. If $a_{11} \neq 0$, then in the first step we eliminate x_1 from the last $(n-1)$ equations by subtracting the multiple

$$l_{i1} = a_{i1}/a_{11}, \quad i = 2:n,$$

of the first equation from the ith equation. This produces a reduce system of $(n-1)$ equations in the $(n-1)$ unknowns $x_2 : x_n$, where the new coefficients are given by

$$a_{ij}^{(2)} = a_{ij} - l_{i1}a_{1j}, \qquad b_i^{(2)} = b_i - l_{i1}b_1, \quad i = 2:n.$$

If $a_{22}^{(2)} \neq 0$, we can next in a similar way eliminate x_2 from the last $(n-2)$ of these equations. After $k-1$ steps, $k \leq \min(m,n)$, of Gaussian elimination the matrix A has been reduced to the form

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \cdots & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} \\ & & \ddots & \vdots & & \vdots \\ & & & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\ & & & \vdots & & \vdots \\ & & & a_{mk}^{(k)} & \cdots & a_{mn}^{(k)} \end{pmatrix}, \qquad b^{(k)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_k^{(k)} \\ \vdots \\ b_m^{(k)} \end{pmatrix}, \tag{7.2.4}$$

where we have put $A^{(1)} = A$, $b^{(1)} = b$. The diagonal elements a_{11}, $a_{22}^{(2)}$, $a_{33}^{(3)}$, $\ldots$, which appear during the elimination are called **pivotal elements**.

Let A_k denote the kth leading principal submatrix of A. Since the determinant of a matrix does not change under row operations the determinant of A_k equals the product of the diagonal elements then by (7.2.5)

$$\det(A_k) = a_{11}^{(1)} \cdots a_{kk}^{(k)}, \quad k = 1:n.$$

For a square system, i.e. $m = n$, this implies that all pivotal elements $a_{ii}^{(i)}$, $i = 1:n$, in Gaussian elimination are nonzero if and only if $\det(A_k) \neq 0$, $k = 1:n$. In this case we can continue the elimination until after $(n-1)$ steps we get the single equation

$$a_{nn}^{(n)} x_n = b_n^{(n)} \qquad (a_{nn}^{(n)} \neq 0).$$

We have now obtained an upper triangular system $A^{(n)}x = b^{(n)}$, which can be solved recursively by back-substitution (7.2.1). We also have

$$\det(A) = a_{11}^{(1)} a_{22}^{(2)} \cdots a_{nn}^{(n)}. \tag{7.2.5}$$

We have seen that if in Gaussian elimination a zero pivotal element is encountered, i.e., $a_{kk}^{(k)} = 0$ for some $k \leq n$, then we cannot proceed and it seems the algorithm breaks down.

Example 7.2.1. Consider the system

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

For $\epsilon \neq 1$ this system is nonsingular and has the unique solution $x_1 = -x_2 = -1/(1-\epsilon)$. However, when $a_{11} = \epsilon = 0$ the first step in Gaussian elimination cannot be carried through. The remedy here is obviously to interchange the two equations, which directly gives an upper triangular system.

Suppose that in step k of Gaussian elimination we have $a_{kk}^{(k)} = 0$. (The equations may have been reordered in previous steps, but we assume that the notations have been changed accordingly.) If A is nonsingular, then in particular its first k columns are linearly independent. This must also be true for the first k columns of the reduced matrix. Hence some element $a_{ik}^{(k)}$, $i = k : n$ must be nonzero, say $a_{pk}^{(k)} \neq 0$. By interchanging rows k and p this element can be taken as pivot and it is possible to proceed with the elimination. The important conclusion is that *any nonsingular system of equations can be reduced to triangular form by Gaussian elimination if appropriate row interchanges are used.*

Note that when rows are interchanged in A the same interchanges must be made in the elements of the vector b. Note also that the determinant formula (7.2.5) must be modified to

$$\det(A) = (-1)^s a_{11}^{(1)} a_{22}^{(2)} \cdots a_{nn}^{(n)}, \tag{7.2.6}$$

where s denotes the total number of row and columns interchanges performed.

If $\operatorname{rank}(A) < n$ then it is possible that at some step k we have $a_{ik}^{(k)} = 0$, $i = k : n$. If the entire submatrix $a_{ij}^{(k)}$, $i, j = k : n$, is zero, then $\operatorname{rank}(A) = k$ and we stop. Otherwise there is a nonzero element, say $a_{pq}^{(k)} \neq 0$, which can be brought into pivoting position by interchanging rows k and p and columns k and q. (Note that when columns are interchanged in A the same interchanges must be made in the elements of the solution vector x.) Proceeding in this way any matrix A can always be reduced to upper **trapezoidal form**,

$$A^{(r)} = \left(\begin{array}{ccc|ccc} a_{11}^{(1)} & \cdots & a_{1r}^{(1)} & a_{1,r+1}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & \ddots & \vdots & & \vdots & \vdots \\ \vdots & & a_{rr}^{(r)} & a_{r,r+1}^{(r)} & \cdots & a_{rn}^{(r)} \\ \hline 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right), \qquad b^{(r)} = \left(\begin{array}{c} b_1^{(1)} \\ \vdots \\ b_r^{(r)} \\ \hline b_{r+1}^{(r+1)} \\ \vdots \\ b_m^{(r+1)} \end{array}\right). \tag{7.2.7}$$

Here the number r of linearly independent rows in a matrix A equals the number of independent columns in A is the **rank** of A.

From the reduced form (7.2.7) we can read off the rank of A. The two rectangular zero blocks in $A^{(r)}$ have dimensions $(m-r) \times r$ and $(m-r) \times (n-r)$, respectively. We deduce the following:

1. The system $Ax = b$ has a unique solution if and only if $r = m = n$.

2. If $b_k^{(r+1)} = 0$, $k = r + 1 : m$, then the system $Ax = b$ is consistent and has an infinite number of solutions. We can assign arbitrary values to the last $n - r$ components of (the possibly permuted) solution vector x. The first r components are then uniquely determined and obtained using back-substitution with the nonsingular triangular matrix in the upper left corner.

3. If $b_k^{(r+1)} \neq 0$, for some $k > r$, the the system $Ax = b$ is inconsistent and has no solution. Then we have to be content with finding x such that the residual vector $r = b - Ax$ is small in some sense.

In principle, the reduced trapezoidal form (7.2.7) obtained by Gaussian elimination yields the rank of a the matrix A, and also answers the question whether the given system is consistent or not. However, this is the case only if exact arithmetic is used. In floating point calculations it may be difficult to decide if a pivot element, or an element in the transformed right hand side, should be considered as zero or nonzero. For example, a zero pivot in exact arithmetic will almost invariably be polluted by roundoff errors in such a way that it equals some small nonzero number. What tolerance to use to decide when a pivot should be taken to be zero will depend on the context. In order to treat this question in a satisfactory way we need the concept **numerical rank**, which will be introduced in Section 8.3.3.

Another question, which we have to leave unanswered for a while, concerns underdetermined and overdetermined systems, which arise quite frequently in practice! Problems where there are more parameters than needed to span the right hand side lead to underdetermined systems. In this case we need additional information in order to decide which solution to pick. On the other hand, overdetermined systems arise when there is more data than needed to determine the solution. In this case the system usually is inconsistent and there is no solution. Now the question can be posed, how to find a solution, which in some sense best approximates the right hand side? These questions are related, are best treated using *orthogonal* transformations rather than GE. This topic is again deferred to Chapter 8.

When the matrix A is square and of a full rank Gaussian Elimination can be described as follows:

Algorithm 7.2.2 Gaussian Elimination; square case.

Given a matrix $A = A^{(1)} \in \mathbf{R}^{n\times n}$ and a vector $b = b^{(1)} \in \mathbf{R}^n$, the following algorithm reduces the system $Ax = b$ to the upper triangular form, provided that the pivotal elements $a_{kk}^{(k)} \neq 0$, $k = 1 : n$:

for $k = 1 : n - 1$
  for $i = k + 1 : n$
    $l_{ik} := a_{ik}^{(k)} / a_{kk}^{(k)}$;
    for $j = k + 1 : n$
      $a_{ij}^{(k+1)} := a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}$;

```
        end
        b_i^(k+1) := b_i^(k) − l_ik b_k^(k);
    end
end
```

Note that when l_{ik} is computed the element $a_{ik}^{(k)}$ is put equal to zero. Thus memory space can be saved by storing the multipliers in the lower triangular part of the matrix. If the multipliers l_{ik} are saved, then the operations on the vector b can be deferred to a later stage. This observation is important in that it shows that *when solving a sequence of linear systems*

$$Ax_i = b_i, \quad i = 1 : p,$$

with the same matrix A but different right hand sides, the operations on A only have to be carried out once!

From Algorithm 7.2.2 it follows that $(n-k)$ divisions and $(n-k)^2$ multiplications and additions are used in step k to transform the elements of A. A further $(n-k)$ multiplications and additions are used to transform the elements of b. Summing over k and neglecting low order terms we find that the total number of flams required by Gaussian elimination is

$$\sum_{k=1}^{n-1}(n-k)^2 \approx n^3/3, \qquad \sum_{k=1}^{n-1}(n-k) \approx n^2/2$$

for A and each right hand side respectively. Comparing with the approximately $\frac{1}{2}n^2$ flops needed to solve a triangular system we conclude that, except for very small values of n, *the reduction of A to triangular form dominates the work.*[12] (see Sections 7.4 and 7.8, respectively). Operation counts like these are meant only as a rough appraisal of the work and one should not assign too much meaning to their precise value. On modern computer architectures the rate of transfer of data between different levels of memory often limits the actual performance.

Algorithm 7.2.2 for Gaussian Elimination algorithm has three nested loops. It is possible to reorder these loops in $3 \cdot 2 \cdot 1 = 6$ ways. In each of those version the operations does the basic operation

$$a_{ij}^{(k+1)} := a_{ij}^{(k)} - a_{kj}^{(k)} a_{ik}^{(k)} / a_{kk}^{(k)},$$

and only the ordering in which they are done differs. The version given above uses row operations and may be called the "kij" variant, where k refers to step number, i to row index, and j to column index. This version is not suitable for Fortran 77, and other languages in which matrix elements are stored and accessed sequentially by columns. In such a language the form "kji" should be preferred, which is the column oriented variant of Algorithm 7.2.2 (see Problem 5).

We now show that Gaussian elimination can be interpreted as computing the matrix factorization $A = LU$. The LU factorization is a prime example of the

[12] This conclusion is not in general true for banded and sparse systems

decompositional approach to matrix computation. This approach came into favor in the 1950s and early 1960s and has been named as one of the ten algorithms with most influence on science and engineering in the 20th century.

For simplicity we assume that $m = n$ and that GE can be carried out without pivoting. We will show that in this case GE provides a factorization of A into the product of a unit lower triangular matrix L and an upper triangular matrix U. Depending on whether the element a_{ij} lies on or above or below the principal diagonal we have

$$a_{ij}^{(n)} = \begin{cases} \ldots = a_{ij}^{(i+1)} = a_{ij}^{(i)}, & i \le j; \\ \ldots = a_{ij}^{(j+1)} = 0, & i > j. \end{cases}$$

Thus in GE the elements a_{ij}, $1 \le i, j \le n$, are transformed according to

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} a_{kj}^{(k)}, \quad k = 1 : p, \quad p = \min(i-1, j). \tag{7.2.8}$$

If these equations are summed for $k = 1 : p$, we obtain

$$\sum_{k=1}^{p} (a_{ij}^{(k+1)} - a_{ij}^{(k)}) = a_{ij}^{(p+1)} - a_{ij} = -\sum_{k=1}^{p} l_{ik} a_{kj}^{(k)}.$$

This can also be written

$$a_{ij} = \begin{cases} a_{ij}^{(i)} + \displaystyle\sum_{k=1}^{i-1} l_{ik} a_{kj}^{(k)}, & i \le j; \\ 0 + \displaystyle\sum_{k=1}^{j} l_{ik} a_{kj}^{(k)}, & i > j, \end{cases}$$

or, if we define $l_{ii} = 1$, $i = 1 : n$,

$$a_{ij} = \sum_{k=1}^{r} l_{ik} u_{kj}, \quad u_{kj} = a_{kj}^{(k)}, \quad r = \min(i, j). \tag{7.2.9}$$

However, these equations are equivalent to the matrix equation $A = LU$, where $L = (l_{ik})$ and $U = (u_{kj})$ are lower and upper triangular matrices, respectively. Hence GE computes a factorization of A into a product of a lower and an upper triangular matrix, the **LU factorization** of A.

It was shown in Section 7.2.2 that if A is nonsingular, then Gaussian elimination can always be carried through provided row interchanges are allowed. Also, such row interchanges are in general needed to ensure the numerical stability of Gaussian elimination. We now consider how the LU factorization has to be modified when such interchanges are incorporated.

Row interchanges and row permutations can be expressed as pre-multiplication with certain matrices, which we now introduce. A matrix

$$I_{ij} = (\ldots, e_{i-1}, e_j, e_{i+1}, \ldots, e_{j-1}, e_i, e_{j+1}),$$

which is equal to the identity matrix except that columns i and j have been interchanged is called a **transposition matrix**. If a matrix A is premultiplied by I_{ij} this results in the interchange of *rows* i and j. Similarly post-multiplication results in the interchange of *columns* i and j. $I_{ij}^T = I_{ij}$, and by its construction it immediately follows that $I_{ij}^2 = I$ and hence $I_{ij}^{-1} = I_{ij}$.

A **permutation matrix** $P \in \mathbf{R}^{n\times n}$ is a matrix whose columns are a permutation of the columns of the unit matrix, that is,

$$P = (e_{p_1}, \ldots, e_{p_n}),$$

where $(p_1, \ldots, p_n)$ is a permutation of $(1, \ldots, n)$. Notice that in a permutation matrix every row and every column contains just one unity element. The transpose P^T of a permutation matrix is therefore again a permutation matrix Since P is uniquely represented by the integer vector $(p_1, \ldots, p_n)$ it need never be explicitly stored.

If P is a permutation matrix then PA is the matrix A with its rows permuted and AP is A with its columns permuted. Any permutation may be expressed as a sequence of transposition matrices. Therefore any permutation matrix can be expressed as a product of transposition matrices $P = I_{i_1,j_1} I_{i_2,j_2} \cdots I_{i_k,j_k}$. Since $I_{i_p,j_p}^{-1} = I_{i_p,j_p}$, we have

$$P^{-1} = I_{i_k,j_k} \cdots I_{i_2,j_2} I_{i_1,j_1} = P^T,$$

that is permutation matrices are orthogonal and P^T effects the reverse permutation.

Assume that in the kth step, $k = 1 : n-1$, we select the pivot element from row p_k, and interchange the rows k and p_k. Notice that in these row interchanges also previously computed multipliers l_{ij} must take part. At completion of the elimination, we have obtained lower and upper triangular matrices L and U. We now make the important observation that these are the same triangular factors that are obtained if we *first* carry out the row interchanges $k \leftrightarrow p_k$, $k = 1 : n-1$, on the *original matrix* A to get a matrix PA, where P is a permutation matrix, and then perform Gaussian elimination on PA *without any interchanges*. This means that Gaussian elimination with row interchanges computes the LU factors of the matrix PA. We now summarize the results and prove the uniqueness of the LU factorization:

Theorem 7.2.1. *The LU factorization*

Let $A \in \mathbf{R}^{n\times n}$ be a given nonsingular matrix. Then there is a permutation matrix P such that Gaussian elimination on the matrix $\tilde{A} = PA$ can be carried out without pivoting giving the factorization

$$PA = LU, \tag{7.2.10}$$

where $L = (l_{ij})$ is a unit lower triangular matrix and $U = (u_{ij})$ an upper triangular matrix. The elements in L and U are given by

$$u_{ij} = \tilde{a}_{ij}^{(i)}, \quad 1 \le i \le j \le n,$$

and

$$l_{ij} = \tilde{l}_{ij}, \quad l_{ii} = 1, \quad 1 \le j < i \le n,$$

where $\tilde{l}_{ij}$ are the multipliers occurring in the reduction of $\tilde{A} = PA$. For a fixed permutation matrix P, this factorization is uniquely determined.

Proof. We prove the uniqueness. Suppose we have two factorizations

$$PA = L_1U_1 = L_2U_2.$$

Since PA is nonsingular so are the factors, and it follows that $L_2^{-1}L_1 = U_2U_1^{-1}$. The left-hand matrix is the product of two unit lower triangular matrices and is therefore unit lower triangular, while the right hand matrix is upper triangular. It follows that both sides must be the identity matrix. Hence $L_2 = L_1$, and $U_2 = U_1$. □

Writing $PAx = LUx = L(Ux) = Pb$ it follows that if the LU factorization of PA is known, then the solution x can be computed by solving the two triangular systems

$$Ly = Pb, \qquad Ux = y, \tag{7.2.11}$$

which involves about $2 \cdot \frac{1}{2}n^2 = n^2$ flops.

Although the LU factorization is just a different interpretation of Gaussian elimination it turns out to have important conceptual advantages. It divides the solution of a linear system into two independent steps:

1. The factorization $PA = LU$.
2. Solution of the systems $Ly = Pb$ and $Ux = y$.

As an example of the use of the factorization consider the problem of solving the transposed system $A^Tx = b$. Since $P^TP = I$, and

$$(PA)^T = A^TP^T = (LU)^T = U^TL^T,$$

we have that $A^TP^TPx = U^T(L^TPx) = b$. It follows that $\tilde{x} = Px$ can be computed by solving the two triangular systems

$$U^Tc = b, \qquad L^T\tilde{x} = c. \tag{7.2.12}$$

We then obtain $x = P^{-1}\tilde{x}$ by applying the interchanges $k \leftrightarrow p_k$, in reverse order $k = n-1 : 1$ to $\tilde{x}$. Note that it is not at all trivial to derive this algorithm from the presentation of Gaussian elimination in the previous section!

In the general case when $A \in \mathbf{R}^{m\times n}$ of $\operatorname{rank}(A) = r \le \min\{m, n\}$, it can be shown that matrix $P_rAP_c \in \mathbf{R}^{m\times n}$ can be factored into a product of a unit lower **trapezoidal matrix** $L \in \mathbf{R}^{m\times r}$ and an upper trapezoidal matrix $U \in \mathbf{R}^{r\times n}$. Here P_r and P_c are permutation matrices performing the necessary row and column permutations, respectively. The factorization can be written in block form as

$$P_rAP_c = LU = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \end{pmatrix}, \tag{7.2.13}$$

where the matrices L_{11} and U_{11} are triangular and non-singular. Note that the block L_{21} is empty if the matrix A has full row rank, i.e. $r = m$; the block U_{12} is empty if the matrix A has full column rank, i.e. $r = n$.

To solve the system

$$P_r A P_c (P_c^T x) = LU\tilde{x} = P_r b = \tilde{b}, \quad x = P_c \tilde{x},$$

using this factorization we set $y = Ux$ and consider

$$\begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} y = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix}.$$

This uniquely determines y as the solution to $L_{11}y = \tilde{b}_1$. Hence *the system is consistent if and only if* $L_{21}y = \tilde{b}_2$. Further, we have $U\tilde{x} = y$, or

$$\begin{pmatrix} U_{11} & U_{12} \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \end{pmatrix} = y.$$

For an arbitrary $\tilde{x}_2$ this system uniquely determines $\tilde{x}_1$ as the solution to the triangular system

$$U_{11}\tilde{x}_1 = y - U_{12}\tilde{x}_2.$$

Thus, if consistent the system has a unique solution only if A has full column rank.

7.2.3 Elementary Elimination Matrices

The reduction of a matrix to triangular form by Gaussian elimination can be expressed entirely in matrix notations using **elementary elimination matrices**. This way of looking at Gaussian elimination, first systematically exploited by J. H. Wilkinson,[13] has the advantage that it suggests ways of deriving other matrix factorization

Elementary elimination matrices are lower triangular matrices of the form

$$L_j = I + l_j e_j^T = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l_{j+1,j} & 1 & \\ & & \vdots & & \ddots \\ & & l_{n,j} & & 1 \end{pmatrix}, \tag{7.2.14}$$

where only the elements *below* the main diagonal in the jth column differ from the

[13] James Hardy Wilkinson (1919–1986) English mathematician graduated from Trinity College, Cambridge. He became Alan Turing's assistant at the National Physical Laboratory in London in 1946, where he worked on the ACE computer project. He did pioneering work on numerical methods for solving linear systems and eigenvalue problems and developed software and libraries of numerical routines.

unit matrix. If a vector x is premultiplied by L_j we get

$$L_j x = (I + l_j e_j^T)x = x + l_j x_j = \begin{pmatrix} x_1 \\ \vdots \\ x_j \\ x_{j+1} + l_{j+1,j} x_j \\ \vdots \\ x_n + l_{n,j} x_j \end{pmatrix},$$

i.e., to the last $n - j$ components of x are *added* multiples of the component x_j. Since $e_j^T l_j = 0$ it follows that

$$(I - l_j e_j^T)(I + l_j e_j^T) = I + l_j e_j^T - l_j e_j^T - l_j (e_j^T l_j) e_j^T = I$$

so we have

$$L_j^{-1} = I - l_j e_j^T.$$

The computational significance of elementary elimination matrices is that they can be used to introduce zero components in a column vector x. Assume that $e_k^T x = x_k \neq 0$. We show that there is a unique elementary elimination matrix $L_k^{-1} = I - l_k e_k^T$ such that

$$L_k^{-1}(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)^T = (x_1, \ldots, x_k, 0, \ldots, 0)^T.$$

Since the last $n-k$ components of $L_k^{-1}x$ are to be zero it follows that we must have $x_i - l_{i,k} x_k = 0$, $i = k+1 : n$, and hence

$$l_k = (0, \ldots, 0, x_{k+1}/x_k, \ldots, x_n/x_k)^T.$$

The product of two elementary elimination matrices $L_j L_k$ is a lower triangular matrix which differs from the unit matrix in the two columns j and k below the main diagonal,

$$L_j L_k = (I + l_j e_j^T)(I + l_k e_k^T) = I + l_j e_j^T + l_k e_k^T + l_j (e_j^T l_k) e_k^T.$$

If $j \leq k$, then $e_j^T l_k = 0$, and the following simple multiplication rule holds:

$$L_j L_k = I + l_j e_j^T + l_k e_k^T, \quad j \leq k. \tag{7.2.15}$$

Note that no products of the elements l_{ij} occur! However, if $j > k$, then in general $e_j^T l_k \neq 0$, and the product $L_j L_k$ has a more complex structure.

We now show that Gaussian elimination with partial pivoting can be accomplished by premultiplication of A by a sequence of elementary elimination matrices combined with transposition matrices to express the interchange of rows. For simplicity we first consider the case when $\operatorname{rank}(A) = m = n$. In the first step assume that $a_{p_1,1} \neq 0$ is the pivot element. We then interchange rows 1 and p_1 in A by premultiplication of A by a transposition matrix,

$$\tilde{A} = P_1 A, \quad P_1 = I_{1,p_1}.$$

If we next premultiply $\tilde{A}$ by the elementary elimination matrix

$$L_1^{-1} = I - l_1 e_1^T, \quad l_{i1} = \tilde{a}_{i1}/\tilde{a}_{11}, \quad i = 2:n,$$

this will zero out the elements under the main diagonal in the first column, i.e.

$$A^{(2)} e_1 = L_1^{-1} P_1 A e_1 = \tilde{a}_{11} e_1.$$

All remaining elimination steps are similar to this first one. The second step is achieved by forming $\tilde{A}^{(2)} = P_2 A^{(2)}$ and

$$A^{(3)} = L_2^{-1} P_2 A^{(2)} = L_2^{-1} P_2 L_1^{-1} P_1 A.$$

Here $P_2 = I_{2,p_2}$, where $a_{p_2,2}^{(2)}$ is the pivot element from the second column and $L_2^{-1} = I - l_2 e_2^T$ is an elementary elimination matrix with nontrivial elements equal to $l_{i2} = \tilde{a}_{i2}^{(2)}/\tilde{a}_{22}^{(2)}$, $i = 3:n$. Continuing, we have after $n-1$ steps reduced A to upper triangular form

$$U = L_{n-1}^{-1} P_{n-1} \cdots L_2^{-1} P_2 L_1^{-1} P_1 A. \tag{7.2.16}$$

To see that (7.2.16) is equivalent with the LU factorization of PA we first note that since $P_2^2 = I$ we have after the first two steps that

$$A^{(3)} = L_2^{-1} \tilde{L}_1^{-1} P_2 P_1 A$$

where

$$\tilde{L}_1^{-1} = P_2 L_1^{-1} P_2 = I - (P_2 l_1)(e_1^T P_2) = I - \tilde{l}_1 e_1^T.$$

Hence $\tilde{L}_1^{-1}$ is again an elementary elimination matrix of the same type as L_1^{-1}, except that two elements in l_1 have been interchanged. Premultiplying by $\tilde{L}_1 L_2$ we get

$$\tilde{L}_1 L_2 A^{(3)} = P_2 P_1 A,$$

where the two elementary elimination matrices on the left hand side combine trivially. Proceeding in a similar way it can be shown that (7.2.16) implies

$$\tilde{L}_1 \tilde{L}_2 \ldots \tilde{L}_{n-1} U = P_{n-1} \ldots P_2 P_1 A,$$

where $\tilde{L}_{n-1} = L_{n-1}$ and

$$\tilde{L}_j = I + \tilde{l}_j e_j^T, \qquad \tilde{l}_j = P_{n-1} \cdots P_{j+1} l_j, \quad j = 1:n-2.$$

Using the result in (7.2.15), the elimination matrices can trivially be multiplied together and it follows that

$$PA = LU, \quad P = P_{n-1} \cdots P_2 P_1,$$

where the elements in L are given by $l_{ij} = \tilde{l}_{ij}$, $l_{ii} = 1$, $1 \le j < i \le n$. This is the LU factorization of Theorem 7.2.1. It is important to note that nothing new, except the notations, has been introduced. In particular, the transposition matrices and

elimination matrices used here are, of course, never explicitly stored in a computer implementation.

In Gaussian elimination we use in the kth step the pivot row to eliminate elements *below* the main diagonal in column k. In **Gauss–Jordan elimination**[14] the elements *above* the main diagonal are eliminated simultaneously. After $n-1$ steps the matrix A has then been transformed into a *diagonal* matrix containing the nonzero pivot elements. Gauss–Jordan elimination was used in many early versions of linear programming and also for implementing stepwise regression in statistics.

Gauss–Jordan elimination can be described by introducing the elementary matrices

$$M_j = \begin{pmatrix} 1 & & & l_{1j} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & l_{j-1,j} & & & \\ & & & 1 & & & \\ & & & l_{j+1,j} & 1 & & \\ & & & \vdots & & \ddots & \\ & & & l_{n,j} & & & 1 \end{pmatrix}. \tag{7.2.17}$$

If partial pivoting is carried out we can write, cf. (7.2.16)

$$D = M_n M_{n-1}^{-1} P_{n-1} \cdots M_2^{-1} P_2 M_1^{-1} P_1 A,$$

where the l_{ij} are chosen to annihilate the (i,j)th element. Multiplying by D^{-1} we get

$$A^{-1} = D^{-1} M_n M_{n-1}^{-1} P_{n-1} \cdots M_2^{-1} P_2 M_1^{-1} P_1. \tag{7.2.18}$$

This expresses the inverse of A as a product of elimination and transposition matrices, and is called the **product form of the inverse**. The operation count for this elimination process is $\approx n^3/2$ flops, i.e., higher than for the LU factorization by Gaussian elimination. For some parallel implementations Gauss–Jordan elimination may still have advantages.

To solve a linear system $Ax = b$ we apply these transformations to the vector b to obtain

$$x = A^{-1}b = D^{-1} M_{n-1}^{-1} P_{n-1} \cdots M_2^{-1} P_2 M_1^{-1} P_1 b. \tag{7.2.19}$$

This requires n^2 flops. Note that no back-substitution is needed!

The stability of Gauss–Jordan elimination has been analyzed by Peters and Wilkinson [53]. They remark that the residuals $b - A\bar{x}$ corresponding to the Gauss–Jordan solution $\bar{x}$ can be a larger by a factor $\kappa(A)$ than those corresponding to the solution by Gaussian elimination. Although the method is not backward stable in general it can be shown to be stable for so called diagonally dominant matrices (see Definition 7.2.4). It is also forward stable, i.e., will give about the same numerical accuracy in the computed solution $\bar{x}$ as Gaussian elimination.

[14] Named after Wilhelm Jordan (1842–1899), who used this method to compute the covariance matrix in least squares problems.

7.2.4 Pivoting Strategies

We saw that in Gaussian elimination row and column interchanges were needed in case a zero pivot was encountered. A basic rule of numerical computation says that if an algorithm breaks down when a zero element is encountered, then we can expect some form of instability and loss of precision also for nonzero but small elements! Again, this is related to the fact that in floating point computation the difference between a zero and nonzero number becomes fuzzy because of the effect of rounding errors.

Example 7.2.2. For $\epsilon \neq 1$ the system

$$\begin{pmatrix} \epsilon & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

is nonsingular and has the unique solution $x_1 = -x_2 = -1/(1-\epsilon)$. Suppose $\epsilon = 10^{-6}$ is accepted as pivot in Gaussian elimination. Multiplying the first equation by 10^6 and subtracting from the second we obtain $(1-10^6)x_2 = -10^6$. By rounding this could give $x_2 = 1$, which is correct to six digits. However, back-substituting to obtain x_1 we get $10^{-6}x_1 = 1-1$, or $x_1 = 0$, which is completely wrong.

The simple example above illustrates that in general it is necessary to perform row (and/or column) interchanges *not only when a pivotal element is exactly zero, but also when it is small*. The two most common pivoting strategies are **partial pivoting** and **complete pivoting**. In partial pivoting the pivot is taken as the largest element in magnitude in the unreduced part of the kth column. In complete pivoting the pivot is taken as the largest element in magnitude in the whole unreduced part of the matrix.

Partial Pivoting. At the start of the kth stage choose interchange rows k and r, where r is the smallest integer for which

$$|a_{rk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|. \tag{7.2.20}$$

Complete Pivoting. At the start of the kth stage interchange rows k and r and columns k and s, where r and s are the smallest integers for which

$$|a_{rs}^{(k)}| = \max_{k \leq i,j \leq n} |a_{ij}^{(k)}|. \tag{7.2.21}$$

Complete pivoting requires $O(n^3)$ in total compared with only $O(n^2)$ for partial pivoting. Hence, complete pivoting involves a fairly high overhead since about as many arithmetic comparisons as floating point operations has to be performed. Since practical experience shows that partial pivoting works well, this is the standard choice. Note, however, that when $\operatorname{rank}(A) < n$ then complete pivoting must be used

A major breakthrough in the understanding of GE came with the famous backward rounding error analysis of Wilkinson [65, 1961]. Using the standard model for

floating point computation Wilkinson showed that the computed triangular factors $\bar{L}$ and $\bar{U}$ of A, obtained by Gaussian elimination *are the exact triangular factors of a perturbed matrix*

$$\bar{L}\bar{U} = A + E, \quad E = (e_{ij})$$

where, since e_{ij} is the sum of $\min(i-1, j)$ rounding errors

$$|e_{ij}| \leq 3u \min(i-1, j) \max_k |\bar{a}_{ij}^{(k)}|. \tag{7.2.22}$$

Note that the above result holds without any assumption about the size of the multipliers. *This shows that the purpose of any pivotal strategy is to avoid growth in the size of the computed elements $\bar{a}_{ij}^{(k)}$, and that the size of the multipliers is of no consequence* (see the remark on possible large multipliers for positive-definite matrices, Section 7.4.2).

The growth of elements during the elimination is usually measured by the **growth ratio**.

Definition 7.2.2.

Let $a_{ij}^{(k)}$, $k = 2 : n$, be the elements in the kth stage of Gaussian elimination applied to the matrix $A = (a_{ij})$. Then the **growth ratio** *in the elimination is*

$$\rho_n = \max_{i,j,k} |a_{ij}^{(k)}| / \max_{i,j} |a_{ij}|. \tag{7.2.23}$$

It follows that $E = (e_{ij})$ can be bounded component-wise by

$$|E| \leq 3\rho_n u \max_{ij} |a_{ij}| F. \tag{7.2.24}$$

where F the matrix with elements $f_{i,j} = \min\{i-1, j\}$. Strictly speaking this is not correct unless we use the growth factor $\bar{\rho}_n$ for the *computed elements*. Since this quantity differs insignificantly from the theoretical growth factor ρ_n in (7.2.23), we ignore this difference here and in the following. Slightly refining the estimate

$$\|F\|_\infty \leq (1 + 2 + \cdots + n) - 1 \leq \frac{1}{2} n(n+1) - 1$$

and using $\max_{ij} |a_{ij}| \leq \|A\|_\infty$, we get the normwise backward error bound:

Theorem 7.2.3.

Let $\bar{L}$ and $\bar{U}$ be the computed triangular factors of A, obtained by GE with floating-point arithmetic with unit roundoff u has been used, there is a matrix E such that

$$\bar{L}\bar{U} = A + E, \quad \|E\|_\infty \leq 1.5n^2 \rho_n u \|A\|_\infty. \tag{7.2.25}$$

If pivoting is employed so that the computed multipliers satisfy the inequality

$$|l_{ik}| \leq 1, \quad i = k+1 : n.$$

Then it can be shown that an estimate similar to (7.2.25) holds with the constant 1 instead of 1.5. For both partial and complete pivoting it holds that

$$|a_{ij}^{(k+1)}| < |a_{ij}^{(k)}| + |l_{ik}||a_{kj}^{(k)}| \leq |a_{ij}^{(k)}| + |a_{kj}^{(k)}| \leq 2 \max_{i,j} |a_{ij}^{(k)}|,$$

and the bound $\rho_n \leq 2^{n-1}$ follows by induction. For partial pivoting this bound is the best possible, and can be attained for special matrices. For complete pivoting a much better bound can be proved, and in practice the growth very seldom exceeds n; see Section 7.6.2.

A pivoting scheme that gives a pivot of size between that of partial and complete pivoting is **rook pivoting**. In this scheme we pick a pivot element which is largest in magnitude in *both its column and its row.*

Rook Pivoting. At the start of the kth stage rows k and r and columns k and s are interchanged, where

$$|a_{rs}^{(k)}| = \max_{k \leq i \leq n} |a_{ij}^{(k)}| = \max_{k \leq j \leq n} |a_{ij}^{(k)}|. \tag{7.2.26}$$

We start by finding the element of maximum magnitude in the first column. If this element is also of maximum magnitude in its row we accept it as pivot. Otherwise we compare the element of maximum magnitude in the row with other elements in its column, etc. The name derives from the fact that the pivot search resembles the moves of a rook in chess; see Figure 7.2.1..

1	10	1	2	4
0	5	2	9	8
3	0	4	1	7
2	2	5	6	1
1	4	3	2	3

Figure 7.2.1. *Illustration of rook pivoting in a* 5×5 *matrix with positive integer entries as shown. The* $(2, 4)$ *element 9 is chosen as pivot.*

Rook pivoting involves at least twice as many comparisons as partial pivoting. In the worst case it can take $O(n^3)$ comparisons, i.e., the same order of magnitude as for complete pivoting. Numerical experience shows that the cost of rook pivoting usually equals a small multiple of the cost for partial pivoting. A pivoting related to rook pivoting is used in the solution of symmetric indefinite systems; see Sec. 7.3.4.

It is important to realize that the choice of pivots is influenced by the scaling of equations and unknowns. If, for example, the unknowns are physical quantities a different choices of units will correspond to a different scaling of the unknowns and the columns in A. Partial pivoting has the important property of being invariant

under column scalings. In theory we could perform partial pivoting by *column* interchanges, which then would be invariant under row scalings. but in practice this turns out to be less satisfactory. Likewise, an unsuitable column scaling can also make complete pivoting behave badly.

For certain important classes of matrices a bound independent of n can be given for the growth ratio in Gaussian elimination without pivoting or with partial pivoting. For these Gaussian elimination is backward stable.

- If A is real symmetric matrix $A = A^T$ and positive definite (i.e. $x^T Ax > 0$ for all $x \neq 0$) then $\rho_n(A) \leq 1$ with no pivoting (see Theorem 7.3.7).
- If A is row or column diagonally dominant then $\rho_n \leq 2$ with no pivoting.
- If A is Hessenberg then $\rho_n \leq n$ with partial pivoting.
- If A is tridiagonal then $\rho_n \leq 2$ with partial pivoting.

For the last two cases we refer to Sec. 7.4. We now consider the case when A is diagonally dominant.

Definition 7.2.4. *A matrix A is said to be* **diagonally dominant** *by rows, if*

$$\sum_{j \neq i} |a_{ij}| \leq |a_{ii}|, \quad i = 1 : n. \tag{7.2.27}$$

A is **diagonally dominant** *by columns if A^T is diagonally dominant by rows.*

Theorem 7.2.5.

Let A be nonsingular and diagonally dominant by rows or columns. Then A has an LU factorization without pivoting and the growth ratio $\rho_n(A) \leq 2$. If A is diagonally dominant by columns, then the multipliers in this LU factorization satisfy $|l_{ij}| \leq 1$, for $1 \leq j < i \leq n$.

Proof. (Wilkinson [65, pp.288–289])

Assume that A is nonsingular and diagonally dominant by columns. Then $a_{11} \neq 0$, since otherwise the first column would be zero and A singular. In the first stage of Gaussian elimination without pivoting we have Hence

$$a_{ij}^{(2)} = a_{ij} - l_{i1} a_{1j}, \quad l_{i1} = a_{i1}/a_{11}, \quad i, j \geq 2, \tag{7.2.28}$$

where

$$\sum_{i=2}^{n} |l_{i1}| \leq \sum_{i=2}^{n} |a_{i1}|/|a_{11}| \leq 1. \tag{7.2.29}$$

For $j = i$, using the definition and (7.2.29), it follows that

$$\begin{aligned} |a_{ii}^{(2)}| \geq |a_{ii}| - |l_{i1}|\,|a_{1i}| &\geq \sum_{j \neq i} |a_{ji}| - \Big(1 - \sum_{j \neq 1,i} |l_{j1}|\Big)|a_{1i}| \\ &= \sum_{j \neq 1,i} \big(|a_{ji}| + |l_{j1}||a_{1i}|\big) \geq \sum_{j \neq 1,i} |a_{ji}^{(2)}|. \end{aligned}$$

Hence the reduced matrix $A^{(2)} = (a_{ij}^{(2)})$, is also nonsingular and diagonally dominant by columns. It follows by induction that all matrices $A^{(k)} = (a_{ij}^{(k)})$, $k = 2 : n$ are nonsingular and diagonally dominant by columns.

Further using (7.2.28) and (7.2.29), for $i \geq 2$,

$$\sum_{i=2}^{n} |a_{ij}^{(2)}| \leq \sum_{i=2}^{n} \left(|a_{ij}| + |l_{i1}||a_{1j}| \right) \leq \sum_{i=2}^{n} |a_{ij}| + |a_{1j}| \sum_{i=2}^{n} |l_{i1}|$$
$$\leq \sum_{i=2}^{n} |a_{ij}| + |a_{1j}| = \sum_{i=1}^{n} |a_{ij}|.$$

Hence the sum of the moduli of the elements of any column of $A^{(k)}$ does not increase as k increases. Hence

$$\max_{i,j,k} |a_{ij}^{(k)}| \leq \max_{i,k} \sum_{j=k}^{n} |a_{ij}^{(k)}| \leq \max_{i} \sum_{j=1}^{n} |a_{ij}| \leq 2 \max_{i} |a_{ii}| = 2 \max_{ij} |a_{ij}|.$$

It follows that

$$\rho_n = \max_{i,j,k} |a_{ij}^{(k)}| / \max_{i,j} |a_{ij}| \leq 2.$$

The proof for matrices which are *row* diagonally dominant is similar. (Notice that Gaussian elimination with pivoting essentially treats rows and columns symmetrically!) □

We conclude that for a row or column diagonally dominant matrix Gaussian elimination without pivoting is backward stable. If A is diagonally dominant by rows then the multipliers can be arbitrarily large, but this does not affect the stability.

If (7.2.27) holds with strict inequality for all i, then A is said to be **strictly diagonally dominant** by rows. If A is strictly diagonally dominant, then it can be shown that all reduced matrices have the same property. In particular, all pivot elements must then be strictly positive and the nonsingularity of A follows. We mention a useful result for strictly diagonally dominant matrices.

Lemma 7.2.6.

Let A be strictly diagonally dominant by rows, and set

$$\alpha = \min_i \alpha_i, \quad \alpha_i := |a_{ii}| - \sum_{j \neq i} |a_{ij}| > 0, \quad i = 1 : n. \tag{7.2.30}$$

Then A is nonsingular, and $\|A^{-1}\|_\infty \leq \alpha^{-1}$.

Proof. By the definition of a subordinate norm (7.1.50) we have

$$\frac{1}{\|A^{-1}\|_\infty} = \inf_{y \neq 0} \frac{\|y\|_\infty}{\|A^{-1}y\|_\infty} = \inf_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \min_{\|x\|_\infty = 1} \|Ax\|_\infty.$$

Assume that equality holds in (7.2.30) for $i = k$. Then

$$\frac{1}{\|A^{-1}\|_\infty} = \min_{\|x\|_\infty=1} \max_i \Big| \sum_j a_{ij}x_j \Big| \geq \min_{\|x\|_\infty=1} \Big| \sum_j a_{kj}x_j \Big|$$
$$\geq |a_{kk}| - \sum_{j,j\neq k} |a_{kj}| = \alpha.$$

□

If A is strictly diagonally dominant by columns, then since $\|A\|_1 = \|A^T\|_\infty$ it holds that $\|A^{-1}\|_1 \leq \alpha^{-1}$. If A is strictly diagonally dominant in both rows and columns, then from $\|A\|_2 \leq \sqrt{\|A\|_1\|A\|_\infty}$ it follows that $\|A^{-1}\|_2 \leq \alpha^{-1}$.

7.2.5 Computational Variants

In Gaussian Elimination, as described in Sec. 7.2.3, the kth step consists of modifying the unreduced part of the matrix by an *outer product* of the vector of multipliers and the pivot row. Using the equivalence of Gaussian elimination and LU factorization it is easy to see that the the calculations can be arranged in several diferent ways so that the elements in L and U are determined directly.

For simplicity, we first assume that any row or column interchanges on A have been carried out in advance. The matrix equation $A = LU$ written in componentwise form (see (7.2.9))

$$a_{ij} = \sum_{k=1}^{r} l_{ik}u_{kj}, \quad 1 \leq i, j \leq n, \quad r = \min(i, j),$$

together with the normalization conditions $l_{ii} = 1$, $i = 1 : n$, can be thought of as $n^2 + n$ equations for the $n^2 + n$ unknown elements in L and U. We can solve these equations in n steps, $k = 1 : n$, where in the kth step we use the equations

$$a_{kj} = \sum_{p=1}^{k} l_{kp}u_{pj}, \quad j \geq k, \qquad a_{ik} = \sum_{p=1}^{k} l_{ip}u_{pk}, \quad i > k \tag{7.2.31}$$

to determine the kth *row* of U and the kth *column* of L. In this algorithm the main work is performed in matrix-vector multiplications.

Algorithm 7.2.3 Doolittle's Algorithm.

for $k = 1 : n$
  for $j = k : n$
    $u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj};$
  end
  for $i = k + 1 : n$

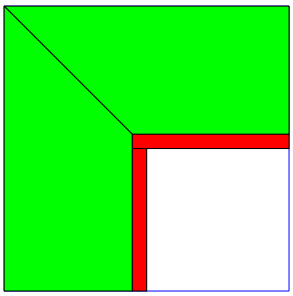


Figure 7.2.2. *Computations in the kth step of Doolittle's method.*

$$l_{ik} = \Big(a_{ik} - \sum_{p=1}^{k-1} l_{ip}u_{pk}\Big)/u_{kk};$$

end

$l_{kk} = 1;$

end

Since the LU factorization is unique this algorithm produces the same factors L and U as Gaussian elimination. In fact, successive partial sums in the equations (7.2.31) equal the elements $a_{ij}^{(k)}$, $j > k$, in Gaussian elimination. It follows that if each term in (7.2.31) is rounded separately, the compact algorithm is also *numerically equivalent* to Gaussian elimination. If the inner products can be accumulated in higher precision, then the compact algorithm is less affected by rounding errors. Algorithm 7.2.3 is usually referred to as Doolittle's algorithm. In Crout's algorithm the upper triangular matrix U is normalized to have a unit diagonal. [15]

Algorithm 7.2.5 can be modified to include partial pivoting. Changing the order of operations, we first calculate $\tilde{l}_{ik} = l_{ik}u_{kk}$, $i = k : n$, and determine the element of maximum magnitude. The corresponding row is then permuted to pivotal position. In this row exchange the already computed part of L and remaining part of A also take part. Next we normalize by setting $l_{kk} = 1$, which determines l_{ik}, $i = 1 : k$, and also u_{kk}. Finally, the remaining part of the kth row in U is computed.

It is possible to sequence the computations in Doolittle's and Crout's algorithms in many different ways. Indeed any element in $(L\backslash U)$ can be computed as soon as the corresponding elements in L to the left and in U above have been deter-

[15] In the days of hand computations these algorithms had the advantage that they did away with the necessity in Gaussian elimination to write down $\approx n^3/3$ intermediate results—one for each multiplication.

mined. For example, three possible orderings are schematically illustrated below,

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 3 & 3 & 3 & 3 \\ 2 & 4 & 5 & 5 & 5 \\ 2 & 4 & 6 & 7 & 7 \\ 2 & 4 & 6 & 8 & 9 \end{pmatrix}, \quad \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 3 & 5 & 7 & 9 \\ 2 & 4 & 5 & 7 & 9 \\ 2 & 4 & 6 & 7 & 9 \\ 2 & 4 & 6 & 8 & 9 \end{pmatrix}, \quad \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 2 & 3 & 5 & 7 & 9 \\ 4 & 4 & 5 & 7 & 9 \\ 6 & 6 & 6 & 7 & 9 \\ 8 & 8 & 8 & 8 & 9 \end{pmatrix}.$$

Here the entries indicate in which step a certain element l_{ij} and r_{ij} is computed, so the first example corresponds to the ordering in the algorithm given above. (Compare the comments after Algorithm 7.2.2.) Note that it is *not* easy to do complete pivoting with any of these variants.

The Bordering Method

Before the kth step, $k = 1 : n$, of the bordering method we have have computed the LU-factorization $A_{11} = L_{11}U_{11}$ of the leading principal submatrix A_{11} of order $k-1$ of A. To proceed we seek the LU-factorization

$$\begin{pmatrix} A_{11} & a_{1k} \\ a_{k1}^T & \alpha_{kk} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ l_{k1}^T & 1 \end{pmatrix} \begin{pmatrix} U_{11} & u_{1k} \\ 0 & u_{kk} \end{pmatrix}.$$

Identifying the (1,2)-blocks we find

$$L_{11}u_{1k} = a_{1k}, \tag{7.2.32}$$

which is a lower triangular system for u_{1k}. Identifying the (2,1)-blocks and transposing gives

$$U_{11}^T l_{k1} = a_{k1}, \tag{7.2.33}$$

another lower triangular system for l_{k1}. Finally, from the (2,2)-block we get $l_{k1}^T u_{1k} + u_{kk} = \alpha_{kk}$, or

$$u_{kk} = \alpha_{kk} - l_{k1}^T u_{1k}. \tag{7.2.34}$$

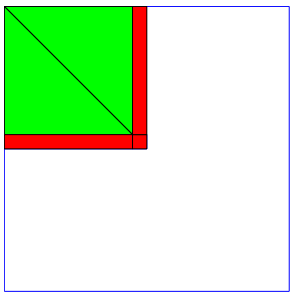


Figure 7.2.3. *Computations in the kth step of the bordering method.*

The main work in this variant is done in solving the triangular systems (7.2.32) and (7.2.33). A drawback of the bordering method is that it cannot be combined with partial pivoting.

The Sweep Methods

In the **column sweep** method at the kth step the first k columns of L and U in LU-factorization of A are computed. Assume that we have computed L_{11}, L_{21}, and U_{11} in the factorization

$$\begin{pmatrix} A_{11} & a_{1k} \\ A_{21} & a_{2k} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & l_{2k} \end{pmatrix} \begin{pmatrix} U_{11} & u_{1k} \\ 0 & u_{kk} \end{pmatrix} \in \mathbf{R}^{n\times k}.$$

As in the bordering method, identifying the (1,2)-blocks we find

$$L_{11}u_{1k} = a_{1k}, \tag{7.2.35}$$

a lower triangular system for u_{1k}. From the (2,2)-blocks we get $L_{21}u_{1k} + l_{2k}u_{kk} = a_{2k}$, or

$$l_{2k}u_{kk} = a_{2k} - L_{21}u_{1k}. \tag{7.2.36}$$

Together with the normalizing condition that the first component in the vector l_{2k} equals one this determines u_{kk} and l_{2k}.

Partial pivoting can be implemented with this method as follows. When the right hand side in (7.2.36) has been evaluated we determine the element of maximum modulus in the vector $a_{2k} - L_{21}u_{1k}$. We then permute this element to top position and perform the same row exchanges in L_{21}^T and the unprocessed part of A.

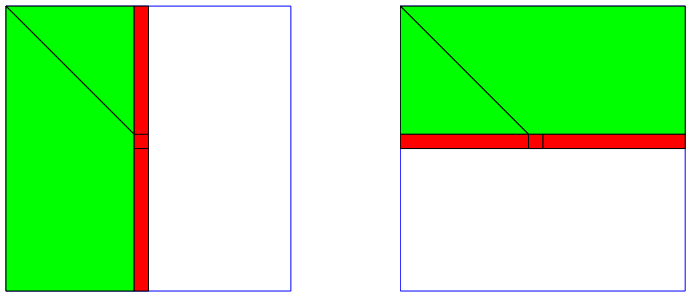


Figure 7.2.4. *Computations in the kth step of the sweep methods. Left: The column sweep method. Right: The row sweep method.*

In the column sweep method L and U are determined column by column. It possible to determine L and U row by row. In the kth step of this **row sweep** method the kth row of A is processed and we write

$$\begin{pmatrix} A_{11} & A_{12} \\ a_{k1}^T & a_{k2}^T \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ l_{k1}^T & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & u_{2k}^T \end{pmatrix} \in \mathbf{R}^{k\times n}.$$

Identifying the (2,1)- and (2,2)-blocks we get

$$U_{11}^T l_{k1} = a_{k1}, \tag{7.2.37}$$

and

$$u_{2k}^T = a_{k2}^T - l_{k1}U_{12}. \tag{7.2.38}$$

Note that Doolittle's method can be viewed as alternating between the two sweep methods.

Consider now the case when $A \in \mathbf{R}^{m\times n}$ is a rectangular matrix with rank $(A) = r = \min(m, n)$. If $m > n$ it is advantageous to process the matrix column by column. Then after n steps we have $AP_c = LU$, where L is lower trapezoidal,

$$L = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \in \mathbf{R}^{m\times n}, \tag{7.2.39}$$

and $U \in \mathbf{R}^{n\times n}$ is square upper triangular. If $m < n$ and the matrix is processed row by row, we have after n steps an LU factorization with $L \in \mathbf{R}^{m\times m}$ and

$$U = (\, U_{11} \quad U_{12}\,) \in \mathbf{R}^{m\times n}$$

upper trapezoidal.

7.2.6 Computing the Inverse

If the inverse matrix A^{-1} is known, then the solution of $Ax = b$ can be obtained through a matrix vector multiplication by $x = A^{-1}b$. This is theoretically satisfying, but in most practical computational problems it is unnecessary and inadvisable to compute A^{-1}. As succinctly expressed by G. E. Forsythe and C. B. Moler [27]:

Almost anything you can do with A^{-1} can be done without it!

The work required to compute A^{-1} is about n^3 flops, i.e., three times greater than for computing the LU factorization. (If A is a band matrix, then the savings can be much more spectacular; see Sec. 7.4.) To solve a linear system $Ax = b$ the matrix vector multiplication $A^{-1}b$ requires n^2 flops. This is exactly the same as for the solution of the two triangular systems $L(Ux) = b$ resulting from LU factorization of A. (Note, however, that on some parallel computers matrix multiplication can be performed much faster than solving triangular systems.)

One advantage of computing the inverse matrix is that A^{-1} can be used to get a strictly reliable error estimate for a computed solution $\bar{x}$. A similar estimate is not directly available from the LU factorization. However, alternative ways to obtain error estimates are the use of a condition estimator (Section 7.5.3) or iterative refinement (Section 7.6.4).

Not only is the inversion approach three times more expensive but if A is ill-conditioned the solution computed from $A^{-1}b$ usually is much less accurate than than that computed from the LU factorization. Using LU factorization the residual vector of the computed solution will usually be of order machine precision even when A is ill-conditioned.

Nevertheless, there are some applications where A^{-1} is required, e.g., in some methods for computing the matrix square root and the logarithm of a matrix; see Sec. 9.2.4. The inverse of a symmetric positive definite matrix is needed to obtain estimates of the covariances in regression analysis. However, usually only certain elements of A^{-1} are needed and not the whole inverse matrix.

We first consider computing the inverse of a lower triangular matrix L. Setting $L^{-1} = Y = (y_1, \ldots, y_n)$, we have $LY = I = (e_1, \ldots, e_n)$. This shows that the columns of Y satisfy

$$Ly_j = e_j, \quad j = 1 : n.$$

These lower triangular systems can be solved by forward substitution. Since the vector e_j has $(j-1)$ leading zeros the first $(j-1)$ components in y_j are zero. Hence L^{-1} is also a lower triangular matrix, and its elements can be computed recursively from

$$y_{jj} = 1/l_{jj}, \qquad y_{ij} = \Big(-\sum_{k=j}^{i-1} l_{ik} y_{kj} \Big) \Big/ l_{ii}, \quad i = j+1 : n, \tag{7.2.40}$$

Note that the diagonal elements in L^{-1} are just the inverses of the diagonal elements of L. If the columns are computed in the order $j = 1 : n$, then Y can overwrite L in storage.

Similarly, if U is upper triangular matrix then $Z = U^{-1}$ is an upper triangular matrix, whose elements can be computed from:

$$z_{jj} = 1/u_{jj}, \qquad z_{ij} = \Big(-\sum_{k=i+1}^{j} u_{ik} z_{kj} \Big) \Big/ u_{ii}, \quad i = j-1 : -1 : 1. \tag{7.2.41}$$

If the columns are computed in the order $j = n : -1 : 1$, the Z can overwrite U in storage. The number of flops required to compute L^{-1} or U^{-1} is approximately equal to $n^3/6$. Variants of the above algorithm can be obtained by reordering the loop indices.

Now let $A^{-1} = X = (x_1, \ldots, x_n)$ and assume that an LU factorization $A = LU$ has been computed. Then

$$Ax_j = L(Ux_j) = e_j, \quad j = 1 : n, \tag{7.2.42}$$

and the columns of A^{-1} are obtained by solving n linear systems, where the right hand sides equal the columns in the unit matrix. Setting (7.2.42) is equivalent to

$$Ux_j = y_j, \qquad Ly_j = e_j, \quad j = 1 : n. \tag{7.2.43}$$

This method for inverting A requires $n^3/6$ flops for inverting L and $n^3/2$ flops for solving the n upper triangular systems giving a total of n^3 flops.

A second method uses the relation

$$A^{-1} = (LU)^{-1} = U^{-1}L^{-1}. \tag{7.2.44}$$

Since the matrix multiplication $U^{-1}L^{-1}$ requires $n^3/3$ flops (show this!) the total work to compute A^{-1} by the second method method (7.2.44) also is n^3 flops. If we take advantage of that $y_{jj} = 1/l_{jj} = 1$, and carefully sequence the computations then L^{-1}, U^{-1} and finally A^{-1} can overwrite A so that no extra storage is needed.

There are many other varaints of computing the inverse $X = A^{-1}$. From $XA = I$ we have

$$XLU = I \quad \text{or} \quad XL = U^{-1}.$$

In the MATLAB function inv(A), U^{-1} is first computed by a column oriented algorithm. Then the system $XL = U^{-1}$ is solved for X. The stability properties of this and several other different matrix inversion algorithms are analyzed in [21]; see also Higham [41, Sec. 14.2].

The inverse can also be obtained from the Gauss–Jordan factorization. Using (7.2.19) where b is taken to be the columns of the unit matrix, we compute

$$A^{-1} = D^{-1}M_{n-1}^{-1}P_{n-1}\cdots M_2^{-1}P_2M_1^{-1}P_1(e_1,\ldots,e_n).$$

Again n^3 flops are required if the computations are properly organized. The method can be arranged so that the inverse emerges in the original array. However, the numerical properties of this method are not as good as for the methods described above.

If row interchanges have been performed during the LU factorization, we have $PA = LU$, where $P = P_{n-1}\cdots P_2P_1$ and P_k are transposition matrices. Then $A^{-1} = (LU)^{-1}P$. Hence we obtain A^{-1} by performing the interchanges in *reverse order on the columns* of $(LU)^{-1}$.

An approximative inverse of a matrix $A = I - B$ can sometimes be computed from a matrix series expansion. To derive this we form the product

$$(I-B)(I+B+B^2+B^3+\cdots+B^k) = I - B^{k+1}.$$

Suppose that $\|B\| < 1$ for some matrix norm. Then it follows that

$$\|B^{k+1}\| \le \|B\|^{k+1} \to 0, \quad k \to \infty,$$

and hence the **Neumann expansion**

$$(I-B)^{-1} = I + B + B^2 + B^3 + \cdots, \tag{7.2.45}$$

converges to $(I-B)^{-1}$. (Note the similarity with the Maclaurin series for $(1-x)^{-1}$.) Alternatively one can use the more rapidly converging **Euler expansion**

$$(I-B)^{-1} = (I+B)(I+B^2)(I+B^4)\cdots. \tag{7.2.46}$$

It can be shown by induction that

$$(I+B)(I+B^2)\cdots(I+B^{2^k}) = I + B + B^2 + B^3 + \cdots B^{2^{k+1}}.$$

Finally we mention an iterative method for computing the inverse, the **Newton–Schultz iteration**

$$X_{k+1} = X_k(2I - AX_k) = (2I - AX_k)X_k. \tag{7.2.47}$$

This is an analogue to the iteration $x_{k+1} = x_k(2 - ax_k)$, for computing the inverse of a scalar. It can be shown that if $X_0 = \alpha_0 A^T$ and $0 < \alpha_0 < 2/\|A\|_2^2$, then $\lim_{k\to\infty} X_k = A^{-1}$. Convergence can be slow initially but ultimately quadratic,

$$E_{k+1} = E_k^2, \quad E_k = I - AX_k \ \text{ or } \ I - X_kA.$$

Since about $2\log_2 \kappa_2(A)$ (see [59]) iterations are needed for convergence it cannot in general compete with direct methods for dense matrices. However, a few steps of the iteration (7.2.47) can be used to improve an approximate inverse.

Review Questions

1. How many operations are needed (approximately) for
   (a) The LU factorization of a square matrix?
   (b) The solution of $Ax = b$, when the triangular factorization of A is known?
2. Show that if the kth diagonal entry of an upper triangular matrix is zero, then its first k columns are linearly dependent.
3. What is meant by partial and complete pivoting in Gaussian elimination? Mention two classes of matrices for which Gaussian elimination can be performed stably without any pivoting?
4. What is the LU-decomposition of an n by n matrix A, and how is it related to Gaussian elimination? Does it always exist? If not, give sufficient conditions for its existence.
5. How is the LU-decomposition used for solving a linear system? What are the advantages over using the inverse of A? Give an approximate operation count for the solution of a dense linear system with p different right hand sides using the LU-decomposition.
6. Let B be a strictly lower or upper triangular matrix. Prove that the Neumann and Euler expansions for $(I - L)^{-1}$ are finite.

Problems

1. (a) Compute the LU factorization of A and $\det(A)$, where

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \\ 1 & 16 & 81 & 256 \end{pmatrix}.$$

   (b) Solve the linear system $Ax = b$, where $b = (2, 10, 44, 190)^T$.
2. (a) Show that $P = (e_n, \ldots, e_2, e_1)$ is a permutation matrix and that $P = P^T = P^{-1}$, and that Px reverses the order of the elements in the vector x.
   (b) Let the matrix A have an LU factorization. Show that there is a related factorization $PAP = UL$, where U is upper triangular and L lower triangular.
3. In Algorithm 7.2.2 for Gaussian elimination the elements in A are assessed in row-wise order in the innermost loop over j. If implemented in Fortran this algorithm may be inefficient since this language stores two-dimensional arrays by columns. Modify Algorithm 7.2.2 so that the innermost loop instead involves a fixed column index and a varying row index.
4. What does M_j^{-1}, where M_j is defined in (7.2.17), look like?

5. Compute the inverse matrix A^{-1}, where

$$A = \begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 3 \\ 4 & 1 & 2 \end{pmatrix},$$

(a) By solving $AX = I$, using Gaussian elimination with partial pivoting.
(b) By LU factorization and using $A^{-1} = U^{-1}L^{-1}$.

7.3 Symmetric Matrices

7.3.1 Symmetric Positive Definite Matrices

Gaussian elimination can be adopted to several classes of matrices of special structure. As mentioned in Sec. sec7.2.5, one case when Gaussian elimination can be performed stably without any pivoting is when A is Hermitian or real symmetric and positive definite. Solving such systems is one of the most important problems in scientific computing.

Definition 7.3.1.
A matrix $A \in \mathbf{C}^{n\times n}$ is called **Hermitian** *if $A = A^H$, the conjugate transpose of A. If A is Hermitian, then the quadratic form $(x^HAx)^H = x^HAx$ is real and A is said to be* **positive definite** *if*

$$x^HAx > 0, \quad \forall x \in \mathbf{C}^n, \quad x \neq 0, \tag{7.3.1}$$

and **positive semidefinite** *if $x^TAx \geq 0$, for all $x \in \mathbf{R}^n$. Otherwise it is called* **indefinite**.

It is well known that all eigenvalues of an Hermitian matrix are real. An equivalent condition for an Hermitian matrix to be positive definite is that all its eigenvalues are positive

$$\lambda_k(A) > 0, \quad k = 1 : n.$$

Since this condition can be difficult to verify the following sufficient condition is useful. A Hermitian matrix A, which has positive diagonal elements and is diagonally dominant

$$a_{ii} > \sum_{j\neq i} |a_{ij}|, \quad i = 1 : n,$$

can be shown to be positive definite, since it follows from Gerschgorin's Theorem (Theorem 9.3.1) that the eigenvalues of A are all positive.

Clearly a positive definite matrix is nonsingular, since if it were singular there should be a null vector $x \neq 0$ such that $Ax = 0$ and then $x^HAx = 0$. Positive definite (semidefinite) matrices have the following important property:

Theorem 7.3.2. *Let $A \in \mathbf{C}^{n\times n}$ be positive definite and let $X \in \mathbf{C}^{n\times p}$ have full column rank. Then X^HAX is positive definite (semidefinite). In particular any*

principal $p \times p$ submatrix

$$\tilde{A} = \begin{pmatrix} a_{i_1 i_1} & \dots & a_{i_1 i_p} \\ \vdots & & \vdots \\ a_{i_p i_1} & \dots & a_{i_p i_p} \end{pmatrix} \in \mathbf{C}^{p\times p}, \quad 1 \le p < n,$$

is positive definite definite (semidefinite). In particular, taking $p = 1$, all diagonal elements in A are real positive (nonnegative).

Proof. Let $x \neq 0$ and let $y = Xx$. Then since X is of full column rank $y \neq 0$ and $x^H(X^HAX)x = y^HAy > 0$ by the positive definiteness of A. In particular any principal submatrix of A can be written as X^HAX, where the columns of X are taken as the columns $k = i_j$, $j = 1, \dots, p$ of the identity matrix. The case when A is positive semidefinite follows similarly. □

A Hermitian or symmetric matrix A of oder n has only $\frac{1}{2}n(n+1)$ independent elements. If A also is positive definite then symmetry can be preserved in Gaussian elimination and the number of operations and storage needed can be reduced by half. Indeed Gauss's derived his original algorithm for the symmetric positive definite systems coming from least squares problems (see Chapter 8). We consider below the special case when A is real and symmetric but all results are easily generalized to the complex Hermitian case.

Lemma 7.3.3. *Let A be a real symmetric matrix. Then if Gaussian elimination can be carried trough without pivoting the reduced matrices*

$$A = A^{(1)}, A^{(2)}, \dots, A^{(n)}$$

are all symmetric.

Proof. Assume that $A^{(k)}$ is symmetric, for some k, where $1 \le k < n$. Then by Algorithm 7.2.2 we have after the k-th elimination step

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}a_{kj}^{(k)} = a_{ji}^{(k)} - \frac{a_{jk}^{(k)}}{a_{kk}^{(k)}}a_{ki}^{(k)} = a_{ji}^{(k+1)},$$

$k+1 \le i, j \le n$. This shows that $A^{(k+1)}$ *is also a symmetric matrix*, and the result follows by induction. □

A more general result is the following. Partition the Hermitian positive definite matrix A as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^H & A_{22} \end{pmatrix}$$

where A_{11} is a square matrix, Then by Theorem 7.3.2 both A_{11} and A_{22} are Hermitian positive definite and therefore nonsingular. It follows that the Schur complement of A_{11} in A, which is

$$S = A_{22} - A_{12}^H A_{11}^{-1} A_{12}$$

exists and is Hermitian. Moreover, for $x \neq 0$, we have

$$x^H(A_{22} - A_{12}^H A_{11}^{-1} A_{12})x = \begin{pmatrix} y^H & -x^H \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^H & A_{22} \end{pmatrix} \begin{pmatrix} y \\ -x \end{pmatrix} > 0$$

where $y = A_{11}^{-1} A_{12} x$, it follows that S is positive definite.

From Lemma 7.3.3 it follows that in Gaussian elimination without pivoting only the elements in $A^{(k)}$, $k = 2 : n$, on and below the main diagonal have to be computed. Since any diagonal element can be brought in pivotal position by a symmetric row and column interchange, the same conclusion holds if pivots are chosen arbitrarily along the diagonal.

Assume that the lower triangular part of the symmetric matrix A is given. The following algorithm computes, *if it can be carried through*, a *unit lower triangular* matrix $L = (l_{ik})$, and a diagonal matrix $D = \operatorname{diag}(d_1, \ldots, d_n)$ such that

$$A = LDL^T. \tag{7.3.2}$$

Algorithm 7.3.1 Symmetric Gaussian Elimination.

for $k = 1 : n-1$
  $d_k := a_{kk}^{(k)}$;
  for $i = k+1 : n$
    $l_{ik} := a_{ik}^{(k)} / d_k$;
    for $j = k+1 : i$
      $a_{ij}^{(k+1)} := a_{ij}^{(k)} - l_{ik} d_k l_{jk}$;
    end
  end
end

In the last line we have substituted $d_k l_{jk}$ for $a_{jk}^{(k)}$.

Note that the elements in L and D can overwrite the elements in the lower triangular part of A, so also the storage requirement is halved to $n(n+1)/2$. The uniqueness of the LDL^T factorization follows trivially from the uniqueness of the LU factorization.

Using the factorization $A = LDL^T$ the linear system $Ax = b$ decomposes into the two triangular systems

$$Ly = b, \qquad L^T x = D^{-1} y. \tag{7.3.3}$$

The cost of solving these triangular systems is about n^2 flams.

Example 7.3.1.

It may not always be possible to perform Gaussian elimination on a symmetric matrix, using pivots chosen from the diagonal. Consider, for example, the

nonsingular symmetric matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & \epsilon \end{pmatrix}.$$

If we take $\epsilon = 0$, then both diagonal elements are zero, and symmetric Gaussian elimination breaks down. If $\epsilon \neq 0$, but $|\epsilon| \ll 1$, then choosing ϵ as pivot will not be stable. On the other hand, a row interchange will in general destroy symmetry!

We will prove that Gaussian elimination without pivoting can be carried out with positive pivot elements if and only if A is real and symmetric positive definite. (The same result applies to complex Hermitian matrices, but since the modifications necessary for this case are straightforward, we discuss here only the real case.) For symmetric semidefinite matrices symmetric pivoting can be used. The *indefinite* case requires more substantial modifications, which will be discussed in Section 7.3.4.

Theorem 7.3.4.

The symmetric matrix $A \in \mathbf{R}^{n\times n}$ is positive definite if and only if there exists a unit lower triangular matrix L and a diagonal matrix D with positive elements such that

$$A = LDL^T, \quad D = \operatorname{diag}(d_1, \ldots, d_n),$$

Proof. Assume first that we are given a symmetric matrix A, for which Algorithm 7.3.1 yields a factorization $A = LDL^T$ with positive pivotal elements $d_k > 0$, $k = 1 : n$. Then for all $x \neq 0$ we have $y = L^T x \neq 0$ and

$$x^T Ax = x^T LDL^T x = y^T Dy > 0.$$

It follows that A is positive definite.

The proof of the other part of the theorem is by induction on the order n of A. The result is trivial if $n = 1$, since then $D = d_1 = A = a_{11} > 0$ and $L = 1$. Now write

$$A = \begin{pmatrix} a_{11} & a^T \\ a & \tilde{A} \end{pmatrix} = L_1 D_1 L_1^T, \quad L_1 = \begin{pmatrix} 1 & 0 \\ d_1^{-1}a & I \end{pmatrix}, \quad D_1 = \begin{pmatrix} d_1 & 0 \\ 0 & B \end{pmatrix},$$

where $d_1 = a_{11}$, $B = \tilde{A} - d_1^{-1}aa^T$. Since A is positive definite it follows that D_1 is positive definite, and therefore $d_1 > 0$, and B is positive definite. Since B is of order $(n-1)$, by the induction hypothesis there exists a unique unit lower triangular matrix $\tilde{L}$ and diagonal matrix $\tilde{D}$ with positive elements such that $B = \tilde{L}\tilde{D}\tilde{L}^T$. Then it holds that $A = LDL^T$, where

$$L = \begin{pmatrix} 1 & 0 \\ d_1^{-1}a & \tilde{L} \end{pmatrix}, \qquad D = \begin{pmatrix} d_1 & 0 \\ 0 & \tilde{D} \end{pmatrix}.$$

□

Example 7.3.2. The Hilbert matrix $H_n \in \mathbf{R}^{n\times n}$ with elements

$$h_{ij} = 1/(i+j-1), \quad 1 \le i,j \le n,$$

is positive definite. Hence, if Gaussian elimination without pivoting is carried out then the pivotal elements are all positive. For example, for $n = 4$, symmetric Gaussian elimination yields the $H_4 = LDL^T$, where

$$D = \text{diag}\,(1, 1/12, 1/180, 1/2800), \qquad L = \begin{pmatrix} 1 & & & \\ 1/2 & 1 & & \\ 1/3 & 1 & 1 & \\ 1/4 & 9/10 & 3/2 & 1 \end{pmatrix}.$$

Theorem 7.3.4 also yields the following useful characterization of a positive definite matrix.

Theorem 7.3.5. Sylvester's Criterion
A symmetric matrix $A \in \mathbf{R}^{n\times n}$ is positive definite if and only if

$$\det(A_k) > 0, \quad k = 1, 2, \ldots, n,$$

where $A_k \in \mathbf{R}^{k\times k}$, $k = 1, 2 : n$, are the leading principal submatrices of A.

Proof. If symmetric Gaussian elimination is carried out without pivoting then

$$\det(A_k) = d_1 d_2 \cdots d_k.$$

Hence $\det(A_k) > 0$, $k = 1 : n$, if and only if all pivots are positive. However, by Theorem 7.3.2 this is the case if and only if A is positive definite. □

In prove a bound on the growth ratio for the symmetric positive definite we first show the following

Lemma 7.3.6. *For a symmetric positive definite matrix $A = (a_{ij}) \in \mathbf{R}^{n\times n}$ the maximum element of A lies on the diagonal.*

Proof. Theorem 7.3.2 and Sylvester's criterion imply that

$$0 < \det \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix} = a_{ii}a_{jj} - a_{ij}^2, \quad 1 \le i,j \le n.$$

Hence

$$|a_{ij}|^2 < a_{ii}a_{jj} \le \max_{1\le i\le n} a_{ii}^2,$$

from which the lemma follows. □

Theorem 7.3.7.

Let A be symmetric and positive definite. Then Gaussian elimination without pivoting is backward stable and the growth ratio satisfies $\rho_n \leq 1$.

Proof. In Algorithm 7.3.1 the diagonal elements are transformed in the k:th step of Gaussian elimination according to

$$a_{ii}^{(k+1)} = a_{ii}^{(k)} - (a_{ki}^{(k)})^2/a_{kk}^{(k)} = a_{ii}^{(k)}\Big(1 - (a_{ki}^{(k)})^2/\big(a_{ii}^{(k)}a_{kk}^{(k)}\big)\Big).$$

If A is positive definite so are $A^{(k)}$ and $A^{(k+1)}$. Using Lemma 7.3.6 it follows that $0 < a_{ii}^{(k+1)} \leq a_{ii}^{(k)}$, and hence the diagonal elements in the successive reduced matrices cannot increase. Thus we have

$$\max_{i,j,k} |a_{ij}^{(k)}| = \max_{i,k} a_{ii}^{(k)} \leq \max_{i} a_{ii} = \max_{i,j} |a_{ij}|,$$

which implies that $\rho_n \leq 1$. ∎

Any matrix $A \in \mathbf{R}^{n\times n}$ can be written as the sum of a symmetric and a skew-symmetric part, $A = H + S$, where

$$A_H = \tfrac{1}{2}(A + A^T), \qquad A_S = \tfrac{1}{2}(A - A^T). \tag{7.3.4}$$

A is symmetric if and only if $A_S = 0$. Sometimes A is called positive definite if its symmetric part A_H is positive definite. If the matrix A has a positive symmetric part then its leading principal submatrices are nonsingular and Gaussian elimination can be carried out to completion without pivoting. However, the resulting LU factorizing may not be stable as shown by the example

$$\begin{pmatrix} \epsilon & 1 \\ -1 & \epsilon \end{pmatrix} = \begin{pmatrix} 1 & \\ -1/\epsilon & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 1 \\ & \epsilon + 1/\epsilon \end{pmatrix}, \quad (\epsilon > 0).$$

These result can be extended to complex matrices with positive definite Hermitian part $A_H = \frac{1}{2}(A + A^H)$, for which its holds that $x^H Ax > 0$, for all nonzero $x \in \mathbf{C}^n$. Of particular interest are complex symmetric matrices, arising in computational electrodynamics, of the form

$$A = B + iC, \quad B, C \in \mathbf{R}^{n\times n}, \tag{7.3.5}$$

where $B = A_H$ and $C = A_S$ both are symmetric positive definite. It can be shown that for this class of matrices $\rho_n < 3$, so LU factorization without pivoting is stable (see [30]).

7.3.2 Cholesky Factorization

Let A be a symmetric positive definite matrix A. Then the $\mathrm{LDL}^{\mathrm{T}}$ factorization (7.3.2) exists and $D > 0$. Hence we can write

$$A = LDL^T = (LD^{1/2})(LD^{1/2})^T, \quad D^{1/2} = \operatorname{diag}(\sqrt{d}_1, \ldots, \sqrt{d}_n). \tag{7.3.6}$$

Defining the upper triangular matrix $R := D^{1/2}L^T$ we obtain the factorization

$$A = R^T R. \tag{7.3.7}$$

If we here take the diagonal elements of L to be positive it follows from the uniqueness of the $\mathrm{LDL}^{\mathrm{T}}$ factorization that this factorization is unique. The factorization (7.3.7) is called the **Cholesky factorization** of A. and R is called the **Cholesky factor** of A.[16]

The Cholesky factorization is obtained if in symmetric Gaussian elimination (Algorithm 7.3.1) we set $d_k = l_{kk} = (a_{kk}^{(k)})^{1/2}$. This gives the outer product version of Cholesky factorization in which in the kth step, the reduced matrix is modified by a rank-one matrix

$$A^{(k+1)} = A^{(k)} - l_k l_k^T,$$

where l_k denotes the column vector of multipliers.

In analogy to the compact schemes for LU factorization (see Section 7.2.6) it is possible to arrange the computations so that the elements in the Cholesky factor $R = (r_{ij})$ are determined directly. The matrix equation $A = R^T R$ with R upper triangular can be written

$$a_{ij} = \sum_{k=1}^{i} r_{ki} r_{kj} = \sum_{k=1}^{i-1} r_{ki} r_{kj} + r_{ii} r_{ij}, \quad 1 \le i \le j \le n. \tag{7.3.8}$$

This is $n(n+1)/2$ equations for the unknown elements in R. We remark that for $i = j$ this gives

$$\max_i r_{ij}^2 \le \sum_{k=1}^{j} r_{kj}^2 = a_j \le \max_i a_{ii},$$

which shows that the elements in R are bounded maximum diagonal element in A. Solving for r_{ij} from the corresponding equation in (7.3.8), we obtain

$$r_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\Big)/r_{ii}, \quad i < j, \qquad r_{jj} = \Big(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2\Big)^{1/2}.$$

If properly sequenced, these equations can be used in a recursive fashion to compute the elements in R. For example the elements in R can be determined one row or one column at a time.

Algorithm 7.3.2 Cholesky Algorithm; column-wise order

for $j = 1 : n$
  for $i = 1 : j - 1$

[16] André-Louis Cholesky (1875–1918) was a French military officer involved in geodesy and surveying in Crete and North Africa just before World War I. He developed the algorithm named after him and his work was posthumously published by a fellow officer, Benoit in 1924.

$$r_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\Big)/r_{ii};$$

end

$$r_{jj} = \Big(a_{jj} - \sum_{k=1}^{j-1} r_{kj}^2\Big)^{1/2};$$

end

The column-wise ordering has the advantage of giving the Cholesky factors of all leading principal submatrices of A. An algorithm which computes the elements of R in row-wise order is obtained by reversing the two loops in the code above.

Algorithm 7.3.3 Cholesky Algorithm; row-wise order.

for $i = 1 : n$

$$r_{ii} = \Big(a_{ii} - \sum_{k=1}^{i-1} r_{ki}^2\Big)^{1/2};$$

for $j = i + 1 : n$

$$r_{ij} = \Big(a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj}\Big)/r_{ii};$$

end

end

These two versions of the Cholesky algorithm are not only mathematically equivalent but also *numerically equivalent*, i.e., they will compute the same Cholesky factor, taking rounding errors into account. In the Cholesky factorization only

$$\begin{pmatrix} 1 & 2 & 4 & 7 & 11 \\ & 3 & 5 & 8 & 12 \\ & & 6 & 9 & 13 \\ & & & 10 & 14 \\ & & & & 15 \end{pmatrix}$$

Figure 7.3.1. *The mapping of array-subscript of an upper triangular matrix of order 5.*

elements in the upper triangular part of A are referenced and only these elements need to be stored. Since most programming languages only support rectangular arrays this means that the lower triangular part of the array holding A is not used. One possibility is then to use the lower half of the array to store R^T and not overwrite the original data. Another option is to store the elements of the upper triangular part of A column-wise in a vector, see Fig. 7.3.1. which is known as **packed storage**. This data is then and overwritten by the elements of R during the computations. Using packed storage complicates somewhat index computations but is useful when economizing storage is worthwhile.

Some applications lead to a linear systems where $A \in \mathbf{R}^{n\times n}$ is a symmetric positive *semidefinite* matrix ($x^T Ax \geq 0\ \forall x \neq 0$) with $\text{rank}(A) = r < n$. One example is rank deficient least squares problems; see Section 8.5. Another example is when the finite element method is applied to a problem where rigid body motion occurs, which implies $r \leq n-1$. In the semidefinite case a Cholesky factorization still exists, but symmetric pivoting needs to be incorporated. In the kth elimination step *a maximal diagonal element* $a_{ss}^{(k)}$ in the reduced matrix $A^{(k)}$ is chosen as pivot, i.e.,

$$a_{ss}^{(k)} = \max_{k\leq i\leq n} a_{ii}^{(k)}. \tag{7.3.9}$$

This pivoting strategy is easily implemented in Algorithm 7.3.1, the outer product version. Symmetric pivoting is also beneficial when A is close to a rank deficient matrix.

Since all reduced matrices are positive semidefinite their largest element lies on the diagonal. Hence diagonal pivoting is equivalent to complete pivoting in Gaussian elimination. In exact computation the Cholesky algorithm stops when all diagonal elements in the reduced matrix are zero. This implies that the reduced matrix is the zero matrix.

If A has rank $r < n$ the resulting Cholesky factorization has the upper trapezoidal form

$$P^T AP = R^T R, \qquad R = \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \tag{7.3.10}$$

where P is a permutation matrix and $R_{11} \in \mathbf{R}^{r\times r}$ with positive diagonal elements. The linear system $Ax = b$, or $P^T AP(P^T x) = P^T b$, then becomes

$$R^T R\tilde{x} = \tilde{b}, \quad \tilde{x} = P^T x, \quad \tilde{b} = P^T b.$$

Setting $z = R\tilde{x}$ the linear system reads

$$R^T z = \begin{pmatrix} R_{11}^T \\ R_{12}^T \end{pmatrix} z = \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix},$$

and from the first r equations we obtain $z = R_{11}^{-T}\tilde{b}_1$. Substituting this in the last $n-r$ equations we get

$$0 = R_{12}^T z - \tilde{b}_2 = \begin{pmatrix} R_{12}^T R_{11}^{-T} & -I \end{pmatrix} \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix}.$$

These equations are equivalent to $b \perp \mathcal{N}(A)$ and express the condition for the linear system $Ax = b$ to be consistent. If they are not satisfied a solution does not exist. It remains to solve $L^T\tilde{x} = z$, which gives

$$R_{11}\tilde{x}_1 = z - R_{12}\tilde{x}_2.$$

For an arbitrarily chosen $\tilde{x}_2$ we can uniquely determine $\tilde{x}_1$ so that these equations are satisfied. This expresses the fact that a consistent singular system has an infinite number of solutions. Finally the permutations are undone to obtain $x = P\tilde{x}$.

Rounding errors can cause negative elements to appear on the diagonal in the Cholesky algorithm even when A is positive semidefinite. Similarly, because of rounding errors the reduced matrix will in general be nonzero after r steps even when $\operatorname{rank}(A) = r$. The question arises when to terminate the Cholesky factorization of a semidefinite matrix. One possibility is to continue until

$$\max_{k \le i \le n} a_{ii}^{(k)} \le 0,$$

but this may cause unnecessary work in eliminating negligible elements. Further discussion of this aspect is postponed until Chapter 8.

7.3.3 Inertia of Symmetric Matrices

Let $A \in \mathbf{C}^{n\times n}$ be an Hermitian matrix. The **inertia** of A is defined as the number triple $\operatorname{in}(A) = (\pi, \nu, \delta)$ of positive, negative, and zero eigenvalues of A. If A is positive definite matrix and $Ax = \lambda x$, we have

$$x^H A x = \lambda x^H x > 0.$$

Hence all eigenvalues must be positive and the inertia is $(n, 0, 0)$.

Hermitian matrices arise naturally in the study of quadratic forms $\psi(x) = x^H A x$. By the coordinate transformation $x = Ty$ this quadratic form is transformed into

$$\psi(Ty) = y^H \hat{A} y, \quad \hat{A} = T^H A T.$$

The mapping of A onto $T^H A T$ is called a **congruence transformation** of A, and we say that A and $\hat{A}$ are **congruent**. (Notice that a congruence transformation with a nonsingular matrix means a transformation to a coordinate system which is usually not rectangular.) Unless T is unitary these transformations do not, in general, preserve eigenvalues. However, Sylvester's famous law of inertia says that the *signs of eigenvalues are preserved by congruence transformations.*

Theorem 7.3.8. Sylvester's Law of Inertia *If $A \in \mathbf{C}^{n\times n}$ is symmetric and $T \in \mathbf{C}^{n\times n}$ is nonsingular then A and $\hat{A} = T^H A T$ have the same inertia.*

Proof. Since A and $\hat{A}$ are Hermitian there exist unitary matrices U and $\hat{U}$ such that

$$U^H A U = D, \qquad \hat{U}^H \hat{A} \hat{U} = \hat{D},$$

where $D = \operatorname{diag}(\lambda_i)$ and $\hat{D} = \operatorname{diag}(\hat{\lambda}_i)$ are diagonal matrices of eigenvalues. By definition we have $\operatorname{in}(A) = \operatorname{in}(D)$, $\operatorname{in}(\hat{A}) = \operatorname{in}(\hat{D})$, and hence, we want to prove that $\operatorname{in}(D) = \operatorname{in}(\hat{D})$, where

$$\hat{D} = S^H D S, \quad S = U^H T \hat{U}.$$

Assume that $\pi \neq \hat{\pi}$, say $\pi > \hat{\pi}$, and that the eigenvalues are ordered so that $\lambda_j > 0$ for $j \le \pi$ and $\hat{\lambda}_j > 0$ for $j \le \hat{\pi}$. Let $x = S\hat{x}$ and consider the quadratic form $\psi(x) = x^H D x = \hat{x}^H \hat{D} \hat{x}$, or

$$\psi(x) = \sum_{j=1}^{n} \lambda_j |\xi_j|^2 = \sum_{j=1}^{n} \hat{\lambda}_j |\hat{\xi}_j|^2.$$

Let $x^* \neq 0$ be a solution to the $n - \pi + \hat{\pi} < n$ homogeneous linear relations

$$\xi_j = 0, \quad j > \pi, \qquad \hat{\xi}_j = (S^{-1}x)_j = 0, \quad j \leq \hat{\pi}.$$

Then

$$\psi(x^*) = \sum_{j=1}^{\pi} \lambda_j |\xi_j^*|^2 > 0, \quad \psi(x^*) = \sum_{j=\hat{\pi}}^{n} \hat{\lambda}_j |\hat{\xi}_j^*|^2 \leq 0.$$

This is a contradiction and hence the assumption that $\pi \neq \hat{\pi}$ is false, so A and $\hat{A}$ have the same number of positive eigenvalues. Using the same argument on $-A$ it follows that also $\nu = \hat{\nu}$, and since the number of eigenvalues is the same $\delta = \hat{\delta}$. □

Let $A \in \mathbf{R}^{n\times n}$ be a real symmetric matrix and consider the quadratic equation

$$x^T Ax - 2bx = c, \quad A \neq 0. \tag{7.3.11}$$

The solution sets of this equation are sometimes called **conical sections**. If $b = 0$, then the surface has its center at the origin reads $x^T Ax = c$. The inertia of A completely determines the geometric type of the conical section.

Sylvester's theorem tells that the geometric type of the surface can be determined without computing the eigenvalues? Since we can always multiply the equation by -1 we can assume that there are at least one positive eigenvalues. Then, for $n = 2$ there are three possibilities:

$$(2, 0, 0) \text{ ellipse}; \quad (1, 0, 1) \text{ parabola}; \quad (1, 1, 0) \text{ hyperbola}.$$

In n dimensions there will be $n(n+1)/2$ cases, assuming that at least one eigenvalue is positive.

7.3.4 Symmetric Indefinite Matrices

As shown by Example 7.3.1, the LDL^T factorization of a symmetric indefinite matrix, although efficient computationally, may not exist and can be unstable. This is true even when symmetric row and column interchanges are used, to select at each stage the largest diagonal element in the reduced matrix as pivot. One stable way of factorizing an indefinite matrix is of course to compute an unsymmetric LU factorization using Gaussian elimination with partial pivoting. However, this factorization does not give the inertia of A and we give up the savings of a factor one half in d storage.

The following example shows that in order to enable a stable $\mathrm{LDL}^{\mathrm{T}}$ factorization for a symmetric *indefinite* matrix A, it is necessary to consider a block factorization where D is block diagonal with also 2×2 diagonal blocks..

Example 7.3.3.

The symmetric matrix

$$A = \begin{pmatrix} \epsilon & 1 \\ 1 & \epsilon \end{pmatrix}, \quad 0 < \epsilon \ll 1,$$

is indefinite since $\det(A) = \lambda_1\lambda_2 = \epsilon^1 - 1 < 0$. If we compute the $\mathrm{LDL}^{\mathrm{T}}$ factorization of A without pivoting we obtain

$$A = \begin{pmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{pmatrix} \begin{pmatrix} \epsilon & 0 \\ 0 & \epsilon - \epsilon^{-1} \end{pmatrix} \begin{pmatrix} 1 & \epsilon^{-1} \\ 0 & 1 \end{pmatrix}.$$

which shows that there is unbounded element growth. However, A is well conditioned with inverse

$$A^{-1} = \frac{1}{\epsilon^2 - 1} \begin{pmatrix} \epsilon & 1 \\ 1 & \epsilon \end{pmatrix}, \quad 0 < \epsilon \ll 1.$$

It is quite straightforward to generalize Gaussian elimination to use any nonsingular 2×2 principal submatrix as pivot. By a symmetric permutation this submatrix is brought to the upper left corner, and the permuted matrix partitioned as

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix}, \quad A_{11} = \begin{pmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{pmatrix}.$$

Then the Schur complement of A_{11}, $S = A_{22} - A_{12}^T A_{11}^{-1} A_{12}$, exists where

$$A_{11}^{-1} = \frac{1}{\delta_{12}} \begin{pmatrix} a_{22} & -a_{21} \\ -a_{21} & a_{11} \end{pmatrix}, \quad \delta_{12} = \det(A_{11}) = a_{11}a_{22} - a_{21}^2. \tag{7.3.12}$$

We obtain the symmetric block factorization

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ L & I \end{pmatrix} \begin{pmatrix} A_{11} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & L^T \\ 0 & I \end{pmatrix}, \tag{7.3.13}$$

where $L = A_{12}^T A_{11}^{-1}$. This determines the first two columns of a unit lower triangular matrix $L = L_{21} = A_{21}A_{11}^{-1}$, in an $\mathrm{LDL}^{\mathrm{T}}$ factorization of A. The block A_{22} is transformed into the symmetric matrix $A_{22}^{(3)} = A_{22} - L_{21}A_{21}^T$ with components

$$a_{ij}^{(3)} = a_{ij} - l_{i1}a_{1j} - l_{i2}a_{2j}, \quad 2 \le j \le i \le n. \tag{7.3.14}$$

It can be shown that $A_{22}^{(3)}$ is the same reduced matrix as if two steps of Gaussian elimination were taken, first pivoting on the element a_{12} and then on a_{21}.

A similar reduction is used if 2×2 pivots are taken at a later stage in the factorization. Ultimately a factorization $A = LDL^T$ is computed in which D is block diagonal with in general a mixture of 1×1 and 2×2 blocks, and L is unit lower triangular with $l_{k+1,k} = 0$ when $A^{(k)}$ is reduced by a 2×2 pivot. Since the effect of taking a 2×2 step is to reduce A by the equivalent of *two* 1×1 pivot steps, the amount of work must be balanced against that. The part of the calculation which dominates the operation count is (7.3.14), and this is twice the work as for an 1×1 pivot. Therefore the leading term in the operations count is always $n^3/6$, whichever type of pivots is used.

The main issue then is to find a pivotal strategy that will give control of element growth without requiring too much search. One possible strategy is comparable to that of complete pivoting. Consider the first stage of the factorization and set

$$\mu_0 = \max_{ij} |a_{ij}| = |a_{pq}|, \qquad \mu_1 = \max_{i} |a_{ii}| = |a_{rr}|.$$

Then if

$$\mu_1/\mu_0 > \alpha = (\sqrt{17}+1)/8 \approx 0.6404,$$

the diagonal element a_{rr} is taken as an 1×1 pivot. Otherwise the 2×2 pivot.

$$\begin{pmatrix} a_{pp} & a_{qp} \\ a_{qp} & a_{qq} \end{pmatrix}, \quad p < q,$$

is chosen. In other words if there is a diagonal element not much smaller than the element of maximum magnitude this is taken as an 1×1 pivot. The magical number α has been chosen so as to minimize the bound on the growth per stage of elements of A, allowing for the fact that a 2×2 pivot is equivalent to two stages. The derivation, which is straight forward but tedious (see Higham [41, Sec. 11.1.1]) is omitted here.

With this choice the element growth can be shown to be bounded by

$$\rho_n \le (1 + 1/\rho)^{n-1} < (2.57)^{n-1}. \tag{7.3.15}$$

This exponential growth may seem alarming, but the important fact is that the reduced matrices cannot grow abruptly from step to step. No example is known where significant element growth occur at every step. The bound in (7.3.15) can be compared to the bound 2^{n-1}, which holds for Gaussian elimination with partial pivoting. The elements in L can be bounded by $1/(1-\alpha) < 2.781$ and this pivoting strategy therefore gives a backward stable factorization.

Since the complete pivoting strategy above requires the whole active submatrix to be searched in each stage, it requires $O(n^3)$ comparisons. The same bound for element growth (7.3.15) can be achieved using the following partial pivoting strategy due to **Bunch–Kaufman** [11]. For simplicity of notations we restrict our attention to the first stage of the elimination. All later stages proceed similarly. First determine the off-diagonal element of largest magnitude in the first column,

$$\lambda = |a_{r1}| = \max_{i \neq 1} |a_{i1}|.$$

If $|a_{11}| \ge \rho\lambda$, then take a_{11} as pivot. Else, determine the largest off-diagonal element in column r,

$$\sigma = \max_{1 \le i \le n} |a_{ir}|, \quad i \neq r.$$

If $|a_{11}| \ge \rho\lambda^2/\sigma$, then again take a_{11} as pivot, else if $|a_{rr}| \ge \rho\sigma$, take a_{rr} as pivot. Otherwise take as pivot the 2×2 principal submatrix

$$\begin{pmatrix} a_{11} & a_{1r} \\ a_{1r} & a_{rr} \end{pmatrix}.$$

Note that at most 2 columns need to be searched in each step, and at most $O(n^2)$ comparisons are needed in all.

Normwise backward stability can be shown to hold also for the Bunch–Kaufman partial pivoting strategy. However, it is no longer true that the elements of L are bounded independently of A. The following example (Higham [41, Sec. 11.1.2]) shows that for partial pivoting L is unbounded:

$$A = \begin{pmatrix} 0 & \epsilon & 0 \\ \epsilon & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 0 & 1 & \\ \epsilon^{-1} & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & \epsilon & \\ \epsilon & 0 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \epsilon^{-1} \\ & 1 & 0 \\ & & 1 \end{pmatrix}. \tag{7.3.16}$$

Note that whenever a 2×2 pivot is used, we have

$$a_{11}a_{rr} \le \rho^2 |a_{1r}|^2 < |a_{1r}|^2.$$

Hence with both pivoting strategies any 2×2 block in the block diagonal matrix D has a negative determinant $\delta_{1r} = a_{11}a_{rr} - a_{1r}^2 < 0$ and by Sylvester's Theorem corresponds to one positive and one negative eigenvalue. Hence a 2×2 pivot cannot occur if A is positive definite and in this case all pivots chosen by the Bunch–Kaufman strategy will be 1×1.

For solving a linear system $Ax = b$ the $\mathrm{LDL}^{\mathrm{T}}$ factorization produced by the Bunch–Kaufman pivoting strategy is satisfactory. For certain other applications the possibility of a large L factor is not acceptable. A bounded L factor can be achieved with the modified pivoting strategy suggested in [4]. This symmetric pivoting is roughly similar to rook pivoting and has a total cost of between $O(n^2)$ and $O(n^3)$ comparisons. Probabilistic results suggest that on the average the cost is only $O(n^2)$. In this strategy a search is performed until two indices r and s have been found such that the element a_{rs} bounds in modulus the other off-diagonal elements in the r and s columns (rows). Then either the 2×2 pivot D_{rs} or the largest in modulus of the two diagonal elements as an 1×1 pivot is taken, according to the test

$$\max(|a_{rr}|, |a_{ss}|) \ge \alpha |a_{rs}|.$$

Aasen [1] has given an algorithm that for a symmetric matrix $A \in \mathbf{R}^{n \times n}$ computes the factorization

$$PAP^T = LTL^T, \tag{7.3.17}$$

where L is unit lower triangular and T symmetric tridiagonal.

None of the algorithms described here preserves the band structure of the matrix A. In this case Gaussian elimination with partial pivoting can be used but as remarked before this will destroy symmetry and does not reveal the inertia. For the special case of a *tridiagonal* symmetric indefinite matrices an algorithm for computing an $\mathrm{LDL}^{\mathrm{T}}$ factorization will be given in Sec. 7.4.3.

A block $\mathrm{LDL}^{\mathrm{T}}$ factorization can also be computed for a real skew-symmetric matrix A. Note that $A^T = -A$ implies that such a matrix has zero diagonal elements. Further, since

$$(x^T Ax)^T = x^T A^T x = -x^T Ax,$$

it follows that all nonzero eigenvalues come in pure imaginary complex conjugate pairs. In the first step of the factorization if the first column is zero there is nothing to do. Otherwise we look for an off-diagonal element $a_{p,q}$, $p > q$ such that

$$|a_{p,q}| = \max\{\max_{1<i\le n} |a_{i,1}|, \max_{1<i\le n} |a_{i,2}|\},$$

and take the 2×2 pivot

$$\begin{pmatrix} 0 & -a_{p,q} \\ a_{p,q} & 0 \end{pmatrix}.$$

It can be shown that this pivoting the growth ratio is bounded by $\rho_n \le (\sqrt{3})^{n-2}$, which is smaller than for Gaussian elimination with partial pivoting for a general matrix.

Review Questions

1. (a) Give two necessary and sufficient conditions for a real symmetric matrix A to be positive definite.

(b) Show that if A is symmetric positive definite so is its inverse A^{-1}.

2. What simplifications occur in Gaussian elimination applied to a symmetric, positive definite matrix?

3. What is the relation of Cholesky factorization to Gaussian elimination? Give an example of a symmetric matrix A for which the Cholesky decomposition does not exist.

4. Show that if A is skew-symmetric, then iA is Hermitian.

5. Show that the Cholesky factorization is unique for positive definite matrices provided R is normalized to have positive diagonal entries.

6. (a) Formulate and prove Sylvester's law of inertia.

(b) Show that for $n = 3$ there are six different geometric types of conical sections $x^T Ax - 2b^T x = c$, provided that $A \neq 0$ and is normalized to have at least one positive eigenvalue.

Problems

1. If A is a symmetric positive definite matrix how should you compute $x^T Ax$ for a given vector x?

2. Show that if A is symmetric and positive definite then $|a_{ij}| \le (a_{ii} + a_{jj})/2$.

3. Show by computing the Cholesky factorization $A = LL^T$ that the matrix

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

is positive definite.

4. The Hilbert matrix $H_n \in \mathbf{R}^{n\times n}$ with elements

$$a_{ij} = 1/(i+j-1), \quad 1 \le i,j \le n,$$

is symmetric positive definite for all n. Denote by $\bar{H}_4$ the corresponding matrix with elements rounded to five decimal places, and compute its Cholesky factor $\bar{L}$. Then compute the difference $(\bar{L}\bar{L}^T - \bar{A})$ and compare it with $(A - \bar{A})$.

5. Let $A + iB$ be Hermitian and positive definite, where $A, B \in \mathbf{R}^{n\times n}$. Show that the real matrix

$$C = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

is symmetric and positive definite. How can a linear system $(A+iB)(x+iy) = b+ic$ be solved using a Cholesky factorization of C?

6. Implement the Cholesky factorization using packed storage for A and R.

7.4 Banded Linear Systems

7.4.1 Banded Matrices

Linear systems $Ax = b$ where the matrix A is banded arise in problems where each variable x_i is coupled by an equation only to a few other variables x_j such that $|j-i|$ is small. We make the following definition (note that it applies also to matrices which are not square):

We recall from Definition 7.1.1 that a matrix A is said to have upper bandwidth r and lower bandwidth s if

$$a_{ij} = 0, \quad j > i + r, \qquad a_{ij} = 0, \quad i > j + s,$$

respectively. This means that the number of non-zero diagonals above and below the main diagonal are r and s respectively. The maximum number of nonzero elements in any row is then $w = r + s + 1$, which is the **bandwidth** of A.

For a matrix $A \in \mathbf{R}^{m\times n}$ which is not square we define the bandwidth as

$$w = \max_{1\le i\le m} \{j - k + 1 \mid a_{ij}a_{ik} \neq 0\}.$$

Note that the bandwidth of a matrix depends on the ordering of its rows and columns. An important, but hard, problem is to find optimal orderings that minimize the bandwidth. However, there are good heuristic algorithms that can be used in practice and give almost optimal results; see Section 7.6.3.

It is convenient to introduce some additional notations for manipulating band matrices.[17]

[17] These notations are taken from MATLAB .

Definition 7.4.1.

If $a = (a_1, a_2, \ldots, a_{n-r})^T$ is a column vector with $n - r$ components then $A = \operatorname{diag}(a, k)$, $|k| < n$, denotes a square matrix of order n with the elements of a on its kth diagonal; $k = 0$ is the main diagonal; $k > 0$ is above the main diagonal; $k < 0$ is below the main diagonal.

If A is a square matrix of order n, then $\operatorname{diag}(A, k) \in \mathbf{R}^{(n-k)}$, $|k| < n$, is the column vector formed from the elements of the kth diagonal of A.

Assume that A and B are banded matrices of order n, which both have a small bandwidth compared to n. Then, since there are few nonzero elements in the rows and columns of A and B the usual algorithms for forming the product AB are not effective on vector computers. We now give an algorithm for multiplying matrices by diagonals, which overcomes this drawback.

Lemma 7.4.2.

Let $A = \operatorname{diag}(a, r)$ and $B = \operatorname{diag}(b, s)$ and set $C = AB$. If $|r + s| \geq n$ then $C = 0$; otherwise $C = \operatorname{diag}(c, r+s)$, where the elements of the vector $c \in \mathbf{R}^{(n-|r+s|)}$ are obtained by pointwise multiplication of shifted vectors a and b:

$$c = \begin{cases} (a_1 b_{r+1}, \ldots, a_{n-r-s} b_{n-s})^T, & \text{if } r, s \geq 0, \\ (a_{|s|+1} b_1, \ldots, a_{n-|r|} b_{n-|r+s|})^T, & \text{if } r, s \leq 0. \\ (0, \ldots, 0, a_1 b_1, \ldots, a_{n-s} b_{n-s})^T, & \text{if } r < 0, \quad s > 0, \quad r + s \geq 0. \\ (0, \ldots, 0, a_1 b_1, \ldots, a_{n-|r|} b_{n-|r|})^T, & \text{if } r < 0, \quad s > 0, \quad r + s < 0. \\ (a_1 b_{|r+s|+1}, \ldots, a_{n-r} b_{n-|s|}, 0, \ldots, 0)^T, & \text{if } r > 0, \quad s < 0, \quad r + s \geq 0. \\ (a_{r+1} b_1, \ldots, a_{n-r} b_{n-|s|}, 0, \ldots, 0)^T, & \text{if } r > 0, \quad s < 0, \quad r + s < 0. \end{cases} \tag{7.4.1}$$

Note that when $rs < 0$, zeros are added at the beginning or end to get a vector c of length $n - |r + s|$.

The number of cases in this lemma looks a bit forbidding, so to clarify the situation a bit more we consider a specific case.

Example 7.4.1.

Let A and B be tridiagonal matrices of size 5×5

$$A = \begin{pmatrix} a_1 & c_1 & & & \\ b_1 & a_2 & c_2 & & \\ & b_2 & a_3 & c_3 & \\ & & b_3 & a_4 & c_4 \\ & & & b_4 & a_5 \end{pmatrix}, \qquad B = \begin{pmatrix} d_1 & f_1 & & & \\ e_1 & d_2 & f_2 & & \\ & e_2 & d_3 & f_3 & \\ & & e_3 & d_4 & f_4 \\ & & & e_4 & d_5 \end{pmatrix}.$$

Then $C = AB$ will be a banded matrix with upper and lower bandwidth equal to two. The five diagonals of C are

$$\begin{aligned} \operatorname{diag}(C, 0) = {} & (a_1 d_1, a_2 d_2, a_3 d_3, a_4 d_4, a_5 d_5) \\ & + (0, b_1 f_1, b_2 f_2, b_3 f_3, b_4 f_4) \\ & + (c_1 e_1, c_2 e_2, c_3 e_3, c_4 e_4, 0), \end{aligned}$$

$$\begin{aligned}\operatorname{diag}(C,1) &= (a_1f_1, a_2f_2, a_3f_3, a_4f_4)\\ &\quad + (c_1d_2, c_2d_3, c_3d_4, c_4d_5),\\ \operatorname{diag}(C,-1) &= (b_1d_1, b_2d_2, b_3d_3, b_4d_4)\\ &\quad + (a_2e_1, a_3e_2, a_4e_3, a_5e_4),\\ \operatorname{diag}(C,2) &= (c_1f_2, c_2f_3, c_3f_4),\\ \operatorname{diag}(C,-2) &= (b_2e_1, b_3e_2, b_4e_3).\end{aligned}$$

The number of operations are exactly the same as in the conventional schemes, but only $3^2 = 9$ pointwise vector multiplications are required.

Lemma 7.4.3.
Let $A, B \in \mathbf{R}^{n\times n}$ have lower (upper) bandwidth r and s respectively. Then the product AB has lower (upper) bandwidth $r + s$.

7.4.2 LU Factorization of Banded Matrices

A matrix A for which all nonzero elements are located in consecutive diagonals is called a **band matrix**.

Many applications give rise to linear systems $Ax = b$, where the nonzero elements in the matrix A are located in a band centered along the principal diagonal. Such matrices are called **band matrices** and are the simplest examples of **sparse** matrices, i.e., matrices where only a small proportion of the n^2 elements are nonzero. Such matrices arise frequently in, for example, the numerical solution of boundary value problems for ordinary and partial differential equations.

Several classes of band matrices that occur frequently have special names. Thus, a matrix for which $r = s = 1$ is called **tridiagonal**, if $r = 0$, $s = 1$ it is called (lower) **bidiagonal** etc. For example, the matrix

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ a_{31} & a_{32} & a_{33} & a_{34} & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & \\ & & a_{53} & a_{54} & a_{55} & a_{56} \\ & & & a_{64} & a_{65} & a_{66} \end{pmatrix}$$

has $r = 1$, $s = 2$ and $w = 4$.

Band matrices are well-suited for Gaussian elimination, since if no pivoting is required *the band structure is preserved*. Recall that pivoting is not needed for stability, e.g., when A is diagonally dominant.

Theorem 7.4.4. *Let A be a band matrix with upper bandwidth r and lower band width s. If A has an LU-decomposition $A = LU$, then U has upper bandwidth r and L lower bandwidth s.*

Proof. The factors L and U are unique and can be computed, for example, by Doolittle's method (7.2.15). Assume that the first $k-1$ rows of U and columns of L have bandwidth r and s, that is, for $p = 1 : k-1$

$$l_{ip} = 0, \quad i > p+s, \qquad u_{pj} = 0, \quad j > p+r. \tag{7.4.2}$$

The proof is by induction in k. The assumption is trivially true for $k = 1$. Since $a_{kj} = 0$, $j > k+r$ we have from (7.2.9) and (7.4.2)

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} l_{kp}u_{pj} = 0 - 0 = 0, \quad j > k+r.$$

Similarly it follows that $l_{ik} = 0$, $i > k+s$, which completes the induction step. □

A band matrix $A \in \mathbf{R}^{n\times n}$ may be stored by diagonals in an array of dimension $n \times (r+s+1)$ or $(r+s+1) \times n$. For example, the matrix above can be stored as

$$\begin{matrix} * & * & a_{11} & a_{12} \\ * & a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} & a_{45} \\ a_{53} & a_{54} & a_{55} & a_{56} \\ a_{64} & a_{65} & a_{66} & * \end{matrix}, \quad \text{or} \quad \begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} & a_{56} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} & a_{66} \\ a_{21} & a_{32} & a_{43} & a_{54} & a_{65} & * \\ a_{31} & a_{42} & a_{53} & a_{64} & * & * \end{matrix}.$$

Notice that except for a few elements indicated by asterisks in the initial and final rows, only nonzero elements of A are stored. For example, passing along a row in the second storage scheme above moves along a diagonal of the matrix, and the columns are aligned.

For a general band matrix Algorithm 7.2.2, Gaussian elimination without pivoting, should be modified as follows to operate only on nonzero elements: The algorithms given below are written as if the matrix was conventionally stored. It is a useful exercise to rewrite them for the case when A, L, and U are stored by diagonals!

Algorithm 7.4.1 Banded Gaussian Elimination.
Let $A = \in \mathbf{R}^{n\times n}$ be a given matrix with upper bandwidth r and lower bandwidth s. The following algorithm computes the LU factorization of A, *provided it exists.* The element a_{ij} is overwritten by l_{ij} if $i > j$ and by u_{ij} otherwise.

```
for k = 1 : n − 1
    for i = k + 1 : min(k + s, n)
        l_ik := a_ik^(k) / a_kk^(k);
        for j = k + 1 : min(k + r, n)
            a_ij^(k+1) := a_ij^(k) − l_ik a_kj^(k);
        end
    end
end
```

An operation count shows that this algorithm requires t flams, where

$$t = \begin{cases} nr(s+1) - \frac{1}{2}rs^2 - \frac{1}{6}r^3, & \text{if } r \le s; \\ ns(s+1) - \frac{2}{3}s^3, & \text{if } r = s; \\ ns(r+1) - \frac{1}{2}sr^2 - \frac{1}{6}s^3, & \text{if } r > s. \end{cases}$$

Whenever $r \ll n$ *or* $s \ll n$ this is much less than the $n^3/3$ flams required in the full case.

Analogous savings can be made in forward- and back-substitution. Let L and U be the triangular factors computed by Algorithm 7.4.2. The solution of the two banded triangular systems $Ly = b$ and $Ux = y$ are obtained from

$$y_i = b_i - \sum_{max(1,i-s)}^{i-1} l_{ij}y_j, \quad i = 1:n$$

$$x_i := \Big(y_i - \sum_{j=i+1}^{\min(i+r,n)} u_{ij}x_j\Big)/u_{ii}, \quad i = n:(-1):1.$$

These algorithms require $ns - \frac{1}{2}s^2$ and $(n - \frac{r}{2})(r+1)$ flops, respectively. They are easily modified so that y and x overwrites b in storage.

Unless A is diagonally dominant or symmetric positive definite, partial pivoting should be used. The pivoting will cause the introduction of elements outside the band. This is illustrated below for the case when $s = 2$ and $r = 1$. The first step of the elimination is shown, where it is assumed that a_{31} is chosen as pivot and therefore rows 1 and 3 interchanged:

$$\begin{matrix} a_{31} & a_{32} & a_{33} & a_{34} & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ a_{11} & a_{12} & & & & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & & \\ & & a_{53} & a_{54} & a_{55} & a_{56} & \\ & & & \cdots & & & \end{matrix} \quad \Longrightarrow \quad \begin{matrix} u_{11} & u_{12} & u_{13} & u_{14} & & \\ l_{21} & a_{22}^{(2)} & a_{23}^{(2)} & \mathbf{a}_{24}^{(2)} & & \\ l_{31} & a_{32}^{(2)} & \mathbf{a}_{33}^{(2)} & \mathbf{a}_{34}^{(2)} & & \\ & a_{42} & a_{43} & a_{44} & a_{45} & \\ & & a_{53} & a_{54} & a_{55} & a_{56} \\ & & & \cdots & & \end{matrix} .$$

where fill-in elements are shown in boldface. Hence *the upper bandwidth of U may increase to $r+s$. The matrix L will still have only s elements below the main diagonal in all columns but no useful band structure.* This can be seen from the example above where, e.g., the elements l_{21} and l_{31} may be subject to later permutations, destroying the band-structure of the first column.

Example 7.4.2.

A class of matrices with unsymmetric band structure is upper (lower) **Hessenberg matrices**[18] for which $s = 1$ ($r = 1$). These are of particular interest

[18] Karl Hessenberg (1904–1959) German mathematician and engineer.

in connection with unsymmetric eigenproblems. An upper Hessenberg matrix of order five has the structure

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ 0 & h_{32} & h_{33} & h_{34} & h_{35} \\ 0 & 0 & h_{43} & h_{44} & h_{45} \\ 0 & 0 & 0 & h_{54} & h_{55} \end{pmatrix}.$$

Performing Gaussian elimination the first step will only affect the first two rows of the matrix. The reduced matrix is again Hessenberg and all the remaining steps are similar to the first. If partial pivoting is used then in the first step either h_{11} or h_{21} will be chosen as pivot. Since these rows have the same structure the Hessenberg form will be preserved during the elimination. Clearly only $t = \frac{1}{2}n(n+1)$ flams are needed. Note that with partial pivoting the elimination will not give a factorization $PA = LU$ with L lower bidiagonal. Whenever we pivot, the interchanges should be applied also to L, which will spread out the elements. Therefore L will be lower triangular with only one nonzero off-diagonal element in each column. However, it is more convenient to leave the elements in L in place.

If $A \in \mathbf{R}^{n\times n}$ is Hessenberg then $\rho_n \leq n$ with partial pivoting. This follows the since at the start of the k stage row $k+1$ of the reduced matrix has not been changed and the elements the pivot row has elements of modulus at most k times the largest element of H.

In the special case when A is a symmetric positive definite banded matrix with upper and lower bandwidth $r = s$, the factor L in the Cholesky factorization $A = LL^T$ has lower bandwidth r. From Algorithm 7.3.2 we easily derive the following banded version:

Algorithm 7.4.2 Band Cholesky Algorithm, column-wise Order.

for $j = 1 : n$
  $p = \max(1, j - r)$;
  for $i = p : j - 1$
    $r_{ij} = \Big(a_{ij} - \sum_{k=p}^{i-1} r_{ki} r_{kj}\Big)/r_{ii}$;
  end
  $r_{jj} = \Big(a_{jj} - \sum_{k=p}^{j-1} r_{kj}^2\Big)^{1/2}$;
end

If $r \ll n$ this algorithm requires about $\frac{1}{2}nr(r+3)$ flops and n square roots. As input we just need the upper triangular part of A, which can be stored in an $n \times (r+1)$ array.

7.4.3 Tridiagonal Linear Systems

A matrix of the form

$$A = \begin{pmatrix} a_1 & c_2 & & & \\ b_2 & a_2 & c_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & c_n \\ & & & b_n & a_n \end{pmatrix}. \tag{7.4.3}$$

is called **tridiagonal**. Note that the $3n-2$ nonzero elements in A are conveniently stored in three vectors a, c, and d. A is said to be irreducible if b_i and c_i are nonzero for $i = 2 : n$. Let A be reducible, say $c_k = 0$. Then A can be written as a lower block triangular form

$$A = \begin{pmatrix} A_1 & 0 \\ L_1 & A_2 \end{pmatrix},$$

where A_1 and A_2 are tridiagonal. If A_1 or A_2 is reducible then this blocking can be applied recursively until a block form with irreducible tridiagonal blocks is obtained..

If Gaussian elimination with partial pivoting is applied to A then a factorization $PA = LU$ is obtained, where L has at most one nonzero element below the diagonal in each column and U has upper bandwidth two (cf. the Hessenberg case in Example 7.4.2). If A is diagonally dominant, then no pivoting is requires and the factorization $A = LU$ exists. By Theorem 7.4.4 it has the form

$$A = LU = \begin{pmatrix} 1 & & & & \\ \gamma_2 & 1 & & & \\ & \gamma_3 & \ddots & & \\ & & \ddots & 1 & \\ & & & \gamma_n & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 & c_2 & & & \\ & \alpha_2 & c_3 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{n-1} & c_n \\ & & & & \alpha_n \end{pmatrix}. \tag{7.4.4}$$

By equating elements in A and LU it is verified that the upper diagonal in U equals that in A, and for the other elements in L and U we obtain the recursions

$$\alpha_1 = a_1, \quad \gamma_k = b_k/\alpha_{k-1}, \qquad \alpha_k = a_k - \gamma_k c_k, \quad k = 2 : n. \tag{7.4.5}$$

Note that the elements γ_k and α_k can overwrite b_k and a_k, respectively. The solution to the system $Ax = f$ can then be computed by solving $Ly = f$ by $Ux = y$ by back- and forward-substitution

$$y_1 = f_1, \qquad y_i = f_i - \gamma_i y_{i-1}, \quad i = 2 : n, \tag{7.4.6}$$

$$x_n = y_n/\alpha_n, \qquad x_i = (y_i - c_{i+1}x_{i+1})/\alpha_i, \quad i = n-1 : 1. \tag{7.4.7}$$

The total number of flops is about $1.5n$ for the factorization and $2.5n$ for the solution.

If A is tridiagonal then it is easily proved by induction that $\rho_n \leq 2$ with partial pivoting. This result is a special case of a more general result.

Theorem 7.4.5. [Bothe [10]] *If $A \in \mathbf{C}^{n\times n}$ has upper and lower bandwidth p then the growth factor in GE with partial pivoting satisfies*

$$\rho_n \le 2^{2p-1} - (p-1)2^{p-2}.$$

In particular for a tridiagonal matrix ($p = 1$) $\rho_n \le 2$.

When A is symmetric positive definite and tridiagonal (7.4.3)

$$A = \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & b_n \\ & & & b_n & a_n \end{pmatrix}, \tag{7.4.8}$$

we can write the factorization

$$A = LDL^T, \quad D = \operatorname{diag}(\alpha_1, \ldots, \alpha_n), \tag{7.4.9}$$

where L is as in (7.4.4). The algorithm then reduces to

$$\alpha_1 = a_1, \quad \gamma_k = b_k/\alpha_{k-1}, \qquad \alpha_k = a_k - \gamma_k b_k, \quad k = 2:n. \tag{7.4.10}$$

Sometimes it is more convenient to write

$$A = U^T D^{-1} U, \quad D = \operatorname{diag}(a_1, \ldots, a_n).$$

In the scalar case U is given by (7.4.4) (with $c_k = b_k$), and the elements in U and D are computed from

$$\alpha_1 = a_1, \qquad \alpha_k = a_k - b_k^2/\alpha_{k-1}. \quad k = 2:n. \tag{7.4.11}$$

The recursion (7.4.5) for the LU factorization of a tridiagonal matrix is highly serial. An algorithm for solving tridiagonal systems, which has considerable inherent parallelism, is **cyclic reduction** also called **odd-even reduction**. This is the most preferred method for solving large tridiagonal systems on parallel computers.

The basic step in cyclic reduction is to eliminate all the odd unknowns to obtain a reduced tridiagonal system involving only even numbered unknowns. This process is repeated recursively until a system involving only a small order of unknowns remains. This is then solved separately and the other unknowns can then be computed in a back-substitution process. We illustrate this process on a tridiagonal system $Ax = f$ of order $n = 2^3 - 1 = 7$. If P is a permutation matrix such that $P(1, 2, \ldots, 7) = (1, 3, 5, 7, 2, 4, 6)^T$ the transformed system $PAP^T(Px) = P^T f$, will have the form

$$\left(\begin{array}{cccc|ccc} a_1 & & & & c_2 & & \\ & a_3 & & & b_3 & c_4 & \\ & & a_5 & & & b_5 & c_6 \\ & & & a_7 & & & b_7 \\ \hline b_2 & c_3 & & & a_2 & & \\ & b_4 & c_5 & & & a_4 & \\ & & b_6 & c_7 & & & a_6 \end{array}\right) \left(\begin{array}{c} x_1 \\ x_3 \\ x_5 \\ x_7 \\ \hline x_2 \\ x_4 \\ x_6 \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_3 \\ f_5 \\ f_7 \\ \hline f_2 \\ f_4 \\ f_6 \end{array}\right).$$

It is easily verified that after eliminating the odd variables from the even equations the resulting system is again tridiagonal. Rearranging these as before the system becomes

$$\left(\begin{array}{cc|c} a_2' & & c_4' \\ & a_6' & b_6' \\ \hline b_4' & c_6' & a_4' \end{array}\right) = \begin{pmatrix} x_2 \\ x_6 \\ x_4 \end{pmatrix} = \begin{pmatrix} f_2' \\ f_6' \\ f_4' \end{pmatrix}.$$

After elimination we are left with one equation in one variable

$$a_4'' x_4 = f_4''.$$

Solving for x_4 we can compute x_2 and x_6 from the first two equations in the previous system. Substituting these in the first four equations we get the odd unknowns x_1, x_3, x_5, x_7. Clearly this scheme can be generalized. For a system of dimension $n = 2^p - 1$, p steps are required in the reduction. Note, however, that it is possible to stop at any stage, solve a tridiagonal system and obtain the remaining variables by substitution. Therefore it can be used for any dimension n.

The derivation shows that cyclic reduction is equivalent to Gaussian elimination without pivoting on a reordered system. Therefore it is stable if the matrix is diagonally dominant or symmetric positive definite. In contrast to the conventional algorithm there is some fill during the elimination and about 2.7 times more operations are needed.

Example 7.4.3.

Consider the linear system $Ax = b$, where A is a symmetric positive definite tridiagonal matrix. Then A has positive diagonal elements and the symmetrically scaled matrix DAD, where $D = \operatorname{diag}(d_1, \ldots, d_n)$, $d_i = 1/\sqrt{a_i}$, has unit diagonal elements. After an odd-even permutation the system has the 2×2 block form

$$\begin{pmatrix} I & F \\ F^T & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix}, \tag{7.4.12}$$

with F lower bidiagonal. After block elimination the Schur complement system becomes

$$(I - F^T F)x = d - F^T c.$$

Here $I - F^T F$ is again a positive definite tridiagonal matrix. Thus the process can be repeated recursively.

Boundary value problems, where the solution is subject to *periodic boundary conditions*, often lead to matrices of the form

$$B = \left(\begin{array}{ccccc|c} a_1 & c_2 & & & & b_1 \\ b_2 & a_2 & c_3 & & & \\ & \ddots & \ddots & \ddots & & \\ & & b_{n-1} & a_{n-1} & & c_n \\ \hline c_1 & & & b_n & & a_n \end{array}\right), \tag{7.4.13}$$

which are tridiagonal except for the two corner elements b_1 and c_1. We now consider the is real symmetric case, $b_i = c_i$, $i = 1 : n$ Partitioning B in 2×2 block form as above, we seek a factorization

$$B = \begin{pmatrix} A & u \\ v^T & a_n \end{pmatrix} = \begin{pmatrix} L & 0 \\ y^T & 1 \end{pmatrix} \begin{pmatrix} U & z \\ 0 & d_n \end{pmatrix}$$

where $u = b_1e_1+c_ne_{n-1}$, $v = c_1e_1+b_ne_{n-1}$. Multiplying out we obtain the equations

$$A = LU, \quad u = Lz, \quad v^T = y^TU, \quad a_n = y^Tz + d_n$$

Assuming that no pivoting is required the factorization $A = LU$, where L and U are bidiagonal, is obtained using (7.4.5). The vectors y and z are obtained from the lower triangular systems

$$Lz = b_1e_1 + c_ne_{n-1}, \qquad U^Ty = c_1e_1 + c_ne_{n-1},$$

and $d_n = a_n - y^Tz$. Note that y and z will be full vectors.

Cyclic reduction can be applied to systems $Bx = f$, where B has the tridiagonal form in (7.4.13). If n is even the reduced system obtained after eliminating the odd variables in the even equations will again have the form (7.4.13). For example, when $n = 2^3 = 8$ the reordered system is

$$\left(\begin{array}{cccc|cccc} a_1 & & & & c_2 & & & b_1 \\ & a_3 & & & b_3 & c_4 & & \\ & & a_5 & & & b_5 & c_6 & \\ & & & a_7 & & & b_7 & c_8 \\ \hline b_2 & c_3 & & & a_2 & & & \\ & b_4 & c_5 & & & a_4 & & \\ & & b_6 & c_7 & & & a_6 & \\ c_1 & & & b_8 & & & & a_8 \end{array}\right) \left(\begin{array}{c} x_1 \\ x_3 \\ x_5 \\ x_7 \\ \hline x_2 \\ x_4 \\ x_6 \\ x_8 \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_3 \\ f_5 \\ f_7 \\ \hline f_2 \\ f_4 \\ f_6 \\ f_8 \end{array}\right).$$

If $n = 2^p$ the process can applied recursively. After p steps one equation in a single unknown is obtained. Cyclic reduction here does not require extra storage and also has a slightly lower operation count than ordinary Gaussian elimination.

We finally consider the case when A is a **symmetric indefinite tridiagonal** matrix. It would be possible to use LU factorization with partial pivoting, but this destroys symmetry and gives no information about the inertia of A. Instead a block factorization $A = LDL^T$ can be computed using no interchanges as follows. Set $\sigma = \max_{1\le i\le n} |a_{ij}|$ and $\alpha = (\sqrt{5} - 1)/2 \approx= 0.62$. In the first stage we take a_{11} as pivot if $\sigma|a_{11}| \ge a_{21}^2$. Otherwise we take the 2×2 pivot

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

This factorization can be shown to be normwise backward stable and is a good way to solve such symmetric indefinite tridiagonal linear systems.

7.4.4 Inverses of Banded Matrices

It is important to note that *the inverse A^{-1} of a banded matrix in general has no zero elements*. Hence one should never attempt to explicitly compute the elements of the inverse of a band matrix. Since banded systems often have very large dimensions even storing the elements in A^{-1} may be infeasible!

The following theorem states that the lower triangular part of the inverse of an upper Hessenberg matrix has a very simple structure.

Theorem 7.4.6.

Let $H =\in \mathbf{R}^{n\times n}$ be an upper Hessenberg matrix with nonzero elements in the subdiagonal, $h_{i+1,i} \neq 0$, $i = 1 : n-1$. Then there are vectors p and q such that

$$(H^{-1})_{ij} = p_i q_j, \quad i \geq j. \tag{7.4.14}$$

Proof. See Ikebe [46] □

A tridiagonal matrix A is both lower and upper Hessenberg. Hence if A is irreducible it follows that there are vectors x, y, p and q such that

$$(A^{-1})_{ij} = \begin{cases} x_i y_j, & i \leq j, \\ p_i q_j, & i \geq j. \end{cases} \tag{7.4.15}$$

Note that $x_1 \neq 0$ and $y_n \neq 0$, since otherwise the entire first row or last column of A^{-1} would be zero, contrary to the assumption of the nonsingularity of A. The vectors x and y (as well as p and q) are unique up to scaling by a nonzero factor. There is some redundancy in this representation since $x_i y_i = p_i q_i$. It can be shown that $3n-2$ parameters are needed to represent the inverse, which equals the number of nonzero elements in A.

The following algorithm has been suggested by N. J. Higham to compute the vectors x, y, p and q:

1. Compute the LU factorization of A.

2. Use the LU factorization to solve for the vectors y and z, where $A^T y = e_1$ and $Az = e_n$. Similarly solve for p and r, where $Ap = e_1$ and $A^T r = e_n$.

3. Set $q = p_n^{-1} r$ and $x = y_n^{-1} z$.

This algorithm is not foolproof and can fail because of overflow.

Example 7.4.4. Let A be a symmetric, positive definite tridiagonal matrix with elements $a_1 = 1$,

$$a_i = 2, \quad b_i = c_i = -1, \quad i = 2 : 5.$$

Although the Cholesky factor L of A is bidiagonal the inverse

$$A^{-1} = \begin{pmatrix} 5 & 4 & 3 & 2 & 1 \\ 4 & 4 & 3 & 2 & 1 \\ 3 & 3 & 3 & 2 & 1 \\ 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

is full. Here $x = p$, $y = q$, can be determined up to a scaling factor from the first and last columns of A^{-1}.

The inverse of any banded matrix has a special structure related to low rank matrices. The first study of inverse of general banded matrices was Asplund [5].

Review Questions

1. Give an example of matrix multiplication by diagonals.

2. (a) If a is a column vector what is meant by $\operatorname{diag}(a, k)$?

(b) If A is a square matrix what is meant by $\operatorname{diag}(A, k)$?

3. (a) Let $A \in \mathbf{R}^{n\times n}$ be a banded matrix with upper bandwidth p and lower bandwidth q. Show how A can be efficiently stored when computing the LU factorization.

(b) Assuming that the LU factorization can be carried out without pivoting, what are the structures of the resulting L and U factors of A?

(c) What can you say about the structure of the inverses of L and U?

4. Let $A \in \mathbf{R}^{n\times n}$ be a banded matrix with upper bandwidth p and lower bandwidth q. Assuming that the LU factorization of A can be carried out without pivoting, roughly how many operations are needed? You need only give the dominating term when $p, q \ll n$.

5. Give a bound for the growth ratio ρ_n in Gaussian elimination with partial pivoting, when the matrix A is: (a) Hessenberg; (b) tridiagonal.

Problems

1. (a) Let $A, B \in \mathbf{R}^{n\times n}$ have lower (upper) bandwidth r and s respectively. Show that the product AB has lower (upper) bandwidth $r + s$.

(b) An upper Hessenberg matrix H is a matrix with lower bandwidth $r = 1$. Using the result in (a) deduce that the product of H and an upper triangular matrix is again an upper Hessenberg matrix.

2. Show that an irreducible nonsymmetric tridiagonal matrix A can be written $A = DT$, where T is symmetric tridiagonal and $D = \operatorname{diag}(d_k)$ is diagonal with

elements

$$d_1 = 1, \quad d_k = \prod_{j=2}^{k} c_j/b_j, \quad k = 2 : n. \tag{7.4.16}$$

3. (a) Let $A \in \mathbf{R}^{n\times n}$ be a symmetric, tridiagonal matrix such that $\det(A_k) \neq 0$, $k = 1 : n$. Then the decomposition $A = LDL^T$ exists and can be computed by the formulas given in (7.4.10). Use this to derive a recursion formula for computing $\det(A_k)$, $k = 1 : n$.

(b) Determine the largest n for which the symmetric, tridiagonal matrix

$$A = \begin{pmatrix} 2 & 1.01 & & & \\ 1.01 & 2 & 1.01 & & \\ & 1.01 & \ddots & \ddots & \\ & & \ddots & \ddots & 1.01 \\ & & & 1.01 & 2 \end{pmatrix} \in \mathbf{R}^{n\times n}$$

is positive definite.

4. (a) Show that for $\lambda \geq 2$ it holds that $B = \mu LL^T$, where

$$B = \begin{pmatrix} \mu & -1 & & & \\ -1 & \lambda & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \lambda & -1 \\ & & & -1 & \lambda \end{pmatrix}, \qquad L = \begin{pmatrix} 1 & & & & \\ -\sigma & 1 & & & \\ & -\sigma & \ddots & & \\ & & \ddots & 1 & \\ & & & -\sigma & 1 \end{pmatrix},$$

and

$$\mu = \lambda/2 \pm (\lambda^2/4 - 1)^{1/2}, \quad \sigma = 1/\mu.$$

Note that L has constant diagonals.

(b) Suppose we want to solve and equation system $Ax = b$, where the matrix A differs from B in the element (1,1),

$$A = B + \delta e_1 e_1^T, \quad \delta = \lambda - \mu, \quad e_1^T = (1, 0, \ldots, 0).$$

Show, using the Sherman–Morrison formula (7.1.26), that the solution $x = A^{-1}b$ can be computed from

$$x = y - \gamma L^{-T} f, \quad \gamma = \delta(e_1^T y)/(\mu + \delta f^T f)$$

where y and f satisfies $\mu LL^T y = b$, $Lf = e_1$.

5. Consider the symmetric tridiagonal matrix

$$A_n = \begin{pmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & 4 & 1 \\ & & & 1 & 4 \end{pmatrix}.$$

For $n = 20, 40$ use the Cholesky factorization of A_n and Higham's algorithm to determine vectors x and y so that $(A_n^{-1})_{ij} = x_i y_j$ for $i, j = 1 : n$. Verify that there is a range of approximately θ^n in the size of the components of these vectors, where $\theta = 2 + \sqrt{3}$.

5. (a) Write a function implementing the multiplication $C = AB$, where $A = \operatorname{diag}(a, r)$ and $B = \operatorname{diag}(b, s)$ both consist of a single diagonal. Use the formulas in Lemma 7.4.2.

(b) Write a function for computing the product $C = AB$ of two banded matrices using the $w_1 w_2$ calls to the function in (a), where and w_2 are the bandwidth of A and B, respectively.

6. Derive expressions for computing δ_k, $k = 1 : n-1$ and α_n in the factorization of the periodic tridiagonal matrix A in (7.4.13).

7. Let B be a symmetric matrix of the form (7.4.13). Show that

$$B = T + \sigma u u^T, \quad u = (1, 0, \ldots, 0, -1)^T.$$

where T is a certain symmetric, tridiagonal matrix. What is σ and T. Derive an algorithm for computing L by modifying the algorithm (7.4.10).

7.5 Perturbation Theory and Condition Estimation

7.5.1 Component-Wise Perturbation Analysis

In Sec. 7.1.8 bounds were derived for the perturbation in the solution x to a linear system $Ax = b$, when the data A and b are perturbed. Sharper bounds can often be obtained in case if the data is subject to the perturbations, which are bounded component-wise. Assume that

$$|\delta a_{ij}| \le \omega e_{ij}, \quad |\delta b_i| \le \omega f_i. \quad i, j = 1 : n,$$

where $e_{ij} \ge 0$ and $f_i \ge 0$ are known. These bounds can be written as

$$|\delta A| \le \omega E, \quad |\delta b| \le \omega f, \tag{7.5.1}$$

where the absolute value of a matrix A and vector b is defined by

$$|A|_{ij} = (|a_{ij}|), \qquad |b|_i = (|b_i|).$$

The partial ordering "$\le$" for matrices A, B and vectors x, y, is to be interpreted component-wise[19] It is easy to show that if $C = AB$, then

$$|c_{ij}| \le \sum_{k=1}^{n} |a_{ik}|\,|b_{kj}|,$$

and hence $|C| \le |A|\,|B|$. A similar rule $|Ax| \le |A|\,|x|$ holds for matrix-vector multiplication.

For deriving the componentwise bounds we need the following result.

[19]Note that $A \le B$ in other contexts means that $B - A$ is positive semidefinite.

Lemma 7.5.1.

Let $F \in \mathbf{R}^{n\times n}$ be a matrix for which $\| \, |F| \, \| < 1$. Then the matrix $(I - |F|)$ is nonsingular and

$$|(I - F)^{-1}| \leq (I - |F|)^{-1}. \tag{7.5.2}$$

Proof. The nonsingularity follows form Lemma 7.1.15. Using the identity $(I - F)^{-1} = I + F(I - F)^{-1}$ we obtain

$$|(I - F)^{-1}| \leq I + |F||(I - F)^{-1}|$$

from which the inequality (7.5.2) follows. □

Theorem 7.5.2.

Consider the perturbed linear system $(A + \delta A)(x + \delta x) = b + \delta b$, where A is nonsingular. Assume that δA and δb satisfy the componentwise bounds in (7.5.1) and that

$$\omega \| \, |A^{-1}| \, E \, \| < 1.$$

Then $(A + \delta A)$ is nonsingular and

$$\|\delta x\| \leq \frac{\omega}{1 - \omega\kappa_E(A)} \| \, |A^{-1}|(E \, |x| + f)\|, \tag{7.5.3}$$

where $\kappa_E(A) = \| \, |A^{-1}|E\|$.

Proof. Taking absolute values in (7.1.62) gives

$$|\delta x| \leq |(I + A^{-1}\delta A)^{-1}| \, |A^{-1}|(|\delta A||x| + |\delta b|). \tag{7.5.4}$$

Using Lemma 7.5.1 it follows from the assumption that the matrix $(I - |A^{-1}|\delta A|)$ is nonsingular and from (7.5.4) we get

$$|\delta x| \leq (I - |A^{-1}||\delta A|)^{-1} \, |A^{-1}|(|\delta A||x| + |\delta b|).$$

Using the componentwise bounds in (7.5.1) we get

$$|\delta x| \leq \omega(I - \omega|A^{-1}|E)^{-1}|A^{-1}|(E|x| + f), \tag{7.5.5}$$

provided that $\omega\kappa_E(A) < 1$ Taking norms in (7.5.5) and using Lemma 7.1.15 with $F = A^{-1}\delta A$ proves (7.5.3). □

Taking $E = |A|$ and $f = |b|$ in (7.5.1) corresponds to bounds for the **componentwise relative errors** in A and b,

$$|\delta A| \leq \omega|A|, \qquad |\delta b| \leq \omega|b|. \tag{7.5.6}$$

For this special case Theorem 7.5.2 gives

$$\|\delta x\| \leq \frac{\omega}{1 - \omega\kappa_{|A|}(A)} \| \, |A^{-1}|(|A| \, |x| + |b|) \, \|, \tag{7.5.7}$$

where

$$\kappa_{|A|}(A) = \| \, |A^{-1}||A| \, \|, \tag{7.5.8}$$

(or cond (A)) is the **Bauer–Skeel condition number** of the matrix A. Note that since $|b| \leq |A| \, |x|$, it follows that

$$\|\delta x\| \leq 2\omega \| \, |A^{-1}||A| \, |x| \, \| + O(\omega^2) \leq 2\omega \kappa_{|A|}(A)\|x\| + O(\omega^2).$$

If $\hat{A} = DA$, $\hat{b} = Db$ where $D > 0$ is a diagonal scaling matrix, then $|\hat{A}^{-1}| = |A^{-1}||D^{-1}|$. Since the perturbations scale similarly, $\delta\hat{A} = D\delta A$, $\delta\hat{b} = D\delta b$, it follows that

$$|\hat{A}^{-1}||\delta\hat{A}| = |A^{-1}||\delta A|, \qquad |\hat{A}^{-1}||\delta\hat{b}| = |A^{-1}||\delta b|.$$

Thus the bound in (7.5.7) and also $\kappa_{|A|}(A)$ are *invariant under row scalings*.

For the l_1-norm and l_∞-norm it holds that

$$\kappa_{|A|}(A) = \| \, |A^{-1}||A| \, \| \leq \| \, |A^{-1}| \, \| \, \| \, |A| \, \| = \|A^{-1}\| \, \|A\| = \kappa(A),$$

i.e., the solution of $Ax = b$ is no more badly conditioned with respect to the component-wise relative perturbations than with respect to normed perturbations. On the other hand, it is possible for $\kappa_{|A|}(A)$ to be much smaller than $\kappa(A)$.

The analysis in Sec. 7.1.8 may not be adequate, when the perturbations in the elements of A or b are of different magnitude, as illustrated by the following example.

Example 7.5.1.

The linear system $Ax = b$, where

$$A = \begin{pmatrix} 1 & 10^4 \\ 1 & 10^{-4} \end{pmatrix}, \quad b = \begin{pmatrix} 10^4 \\ 1 \end{pmatrix},$$

has the approximate solution $x \approx (1, 1)^T$. Assume that the vector b is subject to a perturbation δb such that $|\delta b| \leq (1, 10^{-4})^T$. Using the ∞-norm we have $\|\delta b\|_\infty = 1$, $\|A^{-1}\|_\infty = 1$ (neglecting terms of order 10^{-8}). Theorem 7.1.18 then gives the gross overestimate $\|\delta x\|_\infty \leq 1$.

Multiplying the first equation by 10^{-4}, we get an equivalent system $\hat{A}x = \hat{b}$ where

$$\hat{A} = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 10^{-4} \end{pmatrix}, \quad \hat{b} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

The perturbation in the vector b is now $|\delta\hat{b}| \leq 10^{-4}(1, 1)^T$, and from $\|\delta\hat{b}\|_\infty = 10^{-4}$, $\|(\hat{A})^{-1}\|_\infty = 1$, we get the sharp estimate $\|\delta x\|_\infty \leq 10^{-4}$. The original matrix A is only **artificially ill-conditioned**. By a scaling of the equations we obtain a well-conditioned system. How to scale linear systems for Gaussian elimination is a surprisingly intricate problem, which is further discussed in Sec. 7.6.3.

Consider the linear systems in Example 7.5.1. Neglecting terms of order 10^{-8} we have

$$|\hat{A}^{-1}||\hat{A}| = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 10^{-4} \end{pmatrix} \begin{pmatrix} 10^{-4} & 1 \\ 1 & 10^{-4} \end{pmatrix} = \begin{pmatrix} 1 & 2 \cdot 10^{-4} \\ 2 \cdot 10^{-4} & 1 \end{pmatrix},$$

By the scaling invariance $\operatorname{cond}(A) = \operatorname{cond}(\hat{A}) = 1 + 2 \cdot 10^{-4}$ in the ∞-norm. Thus the componentwise condition number correctly reveals that the system is well-conditioned for componentwise small perturbations.

7.5.2 Backward Error Bounds

We now derive a simple **a posteriori** bound for the backward error of a computed solution $\bar{x}$. These bounds are usually much sharper than a priori bounds and hold regardless of the algorithm used to compute $\bar{x}$.

Given $\bar{x}$, there are an infinite number of perturbations δA and δb for which $(A + \delta A)\bar{x} = b + \delta b$ holds. Clearly δA and δb must satisfy

$$\delta A\bar{x} - \delta b = b - A\bar{x} = r,$$

where $r = b - A\bar{x}$ is the residual vector corresponding to the computes solution. An obvious choice is to take $\delta A = 0$, and $\delta b = -r$. If we instead take $\delta b = 0$, we get the following result.

Theorem 7.5.3.

Let $\bar{x}$ be a purported solution to $Ax = b$, and set $r = b - A\bar{x}$. Then if

$$\delta A = r\bar{x}^T/\|\bar{x}\|_2^2, \tag{7.5.9}$$

$\bar{x}$ satisfies $(A + \delta A)\bar{x} = b$ and this has the smallest l_2-norm $\|\delta A\|_2 = \|r\|_2/\|\bar{x}\|_2$ of any such δA.

Proof. Clearly $\bar{x}$ satisfies $(A + \delta A)\bar{x} = b$ if and only if $\delta A\bar{x} = r$. For any such δA it holds that $\|\delta A\|_2\|\bar{x}\|_2 \geq \|r\|_2$ or $\|\delta A\|_2 \geq \|r\|_2/\|\bar{x}\|_2$. For the particular δA given by (7.5.9) we have $\delta A\bar{x} = r\bar{x}^T\bar{x}/\|\bar{x}\|^2 = r$. From

$$\|r\bar{x}^T\|_2 = \sup_{\|y\|_2=1} \|r\bar{x}^T y\|_2 = \|r\|_2 \sup_{\|y\|_2=1} |\bar{x}^T y| = \|r\|_2\|\bar{x}\|_2,$$

it follows that $\|\delta A\|_2 = \|r\|_2/\|\bar{x}\|_2$ and hence the δA in (7.5.9) is of minimum l_2-norm. □

Similar bounds for the l_1-norm and l_∞-norm are given in Problem 5.

It is often more useful to consider the **component-wise backward error** ω of a computed solution. The following theorem shows that also this can be cheaply computed

Theorem 7.5.4. (Oettli and Prager [1964]).

Let $r = b - A\bar{x}$, E and f be nonnegative and set

$$\omega = \max_i \frac{|r_i|}{(E|\bar{x}| + f)_i}, \tag{7.5.10}$$

where $0/0$ *is interpreted as* 0. *If* $\omega \neq \infty$, *there is a perturbation* δA *and* δb *with*

$$|\delta A| \leq \omega E, \qquad |\delta b| \leq \omega f, \tag{7.5.11}$$

such that

$$(A + \delta A)\bar{x} = b + \delta b. \tag{7.5.12}$$

Moreover, ω *is the smallest number for which such a perturbation exists.*

Proof. From (7.5.10) we have

$$|r_i| \leq \omega(E|\bar{x}| + f)_i,$$

which implies that $r = D(E|\bar{x}| + f)$, where $|D| \leq \omega I$. It is then readily verified that

$$\delta A = DE \operatorname{diag}\big(\operatorname{sign}(\bar{x}_1), \ldots, \operatorname{sign}(\bar{x}_n)\big), \qquad \delta b = -Df$$

are the required backward perturbations.

Further, given perturbations δA and δb satisfying equations (7.5.11)–(7.5.12) for some ω we have

$$|r| = |b - A\bar{x}| = |\delta A\bar{x} - \delta b| \leq \omega(E|\bar{x}| + f).$$

Hence $\omega \geq |r_i|/(E|\bar{x}| + f)_i$, which shows that ω as defined by (7.5.10) is optimal. □

In particular we can take $E = |A|$, and $f = |b|$ in Theorem 7.1.18, to get an expression for the component-wise relative backward error ω of a computed solution. This can then be used in (7.5.6) or (7.5.7) to compute a bound for $\|\delta x\|$.

Example 7.5.2. Consider the linear system $Ax = b$, where

$$A = \begin{pmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{pmatrix}, \qquad b = \begin{pmatrix} 0.8642 \\ 0.1440 \end{pmatrix}.$$

Suppose that we are given the approximate solution $\bar{x} = (0.9911, -0.4870)^T$. The residual vector corresponding to $\bar{x}$ is very small,

$$r = b - A\bar{x} = (-10^{-8}, 10^{-8})^T.$$

However, not a single figure in $\bar{x}$ is correct! The *exact* solution is $x = (2, -2)^T$, as can readily verified by substitution. *Although a zero residual implies an exact solution, a small residual alone does not necessarily imply an accurate solution.* (Compute the determinant of A and then the inverse A^{-1}!)

It should be emphasized that the system in this example is contrived. In practice one would be highly unfortunate to encounter such an ill-conditioned 2×2 matrix.[20]

[20] As remarked by a prominent expert in error-analysis "Anyone unlucky enough to encounter this sort of calamity has probably already been run over by a truck"!

7.5.3 Estimating Condition Numbers

The perturbation analysis has shown that the norm-wise relative perturbation in the solution x of a linear system can be bounded by

$$\|A^{-1}\| \left(\|\delta A\| + \|\delta b\|/\|x\|\right), \tag{7.5.13}$$

or, in case of componentwise analysis, by

$$\| \, |A^{-1}(| \, |E|x| + f|) \, \|. \tag{7.5.14}$$

To compute these upper bounds exactly is costly since $2n^3$ flops are required to compute A^{-1}, even if the LU factorization of A is known (see Section 7.2.5). In practice, it will will suffice with an *estimate* of $\|A^{-1}\|$ (or $\| \, |A^{-1}| \, \|$, which need not be very precise.

The first algorithm for condition estimation to be widely used was suggested by Cline, Moler, Stewart, and Wilkinson [14]. It is based on computing

$$y = (A^T A)^{-1} u = A^{-1}(A^{-T} u) \tag{7.5.15}$$

by solving the two systems $A^T w = u$ and $Ay = w$. A *lower bound* for $\|A^{-1}\|$ is then given by

$$\|A^{-1}\| \geq \|y\|/\|w\|. \tag{7.5.16}$$

If an LU factorization of A is known this only requires $O(n^2)$ flops. The computation of $y = A^{-T} w$ involves solving the two triangular systems

$$U^T v = u, \qquad L^T w = v.$$

Similarly the vector y and w are obtained by solving the triangular systems

$$Lz = w, \qquad Uy = z,$$

For (7.5.16) to be a reliable estimate the vector u must be carefully chosen so that it reflects any possible ill-conditioning of A. Note that if A is ill-conditioned this is likely to be reflected in U, whereas L, being unit upper triangular, tends to be well-conditioned. To enhance the growth of v we take $u_i = \pm 1$, $i = 1 : n$, where the sign is chosen to maximize $|v_i|$. The final estimate is taken to be

$$1/\kappa(A) \leq \|w\|/(\|A\|\|y\|), \tag{7.5.17}$$

since then a singular matrix is signaled by zero rather than by ∞ and overflow is avoided. We stress that (7.5.17) always *underestimates* $\kappa(A)$. Usually the l_1-norm is chosen because the matrix norm $\|A\|_1 = \max_j \|a_j\|_1$ can be computed from the columns a_j of A. This is often referred to as the LINPACK condition estimator. A detailed description of an implementation is given in the LINPACK Guide, Dongarra et al. [18, 1979, pp. 11-13]. In practice it has been found that the LINPACK condition estimator seldom is off by a factor more than 10. However, counter examples can be constructed showing that it can fail. This is perhaps to be expected for any estimator using only $O(n^2)$ operations.

Equation (7.5.15) can be interpreted as performing one step of the inverse power method on A^TA using the special starting vector u. As shown in Section 10.4.2 this is a standard method for computing the largest singular value $\sigma_1(A^{-1}) = \|A^{-1}\|_2$. An alternative to starting with the vector u is to use a *random* starting vector and perhaps carrying out several steps of inverse iteration with A^TA.

An alternative 1-norm condition estimator has been devised by Hager [36] and improved by Higham [39]. This estimates

$$\|B\|_1 = \max_j \sum_{i=1}^{n} |b_{ij}|,$$

assuming that Bx and B^Tx can be computed for an arbitrary vector x. It can also be used to estimate the infinity norm since $\|B\|_\infty = \|B^T\|_1$. It is based on the observation that

$$\|B\|_1 = \max_{x \in S} \|Bx\|_1, \quad S = \{x \in \mathbf{R}^n \mid \|x\|_1 \le 1\}.$$

is the maximum of a convex function $f(x) = \|Bx\|_1$ over the convex set S. This implies that the maximum is obtained at an extreme point of S, i.e. one of the $2n$ points

$$\{\pm e_j \mid j = 1 : n\},$$

where e_j is the jth column of the unit matrix. If $y_i = (Bx)_i \neq 0$, $\forall i$, then $f(x)$ is differentiable and by the chain rule the gradient is

$$\partial f(x) = \xi^T B, \quad \xi_i = \begin{cases} +1 & \text{if } y_i > 0, \\ -1 & \text{if } y_i < 0. \end{cases}$$

If $y_i = 0$, for some i, then $\partial f(x)$ is a subgradient of f at x. Note that the subgradient is not unique. Since f is convex, the inequality

$$f(y) \ge f(x) + \partial f(x)(y - x), \quad \forall x, y \in \mathbf{R}^n.$$

is always satisfied.

The algorithm starts with the vector $x = n^{-1}e = n^{-1}(1, 1, \ldots, 1)^T$, which is on the boundary of S. We set $\partial f(x) = z^T$, where $z = B^T\xi$, and find an index j for which $|z_j| = \max_i |z_i|$. It can be shown that $|z_j| \le z^Tx$ then x is a local maximum. If this inequaity is satisfied then we stop. By the convexity of $f(x)$ and the fact that $f(e_j) = f(-e_j)$ we conclude that $f(e_j) > f(x)$. Replacing x by e_j we repeat the process. Since the estimates are strictly increasing each vertice of S is visited at most once. The iteration must therefore terminate in a finite number of steps. It has been observed that usually it terminates after just four iterations with the exact value of $\|B\|_1$.

We now show that the final point generated by the algorithm is a local maximum. Assume first that $(Bx)_i \neq 0$ for all i. Then $f(x) = \|Bx\|_1$ is linear in a neighborhood of x. It follows that x is a local maximum of $f(x)$ over S if and only if

$$\partial f(x)(y - x) \le 0, \quad \forall y \in S.$$

If y is a vertex of S, then $\partial f(x)y = \pm\partial f(x)_i$, for some i since all but one component of y is zero. If $|\partial f(x)_i| \leq \partial f(x)x$, for all i, it follows that $\partial f(x)(y-x) \leq 0$ whenever y is a vertex of S. Since S is the convex hull of its vertices it follows that $\partial f(x)(y-x) \leq 0$, for all $y \in S$. Hence x is a locla maximum. In case some component of Bx is zero the above argument must be slightly modified; see Hager [36].

Algorithm 7.5.1 Hager's 1-norm estimator.

$x = n^{-1}e$
repeat
  $y = Bx$
  $\xi = \text{sign}\,(y)$
  $z = B^T\xi$
  if $\|z\|_\infty \leq z^T x$
    $\gamma = \|y\|_1$; quit
  end
  $x = e_j, \text{where} |z_j| = \|z\|_\infty$
end

To use this algorithm to estimate $\|A^{-1}\|_1 = \| \, |A^{-1}| \, \|_1$, we take $B = A^{-1}$. In each iteration we are then required to solve systems $Ay = x$ and $A^T z = \xi$.

It is less obvious that Hager's estimator can also be used to estimate the componentwise analysis 7.5.13). The problem is then to estimate a n expression of the form $\| \, |A^{-1}|g\|_\infty$, for a given vector $g > 0$. Using a clever trick devised by Arioli, Demmel and Duff [3], this can be reduced to estimating $\|B\|_1$ where

$$B = (A^{-1}G)^T, \quad G = \text{diag}\,(g_1, \ldots, g_n) > 0.$$

We have $g = Ge$ where $e = (1, 1, \ldots, 1)^T$ and hence

$$\| \, |A^{-1}|g\|_\infty = \| \, |A^{-1}|Ge\|_\infty = \| \, |A^{-1}G|e\|_\infty = \| \, |A^{-1}G|\|_\infty = \|(A^{-1}G)^T\|_1,$$

where in the last step we have used that the ∞ norm is absolute (see Sec. 7.1.5). Since Bx and $B^T y$ can be found by solving linear systems involving A^T and A the work involved is similar to that of the LINPACK estimator. This together with ω determined by (7.5.10) gives an approximate bound for the error in a computed solution $\bar{x}$. Hager's condition estimator is used MATLAB .

We note that the unit lower triangular matrices L obtained from Gaussian elimination with pivoting are not arbitrary but their off-diagonal elements satisfy $|l_{ij}| \leq 1$. When Gaussian elimination without pivoting is applied to a row diagonally dominant matrix it gives a row diagonally dominant upper triangular factor $U \in \mathbf{R}^{n\times n}$ satisfying

$$|u_{ii}| \geq \sum_{j=i+1}^{n} |u_{ij}|, \quad i = 1 : n-1. \tag{7.5.18}$$

and it holds that $cond(U) \leq 2n - 1$; (see [41, Lemma 8.8].

Definition 7.5.5. *For any triangular matrix T the* **comparison matrix** *is*

$$M(T) = (m_{ij}), \qquad m_{ij} = \begin{cases} |t_{ii}|, & i = j; \\ -|t_{ij}|, & i \neq j; \end{cases}$$

Review Questions

1. How is the condition number $\kappa(A)$ of a matrix A defined? How does $\kappa(A)$ relate to perturbations in the solution x to a linear system $Ax = b$, when A and b are perturbed? Outline roughly a cheap way to estimate $\kappa(A)$.

Problems

1. (a) Compute the inverse A^{-1} of the matrix A in Problem 6.4.1 and determine the solution x to $Ax = b$ when $b = (4, 3, 3, 1)^T$.

(b) Assume that the vector b is perturbed by a vector δb such that $\|\delta b\|_\infty \leq 0.01$. Give an upper bound for $\|\delta x\|_\infty$, where δx is the corresponding perturbation in the solution.

(c) Compute the condition number $\kappa_\infty(A)$, and compare it with the bound for the quotient between $\|\delta x\|_\infty/\|x\|_\infty$ and $\|\delta b\|_\infty/\|b\|_\infty$ which can be derived from (b).

2. Show that the matrix A in Example 7.5.2 has the inverse

$$A^{-1} = 10^8 \begin{pmatrix} 0.1441 & -0.8648 \\ -0.2161 & 1.2969 \end{pmatrix},$$

and that $\kappa_\infty = \|A\|_\infty \|A^{-1}\|_\infty = 2.1617 \cdot 1.5130 \cdot 10^8 \approx 3.3 \cdot 10^8$, which shows that the system is "perversely" ill-conditioned.

3. (Higham [41, p. 144]) Consider the triangular matrix

$$U = \begin{pmatrix} 1 & 1 & 0 \\ 0 & \epsilon & \epsilon \\ 0 & 0 & 1 \end{pmatrix}.$$

Show that $\text{cond}\,(U) = 5$ but $\text{cond}\,(U^T) = 1 + 2/\epsilon$. This shows that a triangular system can be much worse conditioned than its transpose.

4. Let the matrix $A \in \mathbf{R}^{n\times n}$ be nonnegative, and solve $A^T x = e$, where $e = (1, 1, \ldots, 1)^T$. Show that then $\|A^{-1}\|_1 = \|x\|_\infty$.

5. Let $\bar{x}$ be a computed solution and $r = b - A\bar{x}$ the corresponding residual. Assume that δA is such that $(A + \delta A)\bar{x} = b$ holds exactly. Show that the error of minimum l_1-norm and l_∞-norm respectively are given by

$$\delta A = r(s_1, \ldots, s_n)/\|\bar{x}\|_1, \quad \delta A = r(0, \ldots, 0, s_m, 0, \ldots, 0)/\|\bar{x}\|_\infty,$$

where $\|\bar{x}\|_\infty = |x_m|$, and $s_i = 1$, if $x_i \geq 0$; $s_i = -1$, if $x_i < 0$.

7.6 Rounding Error Analysis

7.6.1 Floating Point Arithmetic

In this section we first recall some basic results for floating point computations. For a detailed treatment of IEEE floating point standard and floating point arithmetic we refer Sections 2.2–2.3, Volume I.

If x and y are two floating point numbers, we denote by

$$fl\,(x+y), \quad fl\,(x-y), \quad fl\,(x\cdot y), \quad fl\,(x/y)$$

the results of floating addition, subtraction, multiplication, and division, which the machine stores in memory (after rounding or chopping). We will in the following assume that underflow or overflow does not occur. and that the following **standard model** for the arithmetic holds:

Definition 7.6.1.
Assume that $x, y \in F$. Then in the **standard model** *it holds*

$$fl\,(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \le u, \tag{7.6.1}$$

where u is the unit roundoff and "op" stands for one of the four elementary operations $+$, $-$, $\cdot$, and $/$.

The standard model holds with the default rounding mode for computers implementing the IEEE 754 standard. In this case we also have

$$fl\,(\sqrt{x}) = \sqrt{x}(1+\delta), \quad |\delta| \le u, \tag{7.6.2}$$

Bounds for roundoff errors for basic vector and matrix operations can easily be derived (Wilkinson [66, pp. 114–118]) using the following basic result:

Lemma 7.6.2. [N. J. Higham [41, Lemma 3.1]]
Let $|\delta_i| \le u$, $\rho_i = \pm 1$, $i = 1{:}n$, and

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n.$$

If $nu < 1$, then $|\theta_n| < \gamma_n$, where $\gamma_n = nu/(1-nu)$.

For an **inner product** $x^T y$ computed in the natural order we have

$$fl\,(x^T y) = x_1y_1(1+\delta_1) + x_2y_2(1+\delta_2) + \cdots + x_ny_n(1+\delta_n)$$

where

$$|\delta_1| < \gamma_n, \qquad |\delta_r| < \gamma_{n+2-i}, \quad i = 2:n.$$

The corresponding forward error bound becomes

$$|fl\,(x^T y) - x^T y| < \sum_{i=1}^{n} \gamma_{n+2-i}|x_i||y_i| < \gamma_n |x^T||y|, \tag{7.6.3}$$

where $|x|$, $|y|$ denote vectors with elements $|x_i|$, $|y_i|$.This bound is independent of the summation order and is valid also for floating point computation with no guard digit rounding.

For the outer product xy^T of two vectors $x, y \in \mathbf{R}^n$ it holds that $fl\,(x_i y_j) = x_i y_j(1+\delta_{ij})$, $\delta_{ij} \le u$, and so

$$|fl\,(xy^T) - xy^T| \le u\,|xy^T|. \tag{7.6.4}$$

This is a satisfactory result for many purposes, but the computed result is not in general a rank one matrix and it is not possible to find perturbations Δx and Δy such that $fl(xy^T) = (x+\Delta x)(x+\Delta y)^T$. This shows that matrix multiplication in floating point arithmetic is not always backward stable!

Similar error bounds can easily be obtained for matrix multiplication. Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$, and denote by $|A|$ and $|B|$ matrices with elements $|a_{ij}|$ and $|b_{ij}|$. Then it holds that

$$|fl\,(AB) - AB| < \gamma_n |A||B|. \tag{7.6.5}$$

where the inequality is to be interpreted elementwise. Often we shall need bounds for some norm of the error matrix. From (7.6.5) it follows that

$$\|fl\,(AB) - AB\| < \gamma_n \| \,|A|\,\| \,\| \,|B|\,\|. \tag{7.6.6}$$

Hence, for the 1-norm, ∞-norm and the Frobenius norm we have

$$\|fl\,(AB) - AB\| < \gamma_n \|A\|\,\|B\|. \tag{7.6.7}$$

but unless A and B have non-negative elements, we have for the 2-norm only the weaker bound

$$\|fl\,(AB) - AB\|_2 < n\gamma_n \|A\|_2\,\|B\|_2. \tag{7.6.8}$$

In many matrix algorithms there repeatedly occurs expressions of the form

$$y = \Big(c - \sum_{i=1}^{k-1} a_i b_i\Big)\Big/ d.$$

A simple extension of the roundoff analysis of an inner product in Sec, 2.4.1 (cf. Problem 2.3.7) shows that if the term c is added last, then the computed $\bar{y}$ satisfies

$$\bar{y}d(1+\delta_k) = c - \sum_{i=1}^{k-1} a_i b_i (1+\delta_i), \tag{7.6.9}$$

where

$$|\delta_1| \le \gamma_{k-1}, \qquad |\delta_i| \le \gamma_{k+1-i}, \quad i = 2:k-1, \qquad |\delta_k| \le \gamma_2. \tag{7.6.10}$$

and $\gamma_k = ku/(1-ku)$ and u is the unit roundoff. Note that in order to prove a backward error result for GE, that does not perturb the right hand side vector b,

we have formulated the result so that c is not perturbed. It follows that the forward error satisfies

$$\left|\bar{y}d - c + \sum_{i=1}^{k-1} a_i b_i\right| \leq \gamma_k\Big(|\bar{y}d| + \sum_{i=1}^{k-1} |a_i||b_i|\Big), \tag{7.6.11}$$

and this inequality holds independent of the summation order.

7.6.2 Error Analysis of Gaussian Elimination

In the practical solution of a linear system of equations, rounding errors are introduced in each arithmetic operation and cause errors in the computed solution. In the early days of the computer era around 1946 many mathematicians were pessimistic about the numerical stability of Gaussian elimination. It was argued that the growth of roundoff errors would make it impractical to solve even systems of fairly moderate size. By the early 1950s experience revealed that this pessimism was unfounded. In practice Gaussian elimination with partial pivoting is a remarkably stable method and has become the universal algorithm for solving dense systems of equations.

The bound given in Theorem 7.2.3 is satisfactory only if the growth factor ρ_n is not too large, but this quantity is only known *after* the elimination has been completed. In order to obtain an *a priori bound* on ρ_n we use the inequality

$$|a_{ij}^{(k+1)}| = |a_{ij}^{(k)} - l_{ik}a_{kj}^{(k)}| \leq |a_{ij}^{(k)}| + |a_{kj}^{(k)}| \leq 2\max_k |\bar{a}_{ij}^{(k)}|,$$

valid If partial pivoting is employed. By induction this gives the upper bound $\rho_n \leq 2^{n-1}$, which is attained for matrices $A_n \in \mathbf{R}^{n\times n}$ of the form

$$A_4 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & 1 \end{pmatrix}. \tag{7.6.12}$$

Already for $n = 54$ we can have $\rho_n = 2^{53} \approx 0.9 \cdot 10^{16}$ and can lose all accuracy using IEEE double precision ($u = 1.11 \cdot 10^{-16}$). Hence the *worst-case behavior of partial pivoting is very unsatisfactory.*

For complete pivoting, Wilkinson [65, 1961] has proved that

$$\rho_n \leq (n \cdot 2^1 3^{1/2} 4^{1/3} \cdots n^{1/(n-1)})^{1/2} < 1.8\sqrt{n}n^{\frac{1}{4}\log n},$$

and that this bound is not attainable. This bound is *much* smaller than that for partial pivoting, e.g., $\rho_{50} < 530$. It was long conjectured that $\rho_n \leq n$ for real matrices and complete pivoting. This was finally disproved in 1991 when a matrix of order 13 was found for which $\rho_n = 13.0205$. A year later a matrix of order 25 was found for which $\rho_n = 32.986$.

Although complete pivoting has a much smaller worst case growth factor than partial pivoting it is more costly. Moreover, complete (as well as rook) pivoting has the drawback that it cannot be combined with the more efficient blocked methods of GE (see Sec. 7.5.3). Fortunately from decades of experience and extensive

experiments it can be concluded that substantial growth in elements using partial pivoting occurs only for a tiny proportion of matrices arising naturally. We quote Wilkinson [66, pp. 213–214].

> *It is our experience that any substantial increase in the size of elements of successive $A^{(k)}$ is extremely uncommon even with partial pivoting. No example which has arisen naturally has in my experience given an increase by a factor as large as* 16.

So far only two exceptions to the experience related by Wilkinson have been reported. One concerns linear systems arising from a class of two-point boundary value problems, when solved by the shooting method. Another is the class of linear systems arising from a quadrature method for solving a certain Volterra integral equation. These examples show that GE with partial pivoting cannot be unconditionally trusted. When in doubt some safeguard like monitoring the element growth should be incorporated. Another way of checking and improving the reliability of GE with partial pivoting is iterative refinement, which is discussed in Sec. 7.7.4.

Why large element growth rarely occurs with partial pivoting is still not fully understood. Trefethen and Schreiber [62] have shown that for certain distributions of random matrices the average element growth was close to $n^{2/3}$ for partial pivoting.

We now give a component-wise roundoff analysis for the LU factorization of A. Note that all the variants given in Sec. 7.2 for computing the LU factorization of a matrix will essentially lead to the same error bounds, since each does the same operations with the same arguments. Note also that since GE with pivoting is equivalent to GE without pivoting on a permuted matrix, we need not consider pivoting.

Theorem 7.6.3.

If the LU factorization of A runs to completion then the computed factors $\bar{L}$ and $\bar{U}$ satisfy

$$A + E = \bar{L}\bar{U}, \quad |E| \le \gamma_n |\bar{L}|\,|\bar{U}|, \tag{7.6.13}$$

where $\gamma_n = nu/(1-nu)$.

Proof. In the algorithms in Sec 7.2.6) we set $l_{ii} = 1$ and compute the other elements in L and U from the equations

$$u_{ij} = a_{ij} - \sum_{p=1}^{i-1} l_{ip}u_{pj}, \quad j \ge i;$$

$$l_{ij} = \Big(a_{ij} - \sum_{p=1}^{j-1} l_{ip}u_{pj}\Big)/u_{jj}, \quad i > j,$$

Using (7.6.11) it follows that the computed elements $\bar{l}_{ip}$ and $\bar{u}_{pj}$ satisfy

$$\Big|a_{ij} - \sum_{p=1}^{r} \bar{l}_{ip}\bar{u}_{pj}\Big| \le \gamma_r \sum_{p=1}^{r} |\bar{l}_{ip}|\,|\bar{u}_{pj}|, \quad r = \min(i,j).$$

where $\bar{l}_{ii} = l_{ii} = 1$. These inequalities may be written in matrix form □

To prove the estimate the error in a computed solution $\bar{x}$ of a linear system given in Theorem 7.6.3, we must also take into account the rounding errors performed in the solution of the two triangular systems $\bar{L}y = b$, $\bar{U}x = y$. A lower triangular system $Ly = b$ is solved by forward substitution

$$l_{kk}y_k = b_k - \sum_{i=1}^{k-1} l_{ki}y_i, \quad k = 1:n.$$

If we let $\bar{y}$ denote the computed solution, then using (7.6.9) –(7.6.10) it is straightforward to derive a bound for the backward error in solving a triangular system of equations.

Using the bound for the backward error the forward error in solving a triangular system can be estimated. It is a well known fact that the computed solution is far accurate than predicted by the normwise condition number. This has been partly explained by Stewart [60, p. 231] as follows:

> "When a matrix is decomposed by GE with partial pivoting for size, the resulting L-factor tends to be well conditioned while any ill-conditioning in the U-factor tends to be artificial."

This observation does not hold in general, for counter examples exist. However, it is true of many special kinds of triangular matrices.

Theorem 7.6.4. *If the lower triangular system $Ly = b$, $L \in \mathbf{R}^{n\times n}$ is solved by substitution with the summation order outlined above, then the computed solution $\bar{y}$ satisfies*

$$(L + \Delta L)\bar{y} = b, \quad |\Delta l_{ki}| \le \begin{cases} \gamma_2|l_{ki}|, & i = k \\ \gamma_{k+1-i}|l_{ki}|, & i = 1:k-1 \end{cases}, \quad k = 1:n. \tag{7.6.14}$$

Hence $|\Delta L| \le \gamma_n|L|$ and this inequality holds for any summation order.

An analogue result holds for the computed solution to an upper triangular systems. We conclude the backward stability of substitution for solving triangular systems. Note that it is not necessary to perturb the right hand side.

Theorem 7.6.5.

Let $\bar{x}$ be the computed solution of the system $Ax = b$, using LU factorization and substitution. Then $\bar{x}$ satisfies exactly

$$(A + \Delta A)\bar{x} = b, \tag{7.6.15}$$

where δA is a matrix, depending on both A and b, such that

$$|\Delta A| \le \gamma_n(3 + \gamma_n)|\bar{L}|\,|\bar{U}|. \tag{7.6.16}$$

Proof. From Theorem 7.6.4 it follows that the computed $\bar{y}$ and $\bar{x}$ satisfy

$$(\bar{L} + \delta\bar{L})\bar{y} = b, \qquad (\bar{U} + \delta\bar{U})\bar{x} = \bar{y},$$

where

$$|\delta\bar{L}| \le \gamma_n|\bar{L}|, \qquad |\delta\bar{U}| \le \gamma_n|\bar{U}|. \tag{7.6.17}$$

Note that $\delta\bar{L}$ and $\delta\bar{U}$ depend upon b. Combining these results, it follows that the computed solution $\bar{x}$ satisfies

$$(\bar{L} + \delta\bar{L})(\bar{U} + \delta\bar{U})\bar{x} = b,$$

and using equations (7.6.13)–(7.6.17) proves the backward error

$$|\Delta A| \le \gamma_n(3 + \gamma_n)|\bar{L}|\,|\bar{U}|. \tag{7.6.18}$$

for the computed solution $\bar{x}$ given in Theorem 7.6.3. □

Note that although the perturbation δA depends upon b the *bound* on $|\delta A|$ is independent on b. The elements in $\bar{U}$ satisfy $|\bar{u}_{ij}| \le \rho_n\|A\|_\infty$, and with partial pivoting $|\bar{l}_{ij}| \le 1$. Hence

$$\|\,|\bar{L}|\,|\bar{U}|\,\|_\infty \le \tfrac{1}{2}n(n+1)\rho_n,$$

and neglecting terms of order $O((nu)^2)$ in (7.6.18) it follows that

$$\|\delta A\|_\infty \le 1.5n(n+1)\gamma_n\rho_n\|A\|_\infty. \tag{7.6.19}$$

By taking b to be the columns $e_1, e_2, \ldots . e_n$ of the unit matrix in succession we obtain the n computed columns of the inverse X of A. For the kth column we have

$$(A + \Delta A_r)\bar{x}_r = e_r,$$

where we have written ΔA_r to emphasize that the perturbation *is different for each column*. Therefore we cannot say that GE computes the exact inverse corresponding to some matrix $A + \Delta A$! We obtain the estimate

$$\|A\bar{X} - I\|_\infty \le 1.5n(n+1)\gamma_n\rho_n\|A\|_\infty\|\bar{X}\|_\infty. \tag{7.6.20}$$

From $A\bar{X} - I = E$ it follows that $\bar{X} - A^{-1} = A^{-1}E$ and $\|\bar{X} - A^{-1}\|_\infty \le \|A^{-1}\|_\infty\|E\|_\infty$, which together with (7.6.20) can be used to get a bound for the error in the computed inverse. We should stress again that we recommend that computing explicit inverses is avoided.

The residual for the computed solution satisfies $\bar{r} = b - A\bar{x} = \delta A\bar{x}$, and using (7.6.19) it follows that

$$\|\bar{r}\|_\infty \le 1.5n(n+1)\gamma_n\rho_n\|A\|_\infty\|\bar{x}\|_\infty.$$

This shows the remarkable fact that *GE will give a small relative residual even for ill-conditioned systems.* Unless the growth factor is large the quantity

$$\|b - A\bar{x}\|_\infty/(\|A\|_\infty\|\bar{x}\|_\infty)$$

will in practice be of the order nu. It is important to realize that this property of GE is not shared by most other methods for solving linear systems. For example, if we first compute the inverse A^{-1} and then $x = A^{-1}b$ the residual $\bar{r}$ may be much larger even if the accuracy in $\bar{x}$ is about the same.

The error bound in Theorem 7.6.3 is instructive in that it shows that a particularly favourable case is when $|\bar{L}|\,|\bar{U}| = |\bar{L}\bar{U}|$. This is true when $\bar{L}$ and $\bar{U}$ are nonnegative. Then

$$|\bar{L}|\,|\bar{U}| = |\bar{L}\bar{U}| = |A + \Delta A| \leq |A| + |\bar{L}|\,|\bar{U}|,$$

and neglecting terms of order $O((nu)^2)$ we find that the computed $\bar{x}$ satisfies

$$(A + \Delta A)\bar{x} = b, \quad |\Delta A| \leq 3\gamma_n |A|.$$

A class of matrices for which Gaussian elimination without pivoting gives positive factors L and U is the following.

Definition 7.6.6.
A matrix $A \in \mathbf{R}^{n\times n}$ is called **totally positive** *if the determinant of every square submatrix of A is positive.*

It is known (see de Boor and Pinkus [15]) that if A is totally positive, then it has an LU factorization with $L > 0$ and $U > 0$. Since the property of a matrix being totally positive is destroyed under row permutations, *pivoting should not be used when solving such systems.* Totally positive systems occur in spline interpolation.

In many cases there is no a priori bound for the matrix $|\bar{L}|\,|\bar{U}|$ appearing in the componentwise error analysis. It is then possible to compute its ∞-norm in $O(n^2)$ operations without forming the matrix explicitly, since

$$\| \, |\bar{L}|\,|\bar{U}| \, \|_\infty = \| \, |\bar{L}|\,|\bar{U}|e \, \|_\infty = \| \, |\bar{L}|\,(|\bar{U}|e) \, \|_\infty.$$

This useful observation is due to Chu and George [12].

An error analysis for the Cholesky factorization of a symmetric positive definite matrix $A \in \mathbf{R}^{n\times n}$ is similar to that for LU factorization.

Theorem 7.6.7.
Suppose that the Cholesky factorization of a symmetric positive definite matrix $A \in R^{n\times n}$ runs to completion and produces a computed factor $\bar{R}$ and a computed solution $\bar{x}$ to the linear system. Then it holds that

$$A + E_1 = \bar{L}\bar{U}, \quad |E_1| \leq \gamma_{n+1}|\bar{R}^T|\,|\bar{R}|, \tag{7.6.21}$$

and

$$(A + E_2)\bar{x} = b, \quad |E_2| \leq \gamma_{3n+1}|\bar{R}^T|\,|\bar{R}|. \tag{7.6.22}$$

Theorem 7.6.8. [J. H. Wilkinson [67]]

Let $A \in R^{n\times n}$ be a symmetric positive definite matrix. The Cholesky factor of A can be computed without breakdown provided that $2n^{3/2}u\kappa(A) < 0.1$. The computed $\bar{L}$ satisfies

$$\bar{L}\bar{L}^T = A + E, \quad \|E\|_2 < 2.5n^{3/2}u\|A\|_2, \tag{7.6.23}$$

and hence is the exact Cholesky factor of a matrix close to A.

This is essentially the best normwise bounds that can be obtained, although Meinguet [50] has shown that for large n the constants 2 and 2.5 in Theorem 7.6.8 can be improved to 1 and 2/3, respectively.

In practice we can usually expect much smaller backward error in the computed solutions than the bounds derived in this section. It is appropriate to recall here a remark by J. H. Wilkinson (1974):

> "All too often, too much attention is paid to the precise error bound that has been established. The main purpose of such an analysis is either to establish the essential numerical stability of an algorithm or to show why it is unstable and in doing so expose what sort of change is necessary to to make it stable. The precise error bound is not of great importance."

7.6.3 Scaling of Linear Systems

In a linear system of equations $Ax = b$ the ith equation may be multiplied by an arbitrary positive scale factor d_i, $i = 1 : n$, without changing the exact solution. In contrast, such a scaling will usually change the computed numerical solution. In this section we show that *a proper row scaling is important for GE with partial pivoting to give accurate computed solutions*, and give some rules for scaling.

We first show that *if the pivot sequence is fixed* then Gaussian elimination is unaffected by such scalings, or more precisely:

Theorem 7.6.9.

Denote by $\bar{x}$ and $\bar{x}'$ the computed solutions obtained by GE in floating point arithmetic to the two linear systems of equations

$$Ax = b, \qquad (D_r A D_c)x' = D_r b,$$

where D_r and D_c are diagonal scaling matrices. Assume that the elements of D_r and D_c are powers of the base of the number system used, so that no rounding errors are introduced by the scaling. Then if the same pivot sequence is used and no overflow or underflow occurs we have exactly $\bar{x} = D_c\bar{x}'$, i.e., the components in the solution differ only in the exponents.

Proof. The proof follows by examination of the scaling invariance of the basic step in Algorithm 7.2.2

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - (a_{ik}^{(k)} a_{kj}^{(k)})/a_{kk}^{(k)}.$$

□

This result has the important implication that scaling will affect the accuracy of a computed solution only if it leads to a change in the selection of pivots. When partial pivoting is used the row scaling may affect the choice of pivots; indeed we can always find a row scaling which leads to *any predetermined pivot sequence*. However, since only elements in the pivotal column are compared, the choice of pivots is independent of the column scaling. Since a bad choice of pivots can give rise to large errors in the computed solution, it follows that for GE with partial pivoting to give accurate solutions *a proper row scaling is important*.

Example 7.6.1. The system $Ax = b$ in Example 7.5.1 has the solution $x = (0.9999, 0.9999)^T$, correctly rounded to four decimals. Partial pivoting will here select the element a_{11} as pivot. Using three-figure floating point arithmetic, the computed solution becomes

$$\bar{x} = (0, 1.00)^T \qquad \text{(Bad!)}.$$

If GE instead is carried out on the scaled system $\hat{A}x = \hat{b}$, then a_{21} will be chosen as pivot, and the computed solution becomes

$$\bar{x} = (1.00, 1.00)^T \qquad \text{(Good!)}.$$

From the above discussion we conclude that the need for a proper scaling is of great importance for GE to yield good accuracy. An discussed in Sec, 7.5.3, an estimate of $\kappa(A)$ is often used to access the accuracy of the computed solution. If, e.g., the perturbation bound (7.1.64) is applied to the scaled system $(D_r A D_c)x' = D_r b$

$$\frac{\|D_c^{-1}\delta x\|}{\|D_c^{-1}x\|} \leq \kappa(D_r A D_c)\frac{\|D_r \delta b\|}{\|D_r b\|}. \tag{7.6.24}$$

Hence if $\kappa(D_r A D_c)$ can be made smaller than $\kappa(A)$, then it seems that we might expect a correspondingly more accurate solution. Note however that in (7.6.24) the perturbation in x is measured in the norm $\|D_c^{-1}x\|$, and we may only have found a norm in which the error *looks* better! We conclude that the column scaling D_c should be chosen in a way that reflects the importance of errors in the components of the solution. If $|x| \approx c$, and we want the same relative accuracy in all components we may take $D_c = diag(c)$.

We now discuss the choice of row scaling. A scheme which is sometimes advocated is to choose $D_r = \text{diag}\,(d_i)$ so that each row in $D_r A$ has the same l_1-norm, i.e.,

$$d_i = 1/\|a_i^T\|_1, \quad i = 1 : n. \tag{7.6.25}$$

(Sometimes the l_∞-norm, of the rows are instead made equal.) This scaling, called **row equilibration**, can be seen to avoid the bad pivot selection in Example 7.5.1. However, suppose that through an unfortunate choice of physical units the solution x has components of widely varying magnitude. Then, as shown by the following

example, row equilibration can lead to a *worse* computed solution than if no scaling is used!

Example 7.6.2. Consider the following system

$$A = \begin{pmatrix} 3\cdot 10^{-6} & 2 & 1 \\ 2 & 2 & 2 \\ 1 & 2 & -1 \end{pmatrix}, \qquad b = \begin{pmatrix} 3+3\cdot 10^{-6} \\ 6 \\ 2 \end{pmatrix} \qquad |\epsilon| \ll 1$$

which has the exact solution $x = (1,1,1)^T$. The matrix A is *well-conditioned*, $\kappa(A) \approx 3.52$, but the choice of a_{11} as pivot leads to a disastrous loss of accuracy. Assume that through an unfortunate choice of units, the system has been changed into

$$\hat{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2\cdot 10^{6} & 2 & 2 \\ 10^{6} & 2 & -1 \end{pmatrix},$$

with exact solution $\hat{x} = (10^{-6}, 1, 1)^T$. If now the rows are equilibrated, the system becomes

$$\tilde{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 2\cdot 10^{-6} & 2\cdot 10^{-6} \\ 1 & 2\cdot 10^{-6} & -10^{-6} \end{pmatrix}, \qquad \tilde{b} = \begin{pmatrix} 3+3\cdot 10^{-6} \\ 6\cdot 10^{-6} \\ 2\cdot 10^{-6} \end{pmatrix}.$$

GE with column pivoting will now choose a_{11} as pivot. Using floating point arithmetic with precision $u = 0.47\cdot 10^{-9}$ we get the computed solution of $\hat{A}x = \hat{b}$

$$\bar{x} = (0.999894122\cdot 10^{-6},\ 0.999983255,\ 1.000033489)^T.$$

This has only about four correct digits, so almost six digits have been lost!

A theoretical solution to the row scaling problem in GE with partial pivoting has been given by R. D. Skeel [56, 1979]. He shows a pivoting rule in GE should depend not only on the coefficient matrix but also on the solution. Hence separating the matrix factorization from the solution of the linear system may lead to instability. His scaling rule is based on minimizing a bound on the backward error that contains the quantity

$$\frac{\max_i(|D_r A||\bar{x}|)_i}{\min_i(|D_r A||\bar{x}|)_i}.$$

Scaling Rule: (R. D. Skeel) Assume that $\min_i(|A||x|)_i > 0$. Then scale the rows of A and b by $D_r = \operatorname{diag}(d_i)$, where

$$d_i = 1/(|A||x|)_i, \quad i = 1:n. \tag{7.6.26}$$

A measure of **ill-scaling** of the system $Ax = b$ is

$$\sigma(A,x) = \max_i(|A||x|)_i / \min_i(|A||x|)_i. \tag{7.6.27}$$

This scaling rule gives infinite scale factors for rows which satisfy $(|A||x|)_i = 0$. This may occur for sparse systems, i.e., when A (and possibly also x) has many zero components. In this case a large scale factor d_i should be chosen so that the corresponding row is selected as pivot row at the first opportunity.

Unfortunately scaling according to this rule is not in general practical, since it assumes that the solution x is at least approximately known. If the components of the solution vector x are known to be of the same magnitude then we can take $|x| = (1, \ldots, 1)^T$ in (7.6.26), which corresponds to row equilibration. Note that this assumption is violated in Example 7.6.2.

7.6.4 Iterative Refinement of Solutions

So far we have considered ways of *estimating* the accuracy of computed solutions. We now consider methods for *improving* the accuracy. Let $\bar{x}$ be any approximate solution to the linear system of equations $Ax = b$ and let $r = b - A\bar{x}$ be the corresponding residual vector. Then one can attempt to improve the solution by solving the system $A\delta = r$ for a correction δ and taking $x_c = \bar{x} + \delta$ as a new approximation. If no further rounding errors are performed in the computation of δ this is the exact solution. Otherwise this refinement process can be iterated. In floating-point arithmetic with base β this process of **iterative refinement** can be described as follows:

$$
\begin{aligned}
&s := 1; \quad x^{(s)} := \bar{x};\\
&\text{repeat}\\
&\qquad r^{(s)} := b - Ax^{(s)}; \qquad (\text{inprecision } u_2 = \beta^{-t_2})\\
&\qquad \text{solve } \; A\delta^{(s)} = r^{(s)}; \qquad (\text{inprecision } u_1 = \beta^{-t_1})\\
&\qquad x^{(s+1)} := x^{(s)} + \delta^{(s)};\\
&\qquad s := s + 1;\\
&\text{end}
\end{aligned}
$$

When $\bar{x}$ has been computed by GE this approach is attractive since we can use the computed factors $\bar{L}$ and $\bar{U}$ to solve for the corrections

$$\bar{L}(\bar{U}\delta^{(s)}) = r^{(s)}, \quad s = 1, 2, \ldots.$$

The computation of $r^{(s)}$ and $\delta^{(s)}$, therefore, only takes $n^2 + 2 \cdot \frac{1}{2}n^2 = 2n^2$ flops, which is an order of magnitude less than the $n^3/3$ flops required for the initial solution.

We note the possibility of using *extended precision* $t_2 > t_1$ for computing the residuals $r^{(s)}$; these are then rounded to single precision u_1 before solving for $\delta^{(s)}$. Since $x^{(s)}$, A and b are stored in single precision, only the accumulation of the inner product terms are in precision u_2, and no multiplications in extended precision occur. This is also called *mixed precision iterative refinement* as opposed to *fixed precision iterative refinement* when $t_2 = t_1$.

Since the product of two t digit floating point numbers can be exactly represented with at most $2t$ digits inner products can be computed in extended precision

without much extra cost. If fl_e denotes computation with extended precision and u_e the corresponding unit roundoff then the forward error bound for an inner product becomes

$$|fl\,(fl_e((x^Ty)) - x^Ty| < u|x^Ty| + \frac{nu_e}{1-nu_e/2}(1+u)|x^T||y|, \tag{7.6.28}$$

where the first term comes form the final rounding. If $|x^T||y| \leq u|x^Ty|$ then the computed inner product is almost as accurate as the correctly rounded exact result. However, since computations in extended precision are machine dependent it has been difficult to make such programs portable.[21] The recent development of Extended and Mixed Precision BLAS (Basic Linear Algebra Subroutines) (see [49]) may now make this more feasible!

In the ideal case that the rounding errors committed in computing the corrections can be neglected we have

$$x^{(s+1)} - x = (I - (\bar{L}\bar{U})^{-1}A)^s(\bar{x} - x).$$

where $\bar{L}$ and $\bar{U}$ denote the computed LU factors of A. Hence the process converges if

$$\rho = \|(I - (\bar{L}\bar{U})^{-1}A\| < 1.$$

This roughly describes how the refinement behaves in the *early stages*, if extended precision is used for the residuals. If $\bar{L}$ and $\bar{U}$ have been computed by GE using precision u_1, then by Theorem 7.2.3 we have

$$\bar{L}\bar{U} = A + E, \quad \|E\|_\infty \leq 1.5n^2\rho_n u_1\|A\|_\infty,$$

and ρ_n is the growth factor. It follows that an upper bound for the initial rate of convergence is given by

$$\rho = \|(\bar{L}\bar{U})^{-1}E\|_\infty \leq n^2\rho_n u_1\kappa(A).$$

When also rounding errors in computing the residuals $r^{(s)}$ and the corrections $\delta^{(s)}$ are taken into account, the analysis becomes much more complicated. The behavior of iterative refinement, using t_1-digits for the factorization and $t_2 = 2t_1$ digits when computing the residuals, can be summed up as follows:

1. Assume that A is not too ill-conditioned so that the first solution has some accuracy, $\|x - \bar{x}\|/\|x\| \approx \beta^{-k} < 1$ in some norm. Then the relative error diminishes by a factor of roughly β^{-k} with each step of refinement until we reach a stage at which $\|\delta_c\|/\|x_c\| < \beta^{-t_1}$, when we may say that the solution is correct to working precision.

2. In general the attainable accuracy is limited to $\min(k + t_2 - t_1, t_1)$ digits, which gives the case above when $t_2 \geq 2t_1$. Note that although the computed solution improves progressively with each iteration this is *not* reflected in a corresponding decrease in the norm of the residual, which stays about the same.

[21] It was suggested that the IEEE 754 standard should require inner products to be precisely specified, but that did not happen.

Iterative refinement can be used to compute a more accurate solution, in case A is ill-conditioned. However, unless A and b are exactly known this may not make much sense. The exact answer to a poorly conditioned problem may be no more appropriate than one which is correct to only a few places.

In many descriptions of iterative refinement it is stressed that it is essential that the residuals are computed with a higher precision than the rest of the computation, for the process to yield a more accurate solution. This is true if the initial solution has been computed by a backward stable method, such as GE with partial pivoting, and provided that the system is well scaled. However, iterative refinement using single precision residuals, *can considerably improve the quality of the solution, for example, when the system is ill-scaled*, i.e., when $\sigma(A, x)$ defined by (7.6.27) is large, or if the pivot strategy has been chosen for the preservation of sparsity, see Section 7.6.

Example 7.6.3. As an illustration consider again the badly scaled system in Example 7.6.1

$$\tilde{A} = \begin{pmatrix} 3 & 2 & 1 \\ 2 & 2\cdot 10^{-6} & 2\cdot 10^{-6} \\ 1 & 2\cdot 10^{-6} & -10^{-6} \end{pmatrix}, \qquad \tilde{b} = \begin{pmatrix} 3+3\cdot 10^{-6} \\ 6\cdot 10^{-6} \\ 2\cdot 10^{-6} \end{pmatrix},$$

with exact solution $\tilde{x} = (10^{-6}, 1, 1)^T$. Using floating point arithmetic with unit roundoff $u = 0.47 \cdot 10^{-9}$ the solution computed by GE with partial pivoting has only about four correct digits. From the residual $r = \tilde{b} - \tilde{A}\bar{x}$ we compute the Oettli–Prager backward error $\omega = 0.28810 \cdot 10^{-4}$. The condition estimate computed by (7.5.17) is $3.00 \cdot 10^{6}$, and wrongly indicates that the loss of accuracy should be blamed on ill-conditioning.

With one step of iterative refinement using a single precision residual we get

$$\tilde{x} = \bar{x} + d = (\, 0.999999997\cdot 10^{-6} \quad 1.000000000 \quad 1.000000000\,)^T.$$

This is almost as good as for GE with column pivoting applied to the system $Ax = b$. The Oettli–Prager error bound for $\tilde{x}$ is $\omega = 0.54328 \cdot 10^{-9}$, which is close to the machine precision. Hence one step of iterative refinement sufficed to correct for the bad scaling. If the ill-scaling is worse or the system is also ill-conditioned then several steps of refinement may be needed.

The following theorem states that if GE with partial pivoting is combined with iterative refinement in single precision then the resulting method will give a small relative backward error provided that the system is not too ill-conditioned or ill-scaled.

Theorem 7.6.10. (R. D. Skeel.)

As long as the product of $\operatorname{cond}(A^{-1}) = \||A||A^{-1}|\|_\infty$ *and* $\sigma(A, x)$ *is sufficiently less than* $1/u$*, where* u *is the machine unit, it holds that*

$$(A + \delta A)x^{(s)} = b + \delta b, \quad |\delta a_{ij}| < 4n\epsilon_1|a_{ij}|, \quad |\delta b_i| < 4n\epsilon_1|b_i|, \tag{7.6.29}$$

for s large enough. Moreover, the result is often true already for $s = 2$, i.e., after only one improvement.

Proof. For exact conditions under which this theorem holds, see Skeel [57, 1980]. □

As illustrated above, GE with partial or complete pivoting may not provide all the accuracy that the data deserves. How often this happens in practice is not known. In cases where accuracy is important the following scheme, which offers improved reliability for a small cost is recommended.

1. Compute the Oettli–Prager backward error ω using (7.5.10) with $E = |A|$, $f = |b|$, by simultaneously accumulating $r = b - A\bar{x}$ and $|A||\bar{x}| + |b|$. If ω is not sufficiently small go to 2.

2. Perform one step of iterative refinement using the single precision residual r computed in step 1 to obtain the improved solution $\tilde{x}$. Compute the backward error $\tilde{\omega}$ of $\tilde{x}$. Repeat until the test on $\tilde{\omega}$ is passed.

7.6.5 Interval Matrix Computations

In order to treat multidimensional problems interval vectors $[x] = ([x_i])$ with interval components $[x_i] = [\underline{x_i}, \overline{x_i}])$, $i = 1 : n$ and interval matrices $[A] = ([a_{ij}])$ with interval elements $[a_{ij}] = [\underline{a}_{ij}, \overline{a}_{ij}]$, $i = 1 : m$, $j = 1 : n$, are introduced.

Operations between interval matrices and interval vectors are defined in an obvious manner. The interval matrix-vector product $[A][x]$ is the smallest interval vector, which contains the set $\{Ax \mid A \in [A],\ x \in [x]\}$, but normally does not coincide with it. By the inclusion property it holds that

$$\{Ax \mid A \in [A],\ x \in [x]\} \subseteq [A][x] = \left(\sum_{j=1}^{n} [a_{ij}][x_j] \right). \tag{7.6.30}$$

In general there will be an overestimation in enclosing the image with an interval vector caused by the fact that the image of an interval vector under a transformation in general is not an interval vector. This phenomenon, intrinsic to interval computations, is called the **wrapping effect**.

Example 7.6.4.

We have

$$A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad [x] = \begin{pmatrix} [0,1] \\ [0,1] \end{pmatrix}, \quad \Rightarrow \quad A[x] = \begin{pmatrix} [0,2] \\ [-1,1] \end{pmatrix}.$$

Hence $b = (\,2 \quad -1\,)^T \in A[x]$, but there is no $x \in [x]$ such that $Ax = b$. (The solution to $Ax = b$ is $x = (\,3/2 \quad 1/2\,)^T$.)

The magnitude of an interval vector or matrix is interpreted component-wise and defined by

$$|\,[x]\,| = (|\,[x_1]\,|, |\,[x_2]\,|, \ldots, |\,[x_n]\,|)^T,$$

where the magnitude of the components are defined by

$$|\,[a,b]\,| = \max\{|x| \mid x \in [a,b]\}, \tag{7.6.31}$$

The ∞-norm of an interval vector or matrix is defined as the ∞-norm of their magnitude,

$$\|\,[x]\,\|_\infty = \|\,|\,[x]\,|\,\|_\infty, \qquad \|\,[A]\,\|_\infty = \|\,|\,[A]\,|\,\|_\infty. \tag{7.6.32}$$

In implementing matrix multiplication it is important to avoid case distinctions in the inner loops, because that would make it impossible to use fast vector and matrix operations. Using interval arithmetic it is possible to compute strict enclosures of the product of two matrices. Note that this is needed also in the case of the product of two point matrices since rounding errors will in general occur. In this case we want to compute an interval matrix $[C]$ such that $fl(A \cdot B) \subset [C] = [C_{\inf}, C_{\sup}]$. The following simple code does that using two matrix multiplications:

$$\begin{aligned} &\text{setround}(-1); \quad C_{\inf} = A \cdot B; \\ &\text{setround}(1); \quad\;\; C_{\sup} = A \cdot B; \end{aligned}$$

Here and in the following we assume that the command setround(i), $i = -1, 0, 1$ sets the rounding mode to $-\infty$, to nearest, and to $+\infty$, respectively.

We next consider the product of a point matrix A and an interval matrix $[B] = [B_{\inf}, B_{\sup}]$. The following code, suggested by A. Neumeier, performs this using four matrix multiplications:

$$\begin{aligned} &A_- = \min(A,\ 0); \quad A_+ = \max(A,\ 0); \\ &\text{setround}(-1); \\ &C_{\inf} = A_+ \cdot B_{\inf} + A_- \cdot B_{\sup}; \\ &\text{setround}(1); \\ &C_{\sup} = A_- \cdot B_{\inf} + A_+ \cdot B_{\sup}; \end{aligned}$$

(Note that the commands $A_- = \min(A,\ 0)$ and $A_+ = \max(A,\ 0)$ acts componentwise.) Rump [55] gives an algorithm for computing the product of two interval matrices using eight matrix multiplications. He also gives several faster implementations, provided a certain overestimation can be allowed.

A square interval matrix $[A]$ is called nonsingular if it does not contain a singular matrix. An interval linear system is a system of the form $[A]\,x = [b]$, where A is a nonsingular interval matrix and b an interval vector. The solution set of such an interval linear system is the set

$$\mathcal{X} = \{x \mid Ax = b,\ A \in [A],\ b \in [b]\}. \tag{7.6.33}$$

Computing this solution set can be shown to be an intractable problem (NP-complete). Even for a 2×2 linear system this set may not be easy to represent.

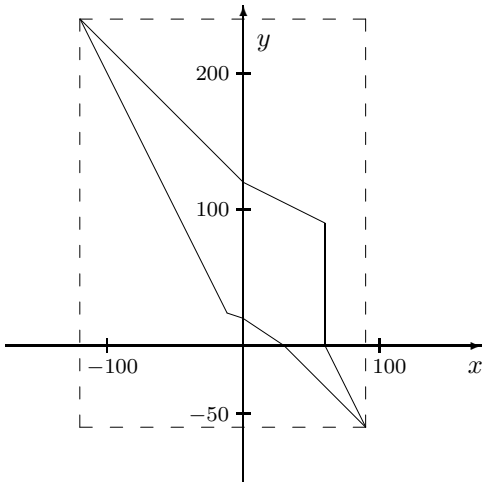


Figure 7.6.1. *The solution set (solid line) and its enclosing hull (dashed line) for the linear system in Example* 7.6.5.

Example 7.6.5. (Hansen [37, Chapter 4])
Consider a linear system with

$$[A] = \begin{pmatrix} [2,3] & [0,1] \\ [1,2] & [2,3] \end{pmatrix}, \quad [b] = \begin{pmatrix} [0,120] \\ [60,240] \end{pmatrix}. \tag{7.6.34}$$

The solution set $\mathcal{X}$ in (7.6.33) is the star shaped region in Fig. 7.6.1.

An enclosure of the solution set of an interval linear system can be computed by a generalization of Gaussian elimination adopted to interval coefficients. The solution of the resulting interval triangular system will give an inclusion of the solution set. Realistic bounds can be obtained in this way only for special classes of matrices, e.g., for diagonally dominant matrices and tridiagonal matrices; see Hargreaves [38]. For general systems this approach unfortunately tends to give interval sizes which grow exponentially during the elimination. For example, if $[x]$ and $[y]$ are intervals then in the 2×2 reduction

$$\begin{pmatrix} 1 & [x] \\ 1 & [y] \end{pmatrix} \sim \begin{pmatrix} 1 & [x] \\ 0 & [y]-[x] \end{pmatrix}.$$

If $[x] \approx [y]$ the size of the interval $[y]-[x]$ will be twice the size of $[x]$. This growth is very likely to happen. Even for well-conditioned linear systems the elimination can break down prematurely, because all remaining possible pivot elements contain zero.

A better way to compute verified bounds on a point or interval linear system uses an idea that goes back to E. Hansen [1965]. In this an approximate inverse C is used to precondition the system. Assuming that an initial interval vector $[x^{(0)}]$

is known, such that $[x^{(0)}] \supseteq \mathcal{X}$ where $\mathcal{X}$ is the solution set (7.6.33). An improved enclosure can then be obtained as follows:

By the inclusion property of interval arithmetic, for all $\tilde{A} \in [A]$ and $\tilde{b} \in [b]$ it holds that

$$\tilde{A}^{-1}\tilde{b} = C\tilde{b} + (I - C\tilde{A})\tilde{A}^{-1}\tilde{b} \in C\,[b] + (I - C\,[A])[x^{(0)}] =: [x^{(1)}].$$

This suggests the iteration known as **Krawczyck's method**

$$[x^{(i+1)}] = \Big(C\,[b] + (I - C[A])[x^{(i)}]\Big) \cap [x^{(i)}], \quad i = 0, 1, 2, \ldots, \tag{7.6.35}$$

for computing a sequence of interval enclosures $[x^{(i)}]$ of the solution. Here the interval vector $[c] = C\,[b]$ and interval matrix $[E] = I - C\,[A]$ need only be computed once. The dominating cost per iteration is one interval matrix-vector multiplication.

As approximate inverse we can take the inverse of the midpoint matrix $C = (\mathrm{mid}\,[A])^{-1}$. An initial interval can be chosen of the form

$$[x^{(0)}] = C\mathrm{mid}\,[b] + [-\beta, \beta]e, \quad e = (1, 1, \ldots, 1),$$

with β sufficiently large. The iterations are terminated when the bounds are no longer improving. A measure of convergence can be computed as $\rho = \|[E]\|_\infty$.

Rump [55, 54] has developed a MATLAB toolbox INTLAB (INTerval LABoratory). This is very efficient and easy to use and includes many useful subroutines. INTLAB uses a variant of Krawczyck's method, applied to a residual system, to compute an enclosure of the difference between the solution and an approximate solution $x_m = C\mathrm{mid}\,[b]$; see Rump [55].

Example 7.6.6.

A method for computing an enclosure of the inverse of an interval matrix can be obtained by taking $[b]$ equal to the identity matrix in the iteration (7.6.35) and solving the system $[A][X] = I$. For the symmetric interval matrix

$$[A] = \begin{pmatrix} [0.999, 1.01] & [-0.001, 0.001] \\ [-0.001, 0.001] & [0.999, 1.01] \end{pmatrix}$$

the identity $C = \mathrm{mid}\,[A] = I$ is an approximate point inverse. We find

$$[E] = I - C[A] = \begin{pmatrix} [-0.01, 0.001] & [-0.001, 0.001] \\ [-0.001, 0.001] & [-0.01, 1.001] \end{pmatrix},$$

and as an enclosure for the inverse matrix we can take

$$[X^{(0)}] = \begin{pmatrix} [0.98, 1.02] & [-0.002, 0.002] \\ [-0.002, 0.002] & [0.98, 1.02] \end{pmatrix}.$$

The iteration

$$[X^{(i+1)}] = \Big(I + E[X^{(i)}]\Big) \cap [X^{(i)}], \quad i = 0, 1, 2, \ldots.$$

converges rapidly in this case.

Review Questions

1. The result of a roundoff error analysis of Gaussian elimination can be stated in the form of a backward error analysis. Formulate this result. (You don't need to know the precise expression of the constants involved.)

2. (a) Describe the main steps in iterative refinement with extended precision for computing more accurate solutions of linear system.

(b) Sometimes it is worthwhile to do a step of iterative refinement in using fixed precision. When is that?

Problems

1. Compute the LU factors of the matrix in (7.6.12).

7.7 Block Algorithms for Gaussian Elimination

7.7.1 Block and Blocked Algorithms

In block matrix algorithms most part of the arithmetic operations consists of matrix multiplications. Because of this these algorithms can achieve high performance on modern computers. The following distinction between two different classes of algorithms is important to emphasize, since they have different stability properties.

As a first example, consider the inverse of a block lower triangular matrix

$$L = \begin{pmatrix} L_{11} & & & \\ L_{21} & L_{22} & & \\ \vdots & \vdots & \ddots & \\ L_{n,1} & L_{n,2} & \cdots & L_{nn} \end{pmatrix}, \tag{7.7.1}$$

If the diagonal blocks L_{ii}, $i = 1:2$, are nonsingular, it is easily verified that the inverse also will be block lower triangular,

$$L^{-1} = \begin{pmatrix} Y_{11} & & & \\ Y_{21} & Y_{22} & & \\ \vdots & \vdots & \ddots & \\ Y_{n,1} & Y_{n,2} & \cdots & Y_{nn} \end{pmatrix}, \tag{7.7.2}$$

In Sec. 7.1.5 we showed that the inverse in the 2×2 case is

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} L_{21} L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}.$$

Note that we do not assume that the diagonal blocks are lower triangular.

In the general case the blocks in the inverse can be computed a block column at a time from a straightforward extension of the scalar algorithm (7.2.40). Identifying blocks in the jth block column, of the equation $LY = I$, we for $j = 1 : n$,

$$L_{jj}Y_{jj} = I, \qquad L_{ii}Y_{ij} = -\sum_{k=j}^{i-1} L_{ik}Y_{kj}, \quad i = j+1 : n. \tag{7.7.3}$$

These equations can be solved for $Y_{jj}, \ldots, Y_{nj}$, by the scalar algorithms described in Sec. 7.2. The main arithmetic work will take place in the matrix-matrix multiplications $L_{ik}Y_{kj}$. This is an example of a true **block algorithm**, which is a obtained by substituting in a scalar algorithm operations on blocks of partitioned matrices regarded as non-commuting scalars.

In the special case that L is a lower triangular matrix this implies that all diagonal blocks L_{ii} and Y_{ii}, $i = 1 : n$, are lower triangular. In this case the equations in (7.7.3) can be solved by back-substitution. The resulting algorithm is then just a scalar algorithm in which the operations have been grouped and reordered into matrix operations. Such an algorithm is called a **blocked algorithm**. Blocked algorithms have the same stability properties as their scalar counterparts. This is not true for general block algorithms, which is why the distinction is important to make.

In Sec. 7.1.5 we gave, using slightly different notations, the block LU factorization

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ 0 & S \end{pmatrix}, \tag{7.7.4}$$

for a 2×2 block matrix, with square diagonal blocks. Here $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement. Note that the diagonal blocks in the block lower triangular factor in (7.7.4) are the identity matrix. Hence, this is a true block algorithm.

In a blocked LU factorization algorithm, the LU factors should have the form

$$A = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix},$$

where L_{11}, L_{22} are unit lower triangular and U_{11} , U_{22} are upper triangular. Such a factorization can be computes as follows. We first compute the scalar LU factorization $A_{11} = L_{11}U_{11}$, and then compute

$$L_{21} = A_{21}U_{11}^{-1}, \qquad U_{12} = L_{11}^{-1}A_{12}, \qquad S_{22} = A_{22} - L_{21}U_{12}.$$

Finally compute the scalar factorization $S_{22} = L_{22}U_{22}$.

In the general case a blocked algorithm for the LU factorization of a block matrix

$$A = \begin{pmatrix} A_{11} & A_{12} & \ldots & A_{1N} \\ A_{21} & A_{22} & \ldots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \ldots & A_{NN} \end{pmatrix}, \tag{7.7.5}$$

having square diagonal blocks. Let L and U be partitioned conformally with A. Equating blocks in the product $A = LU$, we obtain, assuming that all inverses exist, the following block LU algorithm:

Algorithm 7.7.1 Blocked LU Factorization.

for $k = 1 : N$

$$S_{kk} = A_{kk} - \sum_{p=1}^{k-1} L_{kp}U_{pk};$$

$$S_{kk} = L_{kk}U_{kk}$$

for $j = k + 1 : N$

$$L_{jk} = \Big(A_{jk} - \sum_{p=1}^{k-1} L_{jp}U_{pk}\Big)U_{kk}^{-1};$$

end

for $j = 1 : k - 1$

$$U_{jk} = L_{kk}^{-1}\Big(A_{jk} - \sum_{p=1}^{k-1} L_{jp}U_{pj}\Big);$$

end

end

Here the LU-decompositions $S_{kk} = L_{kk}U_{kK}$ of the modified diagonal blocks are computed by a scalar LU factorization algorithm. However, the dominating part of the work is performed in matrix-matrix multiplications. The inverse of the triangular matrices L_{kk}^{-1} and U_{kk}^{-1} are *not* formed but the off-diagonal blocks U_{kj} and L_{jk} (which in general are full matrices) are computed by triangular solves. Pivoting can be used in the factorization of the diagonal blocks. As described the algorithm does not allow for row interchanges between blocks. This point is addressed In the next section.

As with the scalar algorithms there are many possible ways of sequencing the block factorization. The block algorithm above computes in the kth major step the kth block column of L and U. In this variant at step k only the kth block column of A is accessed, which is advantageous from the standpoint of data access.

A block LU factorization algorithm differs from the blocked algorithm above in that the lower block triangular matrix L has diagonal blocks equal to unity. Although such a block algorithm may have good numerical stability properties this cannot be taken for granted, since in general they do not perform the same arithmetic operations as in the corresponding scalar algorithms. It has been shown that block LU factorization can fail even for symmetric positive definite and row diagonally dominant matrices.

One class of matrices for which the block LU algorithm is known to be stable is block tridiagonal matrices that are **block diagonally dominant**. .

Definition 7.7.1. *(see Demmel et al. [16])*

A general matrix $A \in \mathbf{R}^{n\times n}$ is said to be block diagonally dominant by columns, with respect to a given partitioning, if it holds i.e.

$$\|A_{jj}^{-1}\|^{-1} \geq \sum_{i\neq j} \|A_{ij}\|, \quad j = 1 : n. \tag{7.7.6}$$

A is block **diagonally dominant** *by rows, if A^T is (strictly) diagonally dominant by columns.*

Note that for block size 1 the usual property of (point) diagonal dominance is obtained. For the 1 and ∞-norms diagonal dominance does not imply block diagonal dominance Neither does and the reverse implications hold.

Analogous to the Block LU Algorithm in Section 7.7.1 block versions of the Cholesky algorithm can be developed. If we assume that A has been partitioned into $N \times N$ blocks with square diagonal blocks we get using a block column-wise order:

Algorithm 7.7.2 Blocked Cholesky Algorithm.

$$
\begin{aligned}
&\textbf{for } j = 1 : N \\
&\quad S_{jj} = A_{jj} - \sum_{k=1}^{j-1} R_{jk}^T R_{jk}; \\
&\quad S_{jj} = R_{jj}^T R_{jj} \\
&\quad \textbf{for } i = j+1 : N \\
&\qquad R_{ij}^T = \Big(A_{ij} - \sum_{k=1}^{j-1} R_{ik}^T R_{jk} \Big) (R_{jj})^{-1}; \\
&\quad \textbf{end} \\
&\textbf{end}
\end{aligned}
$$

Note that the diagonal blocks R_{jj} are obtained by computing the Cholesky factorizations of matrices of smaller dimensions. The right multiplication with $(R_{jj})^{-1}$ in the computation of R_{jk}^T is performed by solving the triangular equations of the form $R_{jj}^T R_{ij} = S^T$. The matrix multiplications dominate the arithmetic work in the block Cholesky algorithm.

In deriving the block LU and Cholesky algorithms we assumed that the block sizes were determined in advance. However, this is by no means necessary. A more flexible way is to advance the computation by deciding at each step the size of the current pivot block. The corresponding blocked formulation then uses a 3×3 block structure, but the partitioning changes after each step.

Suppose that a partial LU factorization so that

$$
P_1 A = \begin{pmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & \tilde{A}_{22} & \tilde{A}_{23} \\ & \tilde{A}_{32} & \tilde{A}_{33} \end{pmatrix}.
$$

has been obtained, where P_1 is a permutation matrix and L_{11}, $U_{11} \in \mathbf{R}^{n_1 \times n_1}$. To advance the factorization compute the LU factorization with row pivoting

$$
P_2 \begin{pmatrix} \tilde{A}_{22} \\ \tilde{A}_{32} \end{pmatrix} = \begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22},
$$

where L_{22}, $U_{22} \in \mathbf{R}^{n_2 \times n_2}$. The permutation matrix P_2 has to be applied also to

$$\begin{pmatrix} \tilde{A}_{23} \\ \tilde{A}_{33} \end{pmatrix} := P_2 \begin{pmatrix} \tilde{A}_{23} \\ \tilde{A}_{33} \end{pmatrix}, \qquad \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} := P_2 \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix}.$$

We then solve for U_{23} and update A_{33} using

$$L_{22}U_{23} = \tilde{A}_{23}, \qquad \tilde{A}_{33} = A_{33} - L_{32}U_{23}.$$

The factorization has now been advanced one step to become

$$P_2PA = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & A_{33} \end{pmatrix}.$$

We can now repartition so that the first two block-columns in L are joined into a block of $n_1 + n_2$ columns and similarly the first two block-rows in U joined into one block of $n_1 + n_2$ rows. The blocks I and A_{33} in L and U are partitioned into 2×2 block matrices and we advance to the next block-step. This describes the complete algorithm since we can start the algorithm by taking $n_1 = 0$.

The above algorithm is sometimes called **right-looking**, referring to the way in which the data is accessed. The corresponding **left-looking** algorithm goes as follows. Assume that we have computed

$$PA = \begin{pmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} U_{11} & A_{12} & A_{13} \\ & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{pmatrix}.$$

To advance the factorization we solve first a triangular system $L_{11}U_{12} = A_{12}$ to obtain U_{12} and then compute

$$\begin{pmatrix} \tilde{A}_{22} \\ \tilde{A}_{32} \end{pmatrix} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{12},$$

We then compute the LU factorization with row pivoting

$$P_2 \begin{pmatrix} \tilde{A}_{22} \\ \tilde{A}_{32} \end{pmatrix} = \begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} U_{22},$$

and replace

$$\begin{pmatrix} \tilde{A}_{23} \\ \tilde{A}_{33} \end{pmatrix} = P_2 \begin{pmatrix} A_{23} \\ A_{33} \end{pmatrix}, \qquad \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} := P_2 \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix}.$$

The factorization has now been advanced one step to become

$$P_2PA = \begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & A_{13} \\ & U_{22} & A_{23} \\ & & A_{33} \end{pmatrix}.$$

Note that in this version the blocks in the last block column of A are referenced only in the pivoting operation, but this can be postponed.

Block LU factorizations appears to have been first proposed for **block tridiagonal** matrices, which often arise from the discretization of partial differential equations. For a symmetric positive definite matrix the recursion (7.4.11) is easily generalized to compute the following block-factorization:

$$A = U^T D^{-1} U, \quad D = \operatorname{diag}(\Sigma_1, \ldots, \Sigma_n),$$

of a symmetric positive definite **block-tridiagonal** matrix with square diagonal blocks. We obtain

$$A = \begin{pmatrix} D_1 & A_2^T & & & \\ A_2 & D_2 & A_3^T & & \\ & A_3 & \ddots & \ddots & \\ & & \ddots & \ddots & A_N^T \\ & & & A_N & D_N \end{pmatrix}, \qquad U^T = \begin{pmatrix} \Sigma_1 & & & & \\ A_2 & \Sigma_2 & & & \\ & A_3 & \ddots & & \\ & & \ddots & \ddots & \\ & & & A_N & \Sigma_N \end{pmatrix},$$

where

$$\Sigma_1 = D_1, \qquad \Sigma_k = D_k - A_k \Sigma_{k-1}^{-1} A_k^T, \quad k = 2 : N. \tag{7.7.7}$$

To perform the operations with Σ_k^{-1}, $k = 1 : N$ the Cholesky factorization of these matrices are computed by a scalar algorithm. After this factorization has been computed the solution of the system

$$Ax = U^T D^{-1} U x = b$$

can be obtained by block forward- and back-substitution $U^T z = b$, $Ux = Dz$.

Note that the blocks of the matrix A may again have band-structure, which should be taken advantage of! A similar algorithm can be developed for the unsymmetric block-tridiagonal case.

For block tridiagonal matrices the following result is known:

Theorem 7.7.2. *(*Varah [64]*)*

Let the matrix $A \in \mathbf{R}^{n\times n}$ be block tridiagonal and have the block LU factorization $A = LU$, where L and U are block bidiagonal, and normalized so that $U_{i,i+1} = A_{i,i+1}$. Then if A is block diagonally dominant by columns

$$\|L_{i,i-1}\| \le 1, \qquad \|U_{i,i}\| \le \|A_{i,i}\| + \|A_{i-1,i}\|. \tag{7.7.8}$$

If A is block diagonally dominant by rows

$$\|L_{i,i-1}\| \le \frac{\|A_{i-1,i}\|}{\|A_{i,i-1}\|}, \qquad \|U_{i,i}\| \le \|A_{i,i}\| + \|A_{i-1,i}\|. \tag{7.7.9}$$

These results can be extended to full block diagonally dominant matrices, by using the key property that block diagonal dominance is inherited by the Schur complements obtained in the factorizations.

7.7.2 Recursive Algorithms

In the 2×2 case the block LU factorization Algorithm 7.7.1 is obtained by equating

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}. \tag{7.7.10}$$

We get

$$\begin{aligned} &L_{11}U_{11} = A_{11}, \\ &L_{21} = A_{21}U_{11}^{-1}, \qquad U_{12} = L_{11}^{-1}A_{12}, \\ &\tilde{A}_{22} = A_{22} - L_{21}U_{12}, \\ &L_{22}U_{22} = \tilde{A}_{22}. \end{aligned}$$

Hence the LU factorization of A can be reduced to the LU factorization of two smaller matrices A_{11} and $\tilde{A}_{22}$, two triangular solves with matrix right hand sides, and one matrix update $A_{22} - L_{21}U_{12}$.

Similarly for the Cholesky factorization equating

$$\begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{pmatrix}, \tag{7.7.11}$$

gives

$$\begin{aligned} &L_{11}L_{11}^T = A_{11}, \\ &L_{21}^T = L_{11}^{-1}A_{21}^T, \\ &L_{22}L_{22}^T = A_{22} - L_{21}L_{21}^T. \end{aligned}$$

It is possible to derive "divide and conquer" algorithms for the LU and Cholesky algorithms, by using the 2×2 block versions recursively. see [24].

Algorithm 7.7.3 Recursive Cholesky Factorization.

Let $A \in \mathbf{R}^{n \times n}$ be a symmetric positive definite matrix. The following recursive algorithm computes the Cholesky factorization of A.

```
function L = rchol(A);
[n, n] = size(A);
if n ≠ 1
%RecursiveCholesky
   k = floor(n/2)
   L11 = rchol(A(1 : k, 1 : k));
   L21 = (L11^{-1} A(1 : k, k + 1 : n))';
   L22 = rchol(A(1 : k, 1 : k) − L21 ∗ L21');
   L = [L11 zeros(k, n − k); L21 L22];
```

else
$L = sqrt(A)$;
end;

This is not a toy algorithm, but can be developed into an efficient algorithm for parallel high performance computers!

An intriguing question is whether it is possible to multiply two matrices $A \in \mathbf{R}^{m\times n}$, and $B \in \mathbf{R}^{n\times p}$ in less than mnp (scalar) multiplications. The answer is yes! Strassen [1969] developed a fast algorithm for matrix multiplication based on the following method for multiplying 2×2 block matrices. Assume that m, n, p are even and partition each of A, B and the product $C \in \mathbf{R}^{m\times p}$, into four equally sized blocks. Then, as can be verified by substitution, the product $C = AB$ can be computed using the following formulas:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 + P_3 - P_2 + P_6 \end{pmatrix},$$

where

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \qquad P_2 = (A_{21} + A_{22})B_{11}, \\ P_3 &= A_{11}(B_{12} - B_{22}), \qquad P_4 = A_{22}(B_{21} - B_{11}), \\ P_5 &= (A_{11} + A_{12})B_{22}, \qquad P_6 = (A_{21} - A_{11})(B_{11} + B_{12}), \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}). \end{aligned}$$

The key property of **Strassen's algorithm** is that only *seven* matrix multiplications and eighteen matrix additions are needed, instead of the *eight* matrix multiplications and four matrix additions required using conventional block matrix multiplications. Since for large dimensions multiplication of two matrices is more expensive (n^3) than addition (n^2) this will lead to a saving in operations.

Strassen's algorithm can be used recursively. to multiply two square matrices of dimension $n = 2^k$. The number of multiplications is then reduced from n^3 to $n^{\log_2 7} = n^{2.807\cdots}$. (The number of additions is of the same order.) Even with just one level of recursion Strassen's method is faster in practice when n is larger than about 100, see Problem 2. However, there is some loss of numerical stability compared to conventional matrix multiplication, see Higham [41, Ch. 23].

By using the block formulation recursively, and Strassen's method for the matrix multiplication it is possible to perform the LU factorization in $O(n^{\log_2 7})$ operations.

7.7.3 Kronecker Systems

Linear systems where the matrix is a **Kronecker product**[22] arise in several application areas such as signal and image processing, photogrammetry, multidimensional data fitting, etc. Such systems can be solved with great savings in storage

[22] Leopold Kronecker (1823–1891) German mathematician. He is known also for his remark "God created the integers, all else is the work of man".

and operations. Since often the size of the matrices A and B is large, resulting in models involving several hundred thousand equations and unknowns, such savings may be essential.

Definition 7.7.3.
Let $A \in \mathbf{C}^{m\times n}$ and $B \in \mathbf{C}^{p\times q}$ be two matrices. Then the **Kronecker product** *of A and B is the $mp \times nq$ block matrix*

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}. \tag{7.7.12}$$

We now state without proofs some elementary facts about Kronecker products. From the definition (7.7.12) it follows that

$$\begin{aligned} (A+B)\otimes C &= (A\otimes C)+(B\otimes C),\\ A\otimes(B+C) &= (A\otimes B)+(A\otimes C),\\ A\otimes(B\otimes C) &= (A\otimes B)\otimes C,\\ (A\otimes B)^T &= A^T\otimes B^T. \end{aligned}$$

Further we have the important mixed-product relation, which is not so obvious:

Lemma 7.7.4.
Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{p\times q}$, $C \in \mathbf{R}^{n\times k}$, and $D \in \mathbf{R}^{q\times r}$. Then the ordinary matrix products AC and BD are defined, and

$$(A\otimes B)(C\otimes D) = AC\otimes BD. \tag{7.7.13}$$

Proof. Let $A = (a_{ik})$ and $C = (c_{kj})$. Partitioning according to the sizes of B and D, $A\otimes B = (a_{ik}B)$ and $C\otimes D = (c_{kj}D)$. Hence, the (i,j)th block of $(A\otimes B)(C\otimes D)$ equals

$$\sum_{k=1}^{n} a_{ik}Bc_{kj}D = \Big(\sum_{k=1}^{n} a_{ik}c_{kj}\Big)BD,$$

which is the (i,j)th element of AC times BD, which is the (i,j)th block of $(A\otimes B)(C\otimes D)$. ∎

If $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{p\times p}$ are non-singular, then then by Lemma 7.7.4

$$(A^{-1}\otimes B^{-1})(A\otimes B) = I_n \otimes I_p = I_{np}.$$

It follows that $A\otimes B$ is nonsingular and

$$(A\otimes B)^{-1} = A^{-1}\otimes B^{-1}. \tag{7.7.14}$$

We now introduce an operator closely related to the Kronecker product, which converts a matrix into a vector.

Definition 7.7.5. *Given a matrix* $C = (c_1, c_2, \ldots, c_n) \in \mathbf{R}^{m\times n}$ *we define*

$$\operatorname{vec}(C) = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}, \tag{7.7.15}$$

that is, the vector formed by **stacking** *the columns of* C *into one long vector.*

We now state an important result which shows how the vec-function is related to the Kronecker product.

Lemma 7.7.6.
If $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{p\times q}$, *and* $C \in \mathbf{R}^{q\times n}$, *then*

$$(A \otimes B)\operatorname{vec} C = \operatorname{vec} X, \quad X = BCA^T. \tag{7.7.16}$$

Proof. Denote the kth column of a matrix M by M_k. Then

$$\begin{aligned}(BCA^T)_k = BC(A^T)_k &= B\sum_{i=1}^{n} a_{ki}C_i \\ &= (\, a_{k1}B \quad a_{k2}B \quad \ldots a_{kn}B\,)\operatorname{vec} C,\end{aligned}$$

where Let $A = (a_{ij})$. But this means that $\operatorname{vec}(BCA^T) = (A \otimes B)\operatorname{vec} C$. □

Let $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{p\times p}$ be non-singular, and $C \in \mathbf{R}^{p\times n}$. Consider the Kronecker linear system

$$(A \otimes B)x = \operatorname{vec} C, \tag{7.7.17}$$

which is of order np. Then by (7.7.14) the solution can be written

$$x = (A^{-1} \otimes B^{-1})\operatorname{vec} C = \operatorname{vec}(X), \quad X = B^{-1}CA^{-T}. \tag{7.7.18}$$

where C is the matrix such that $c = \operatorname{vec}(C)$. This reduces the operation count for solving (7.7.17) from $O(n^3p^3)$ to $O(n^2p + np^2)$.

7.7.4 Linear Algebra Software

The first collection of high quality software was a series of algorithms written in Algol 60 that appeared in a handbook edited by Wilkinson and Reinsch [68, 1971]. This contains 11 subroutines for linear systems, least squares, and linear programming and 18 routines for the algebraic eigenvalue problem.

The collection LINPACK of Fortran subroutines for linear systems that followed contained several important innovations; see Dongarra et al. [18, 1979]. As much as possible of the computations were performed by calls to so called Basic Linear Algebra Subprograms (BLAS) [48]. These identified frequently occurring

vector operations in linear algebra such as scalar product, adding of a multiple of one vector to another. For example, the operations

$$y := \alpha x + y, \qquad \alpha := \alpha + x^T y$$

in single precision was named SAXPY. By carefully optimizing these BLAS for each specific computer performance could be enhanced without sacrificing portability. LINPACK was followed by EISPACK, a collection of routines for the algebraic eigenvalue problem; see Smith et al. [58, 1976], B. S. Garbow et al. [29, 1977].

The original BLAS, now known as Level 1 BLAS, were found to be unsatisfactory for vector computers, the level 2 BLAS for matrix-matrix operations were introduced in 1988 [20]. These operations involve one matrix A and one or several vectors x and y, e.g., the real matrix-vector products

$$y := \alpha Ax + \beta y, \qquad y := \alpha A^T x + \beta y,$$

and

$$x := Tx, \qquad x := T^{-1}x, \qquad x := T^T x,$$

where α and β are scalars, x and y are vectors, A a matrix and T an upper or lower triangular matrix. The corresponding operations on complex data are also provided.

In most computers in use today the key to high efficiency is to avoid as much as possible data transfers between memory, registers and functional units, since these can be more costly than arithmetic operations on the data. This means that the operations have to be carefully structured. The LAPACK collection of subroutines [2] was initially released in 1992 to address these questions. LAPACK was designed to supersede and integrate the algorithms in both LINPACK and EISPACK. The subroutines are restructured to achieve much greater efficiency on modern high-performance computers. This is achieved by performing as much as possible of the computations by calls to so called Level 2 and 3 BLAS. These enables the LAPACK routines to combine high performance with portable code and is also an aid to clarity, portability and modularity.

Level 2 BLAS involve $O(n^2)$ data, where n is the dimension of the matrix involved, and the same number of arithmetic operations. However, on computers with hierarchical memories, as is now the rule, they failed to obtain adequate performance. Therefore level 3 BLAS were finally introduced in 1990 [19]. These were derived in a fairly obvious manner from some level 2 BLAS, by replacing the vectors x and y by matrices B and C,

$$C := \alpha AB + \beta C, \qquad C := \alpha A^T B + \beta C, \qquad C := \alpha AB^T + \beta C,$$

and

$$B := TB, \qquad B := T^{-1}B, \qquad B := T^T B,$$

Since level 3 BLAS use $O(n^2)$ data but perform $O(n^3)$ arithmetic operations and gives a surface-to-volume effect for the ratio of data movement to operations. This avoids excessive data movements between different parts of memory hierarchy. Level 3 BLAS are used in LAPACK, the linear algebra package that is the successor

of LINPACK, which achieves close to optimal performance on a large variety of computer architectures.

LAPACK is continually improved and updated and is available for free from `http://www.netlib.org/lapack95/`. Several specila forms of matrices are supported by LAPACK: General

General band
Positive definite
Positive definite packed
Positive definite band
Symmetric (Hermitian) indefinite
Symmetric (Hermitian) indefinite packed
Triangular
General tridiagonal
Positive definite tridiagonal

The LAPACK subroutines form the backbone of Cleve Moler's MATLAB system, which has simplified matrix computations tremendously.

LAPACK95 is a Fortran 95 interface to the Fortran 77 LAPACK library. It is relevant for anyone who writes in the Fortran 95 language and needs reliable software for basic numerical linear algebra. It improves upon the original user-interface to the LAPACK package, taking advantage of the considerable simplifications that Fortran 95 allows. LAPACK95 Users' Guide provides an introduction to the design of the LAPACK95 package, a detailed description of its contents, reference manuals for the leading comments of the routines, and example programs.

Review Questions

1. How many operations are needed (approximately) for
   (a) The LU factorization of a square matrix?
   (b) The solution of $Ax = b$, when the triangular factorization of A is known?

2 To compute the matrix product $C = AB \in \mathbf{R}^{m\times p}$ we can either use an outer product or an inner product formulation. Discuss the merits of the two resulting algorithms when A and B have relatively few nonzero elements.

Problems

1. Assume that for the nonsingular matrix $A_{n-1} \in \mathbf{R}^{(n-1)\times(n-1)}$ we know the LU factorization $A_{n-1} = L_{n-1}U_{n-1}$. Determine the LU factorization of the **bordered matrix** $A_n \in \mathbf{R}^{n\times n}$,

$$A_n = \begin{pmatrix} A_{n-1} & b \\ c^T & a_{nn} \end{pmatrix} = \begin{pmatrix} L_{n-1} & 0 \\ l^T & 1 \end{pmatrix} \begin{pmatrix} U_{n-1} & u \\ 0 & u_{nn} \end{pmatrix}.$$

Here $b, c \in \mathbf{R}^{n-1}$ and a_{nn} are given and $l, u \in \mathbf{R}^{n-1}$ and u_{nn} are to be determined.

2. The methods of forwards- and back-substitution extend to block triangular systems. Show that the 2×2 block upper triangular system

$$\begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

can be solved by block back-substitution provided that the diagonal blocks U_{11} and U_{22} are square and nonsingular.

3. Write a recursive LU Factorization algorithm based on the 2×2 block LU algorithm.

4. (a) Let $A \in \mathbf{R}^{m\times n}$, $B \in \mathbf{R}^{n\times p}$, with m and n even. Show that, whereas conventional matrix multiplication requires mnp multiplications (M) and $m(n-1)p$ additions (A) to form the product $C = AB \in \mathbf{R}^{m\times p}$, Strassen's algorithm, using conventional matrix multiplication at the block level, requires

$$\frac{7}{8}mnp \text{ M } + \frac{7}{8}m(n-2)p + \frac{5}{4}n(m+p) + 2mp \text{ A}.$$

(b) Show, using the result in (a), that if we assume that "M $\approx$ A", Strassen's algorithm is cheaper than conventional multiplication when $mnp \leq 5(mn + np + mp)$.

5. Show the equality

$$\text{vec}\,(A)^T \text{vec}\,(B) = \text{trace}\,(A^T B). \tag{7.7.19}$$

7.8 Sparse Linear Systems

7.8.1 Introduction

A matrix $A \in \mathbf{R}^{n\times n}$ is called **sparse** if only a small fraction of its elements are nonzero. Similarly, a linear systems $Ax = b$ is called sparse if its matrix A is sparse. The simplest class of sparse matrices is the class of banded matrices treated in Sec. 7.4. These have the property that in each row all nonzero elements are contained in a relatively narrow band centered around the main diagonal. Matrices of small bandwidth occur naturally, since they correspond to a situation where only variables "close" to each other are coupled by observations.

Large sparse linear systems of more general structure arise in numerous areas of application such as the numerical solution of partial differential equations, mathematical programming, structural analysis, chemical engineering, electrical circuits and networks, etc. Large could imply a value of n in the range 1,000–1,000,000. Typically, A will have only a few (say, 5–30) nonzero elements in each row, regardless of the value of n. In Fig. 7.8.1 we show a sparse matrix of order 479 with 1887 nonzero elements and its LU factorization. It is a matrix W called west0479 in the Harwell–Boeing sparse matrix test collection, see Duff, Grimes and Lewis [23]. It comes from a model due to Westerberg of an eight stage chemical distillation column. Other applications may give pattern with quite different characteristics.

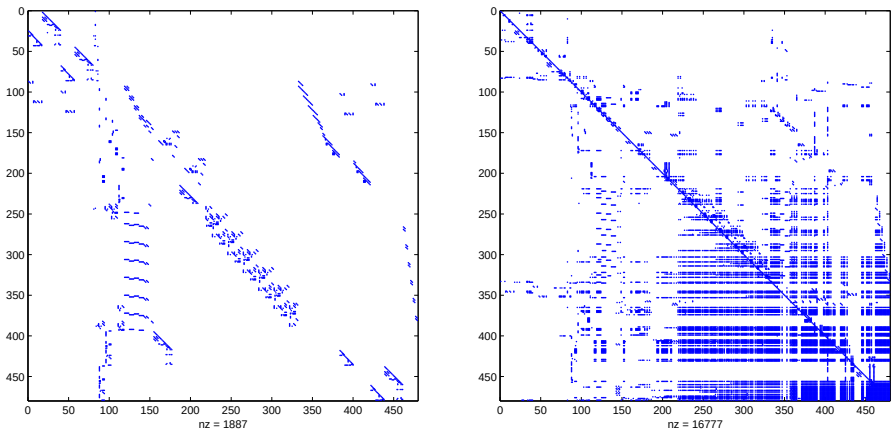


Figure 7.8.1. *Nonzero pattern of a matrix W and its LU factors.*

For many sparse linear systems iterative methods (see Chapter 11) may be preferable to use. This is particularly true of linear systems derived by finite difference methods for partial differential equations in two and three dimensions. In this section we will study elimination methods for sparse systems. These are easier to develop as black box algorithms. Iterative methods, on the other hand, often have to be specially designed for a particular class of problems.

When solving sparse linear systems by direct methods it is important to avoid storing and operating on the elements which are known to be zero. One should also try to minimize **fill-in** as the computation proceeds, which is the term used to denote the creation of new nonzeros during the elimination. For example, as shown in Fig. 7.8.1, the LU factors of W contain 16777 nonzero elements about nine times as many as in the original matrix. The object is to reduce storage and the number of arithmetic operations. Indeed, without exploitation of sparsity, many large problems would be totally intractable.

7.8.2 Storage Schemes for Sparse Matrices

A simple scheme to store a sparse matrix is to store the nonzero elements in an unordered one-dimensional array AC together with two integer vectors ix and jx containing the corresponding row and column indices.

$$ac(k) = a_{i,j}, \quad i = ix(k), \quad j = jx(k), \quad k = 1 : nz.$$

Hence A is stored in "coordinate form" as an unordered set of triples consisting of a numerical value and two indices. This scheme is very convenient for the initial representation of a general sparse matrix. Note that further nonzero elements are easily added to the structure. This coordinate form is very convenient for the original input of a sparse matrix. A drawback is that using this storage structure it is difficult to access the matrix A by rows or by columns, which is needed for the implementation of Gaussian elimination.

Example 7.8.1. The matrix

$$A = \begin{pmatrix} a_{11} & 0 & a_{13} & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{24} & 0 \\ 0 & a_{32} & a_{33} & 0 & a_{35} \\ 0 & a_{42} & 0 & a_{44} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix},$$

is stored in coordinate form as

$$\begin{aligned} AC &= (a_{13}, a_{22}, a_{21}, a_{33}, a_{35}, a_{24}, a_{32}, a_{42}, a_{44}, a_{55}, a_{54}, a_{11}) \\ i &= (1, 2, 2, 3, 3, 2, 3, 4, 4, 5, 5, 1) \\ j &= (3, 2, 1, 3, 5, 4, 2, 2, 4, 5, 4, 1) \end{aligned}$$

In some applications, one encounters matrices of banded structure, where the bandwidth differs from row to row. For this class of matrices, called **variable-band matrices**, we define

$$f_i = f_i(A) = \min\{j \mid a_{ij} \neq 0\}, \quad l_j = l_j(A) = \min\{i \mid a_{ij} \neq 0\}. \tag{7.8.1}$$

Here f_i is the column subscript of the first nonzero in the i-th *row* of A, and similarly l_j the row subscript of the first nonzero in the jth *column* of A. We assume here and in the following that A has a zero free diagonal. From the definition it follows that $f_i(A) = l_i(A^T)$. Hence for a symmetric matrix A we have $f_i(A) = l_i(A)$, $i = 1 : n$.

Definition 7.8.1.
The **envelope** *(or profile) of A is the index set*

$$\text{Env}\,(A) = \{(i, j) \mid f_i \leq j \leq i; \text{ or } l_j \leq i < j;\}. \tag{7.8.2}$$

The envelope of a symmetric matrix is defined by the envelope of its lower (or upper) triangular part including the main diagonal.

For a variable band matrix it is convenient to use a storage scheme, in which every element a_{ij}, $(i, j) \in \text{Env}\,(A)$ is stored. This means that zeros outside the envelope are exploited, but those inside the envelope are stored. This storage scheme is useful because of the important fact that only zeros inside the envelope will suffer fill-in during Gaussian elimination.

The proof of the following theorem is left as an exercise.

Theorem 7.8.2.
Assume that the triangular factors L and U of A exist. Then it holds that

$$\text{Env}\,(L + U) = \text{Env}\,(A),$$

i.e., the nonzero elements in L and U are contained in the envelope of A.

One of the main objectives of a sparse matrix data structure is to economize on storage while at the same time facilitating subsequent operations on the matrix. We now consider storage schemes that permits rapid execution of the elimination steps when solving general sparse linear systems. Usually the pattern of nonzero elements is very irregular, as illustrated in Fig. 7.8.1. We first consider a storage scheme for a sparse vector x. The nonzero elements of x can be stored in **compressed form** in a vector xc with dimension nnz, where nnz is the number of nonzero elements in x. Further, we store in an integer vector ix the indices of the corresponding nonzero elements in xc. Hence the sparse vector x is represented by the triple (nnz, xc, ix), where

$$xc_k = x_{ix(k)}, \quad k = 1 : nnz.$$

Example 7.8.2. The vector $x = (0, 4, 0, 0, 1, 0, 0, 0, 6, 0)$ can be stored in compressed form as

$$xc = (1, 4, 6), \qquad ix = (5, 2, 9), \qquad nnz = 3$$

Operations on sparse vectors are simplified if *one* of the vectors is first **uncompressed**, i.e, stored in a full vector of dimesnion n. Clearly this operation can be done in time proportional to the number of nonzeros, and allows direct random access to specified element in the vector. Vector operations, e.g., adding a multiple a of a sparse vector x to an uncompressed sparse vector y, or computing the inner product $x^T y$ can then be performed in *constant time per nonzero element*. Assume, for example, that the vector x is held in compressed form as nnz pairs of values and indices, and y is held in a full length array. Then the operation $y := a * x + y$ may be expressed as

$$\text{for } k = 1 : nnz, \quad y(ix(k)) := a * xc(k) + y(ix(k));$$

A matrix can be stored as a collection of sparse row vectors, where each row vector is stored in AC in compressed form. The corresponding column subscripts are stored in the integer vector jx, i.e., the column subscript of the element ac_k is given in $jx(k)$. Finally we need a third vector $ia(i)$, which gives the position in the array AC of the first element in the ith row of A. For example, the matrix in Example 7.8.1 is stored as

$$\begin{aligned} AC &= (a_{11}, a_{13} \mid a_{21}, a_{22}, a_{24} \mid a_{32}, a_{33}, a_{35} \mid a_{42}, a_{44} \mid a_{54}, a_{55}), \\ ia &= (1, 3, 6, 9, 11, 13), \\ jx &= (1, 3, 1, 2, 4, 2, 3, 5, 2, 4, 4, 5). \end{aligned}$$

Alternatively a similar scheme storing A as a collection of column vectors may be used. A drawback with these schemes is that it is expensive to insert new nonzero elements in the structure when fill-in occurs.

The components in each row need not be ordered; indeed there is often little advantage in ordering them. To access a nonzero a_{ij} there is no direct method

of calculating the corresponding index in the vector AC. Some testing on the subscripts in jx has to be done. However, more usual is that a complete row of A has to be retrieved, and this can be done quite efficiently. This scheme can be used unchanged for storing the lower triangular part of a symmetric positive definite matrix.

If the matrix is stored as a collection of sparse row vectors, the entries in a particular column cannot be retrieved without a search of nearly all elements. This is needed, for instance, to find the rows which are involved in a stage of Gaussian elimination. A solution is then to store also the structure of the matrix as a set of column vectors. If a matrix is input in coordinate form it the conversion to this storage form requires a sorting of the elements, since they may be in arbitrary order. Such a sortin can be done very efficiently in $O(n) + O(\tau)$ time.

Another way to avoid extensive searches in data structures is to use a linked list to store the nonzero elements. Associated with each element is a pointer to the location of the next element in its row and a pointer to the location of the next element in its column. If also pointer to the first nonzero in each row and column are stored there is a total overhead of integer storage of $2(\tau + n)$, where τ is the number of nonzero elements in the factors and n is the order of the matrix. This allows fill-ins to be added to the data structure with only two pointers being altered. Also the fill-in can be placed anywhere in storage so no reorderings are necessary. Disadvantages are that indirect addressing must be used when scanning a row or columnand that the elements in one row or column can be scattered over a wide range of memory.

An important distinction is between **static** storage structures that remain fixed and **dynamic** structures that can accommodate fill-in. If only nonzeros are to be stored, the data structure for the factors must dynamically allocate space for the fill-in during the elimination. A static structure can be used when the location of the nonzeros in the factors can be predicted in advance, as is the case for the Cholesky factorization.

7.8.3 Graph representation of sparse matrices.

In the method of normal equations for solving sparse linear least squares problems an important step is to determine a column permutation P such that the matrix $P^T A^T A P$ has a sparse Cholesky factor R, and to then generate a storage structure for R. This should be done symbolically using only the nonzero structure of A (or $A^T A$) as input. To perform such tasks the representation of the structure of a sparse matrix as a directed or undirected graph is a powerful tool.

A useful way to represent the structure of a symmetric matrix is by an **undirected graph** $G = (X, E)$, consisting of a set of **nodes** X and a set of **edges** E (unordered pairs of nodes). A graph is **ordered** (labeled) if its nodes are labeled. The ordered graph $G(A) = (X, E)$, representing the structure of a symmetric matrix $A \in \mathbf{R}^{n\times n}$, consists of nodes labeled $1, \ldots, n$ and edges $(x_i, x_j) \in E$ if and only if $a_{ij} = a_{ji} \neq 0$. Thus there is a direct correspondence between nonzero elements and edges in its graph; see Figure 6.4.1.

$$A = \begin{pmatrix} \times & \times & & \times & \times & & \\ \times & \times & \times & & & & \\ & \times & \times & & & \times & \times \\ \times & & & \times & & & \\ \times & & & & \times & & \\ & & \times & & & \times & \\ & & \times & & & & \times \end{pmatrix}$$

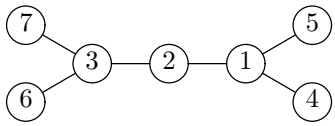


Figure 7.8.2. *The matrix A and its labeled graph.*

Two nodes, x and y, are said to be **adjacent** if there is an edge $(x, y) \in E$. The adjacency set of x in G is defined by

$$\mathrm{Adj}_G(x) = \{y \in X \mid x \text{ and } y \text{ are adjacent}\}.$$

The number of nodes adjacent to x is denoted by $|\mathrm{Adj}_G(x)|$, and is called the **degree** of x. A **path** of length $l \geq 1$ between two nodes, u_1 and u_{l+1}, is an ordered set of distinct nodes $u_1, \ldots, u_{l+1}$, such that

$$(u_i, u_{i+1}) \in E, \quad i = 1, \ldots, l.$$

If there is such a chain of edges between two nodes, then they are said to be **connected**. If there is a path between every pair of distinct nodes, then the graph is connected. A disconnected graph consists of at least two separate connected subgraphs. ($\bar{G} = (\bar{X}, \bar{E})$ is a subgraph of $G = (X, E)$ if $\bar{X} \subset X$ and $\bar{E} \subset E$.) If $G = (X, E)$ is a connected graph, then $Y \subset X$ is called a separator if G becomes disconnected after the removal and the nodes Y.

A symmetric matrix A is said to be **reducible** if there is a permutation matrix P such that $P^T AP$ is block diagonal. Such a symmetric permutation $P^T AP$ of A corresponds to a reordering of the nodes in $G(A)$ without changing the graph. It follows that the graph $G(P^T AP)$ is connected if and only if $G(A)$ is connected. It is then easy to prove that A is reducible if and only if its graph $G(A)$ is disconnected.

The structure of an unsymmetric matrix can similarly be represented by a **directed graph** $G = (X, E)$, where the edges now are ordered pairs of nodes. A directed graph is **strongly connected** if there is a path between every pair of distinct nodes along directed edges.

The structure of a symmetric matrix A can be represented by the **undirected graph of** A.

Definition 7.8.3.

The ordered undirected graph $G(A) = (X, E)$ of a symmetric matrix $A \in \mathbf{R}^{n\times n}$ consists of a set of n nodes *X together with a set E of* edges*, which are unordered pairs of nodes. The nodes are labeled $1, 2 : n$ where n, and nodes i and j are joined by an edge if and only if $a_{ij} = a_{ji} \neq 0$, $i \neq j$. We then say that the nodes i and j are* adjacent*. The number of edges incident to a node is called the* degree *of the node.*

The important observation is that for any permutation matrix $P \in \mathbf{R}^{n\times n}$ *the graphs $G(A)$ and $G(PAP^T)$ are the same except that the labelling of the nodes are different.* Hence the unlabeled graph represents the structure of A without any particular ordering. Finding a good permutation for A is equivalent to finding a good labeling for its graph.

7.8.4 Nonzero Diagonal and Block Triangular Form

Before performing a factorization of a sparse matrix it is often advantageous to perform some pre-processing. An arbitrary square nonsingular matrix $A \in \mathbf{R}^{n\times n}$ there always is a row permutation P such that PA has nonzero elements on it diagonal. Further, there is a row permutation P and column permutation Q such that PAQ has a nonzero diagonal and **block triangular structure**

$$PAQ = \begin{pmatrix} A_{11} & A_{12} & \dots & A_{1,t} \\ & A_{22} & \dots & A_{2,t} \\ & & \ddots & \vdots \\ & & & A_{tt} \end{pmatrix} \tag{7.8.3}$$

with square nonsingular diagonal blocks $A_{11}, \dots, A_{tt}$. The off-diagonal blocks are possibly nonzero matrices of appropriate dimensions. Using this structure a linear system $Ax = b$ or $PAQy = c$, where $y = Q^T x$, $c = Pb$, reduces to

$$A_{ii}y_i = c_i - \sum_{j=i+1}^{n} A_{ij}x_j, \quad j = n : -1 : 1. \tag{7.8.4}$$

Hence we only need to factorize the diagonal blocks. This block back-subsitution can lead to significant savings.

If we require that the diagonal blocks are irreducible, then the block triangular form (7.8.3) can be shown to be essentially unique. Any one block triangular form can be obtained from any other by applying row permutations that involve the rows of a single block row, column permutations that involve the columns of a single block column, and symmetric permutations that reorder the blocks. A square matrix which can be permuted to the form (7.8.3), with $t > 1$, is said to be **reducible**; otherwise it is called **irreducible**.

In the symmetric positive definite case a similar reduction to block upper triangular form can be considered, where $Q = P^T$. Some authors reserve the terms reducible for the case, and use the terms bi-reducible and bi-irreducible for the general case.

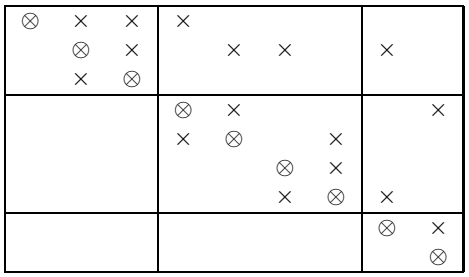


Figure 7.8.3. *The block triangular decomposition of A.*

An arbitrary rectangular matrix $A \in \mathbf{R}^{m\times n}$ has a block triangular form called the **Dulmage–Mendelsohn form**. If A is square and nonsingular this is the form (7.8.3). The general case is based on a canonical decomposition of bipartite graphs discovered by Dulmage and Mendelsohn. In the general case the first diagonal block may have more columns than rows, the last diagonal block more rows than column. All the other diagonal blocks are square and nonzero diagonal entries.This block form can be used for solving least squares problems by a method analogous to back-substitution.

The **bipartite graph** associated with A is denoted by $G(A) = \{R, C, E\}$, where $R = (r_1, \ldots, r_m)$ is a set of vertices corresponding to the rows of A and $C = (c_1, \ldots, c_m)$ a set of vertices corresponding to the columns of A. E is the set of edges, and $\{r_i, c_j\} \in E$ if and only if $a_{ij} \neq 0$. A **matching** in $G(A)$ is a subset of its edges with no common end points. In the matrix A this corresponds to a subset of nonzeros, no two of which belong to the same row or column. A **maximum matching** is a matching with a maximum number $r(A)$ of edges. The **structural rank** of A equals $r(A)$. Note that the mathematical rank is always less than or equal to its structural rank. For example, the matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

has structural rank 2 but numerical rank 1.

For the case when A is structurally nonsingular matrix there is a two-stage algorithm for permuting A to block upper triangular form. In the first stage a maximum matching in the bipartite graph $G(A)$ with row set R and column set C is found. In the second stage the block upper triangular form of each submatrix determined from the strongly connected components in the graph $G(A)$, with edges directed from columns to rows.

If A has structural rank n but is *numerically* rank deficient it will not be possible to factorize all the diagonal blocks in (7.8.3). In this case the block triangular structure given by the Dulmage–Mendelsohn form cannot be preserved, or some blocks may become severely ill-conditioned.

Note that for some applications, e.g., for matrices arising from discretizations of partial differential equations, it may be known a priori that the matrix is irre-

ducible. In other applications the block triangular decomposition may be known in advance from the underlying physical structure. In both these cases the algorithm discussed above is not useful.

7.8.5 LU Factorization of Sparse Matrices

Hence the first task in solving a sparse system is to order the rows and columns so that Gaussian elimination applied to the permuted matrix PAQ does not introduce too much fill-in. To find the *optimal* ordering, which minimizes the number of nonzero in L and U is unfortunately a hard problem. This is because the number of possible orderings of rows and columns is very large, $(n!)^2$, wheras solving a linear system only takes $O(n^3)$ operations. Fortunately, there are heuristic ordering algorithms which do a god job at approximately minimizing fill-in. These orderinga usually also nearly minimize the arithmetic operation count.

Example 7.8.3.

The ordering of rows and columns in Gaussian elimination may greatly affect storage and number of arithmewtic operations as shown by the following example. Let

$$A = \begin{pmatrix} \times & \times & \times & \dots & \times \\ \times & \times & & & \\ \times & & \times & & \\ \vdots & & & \ddots & \\ \times & & & & \times \end{pmatrix}, \qquad PAP^T = \begin{pmatrix} \times & & & & \times \\ & \ddots & & & \vdots \\ & & \times & & \times \\ & & & \times & \times \\ \times & \dots & \times & \times & \times \end{pmatrix}.$$

Matrices, or block matrices of this structure are called **arrowhead matrices** and occur in many applications.

If the $(1,1)$ element in A is chosen as the first pivot the fill in will be total and $n^3/3$ operations required for the LU factorization. In PAP^T the orderings of rows and columns have been reversed. Now there is no fill-in except in the last step of, when pivots are chosen in natural order.Only about $2n$ flops are required to perform the factorization.

For variable-band matrices no fill-in occurs in L and U outside the envelope. One strategy therefore is to choose P and Q to approximately minimize the envelope of PAQ. (Note that the reordered matrix PAP^T in Example 7.8.3 has a small envelope but A has a full envelope!) For symmetric matrices the reverse Cuthill–McKee ordering is often used. In the unsymmetirc case one can determine a reordering of the columns by applying this algorithm to the symmetric structure of $A + A^T$.

Perhaps surprisingly, the orderings that approximately minimize the total fill-in in LU factorization tend *not* to give a small bandwidth. Typically, the factors L and U instead have their nonzeros scattered throughout their triangular parts. A simple column reordering is to sort the columns by increasing column count, i.e. by the number of nonzeros in each column. This can often give a substantial reduction of the fill-in in Gaussian elimination. In Figure 7.8.5 we show the LU factorization

of the matrix W reordered after column count and its LU factors. The number of nonzeros in L and U now are 6604, which is a substantial reduction.

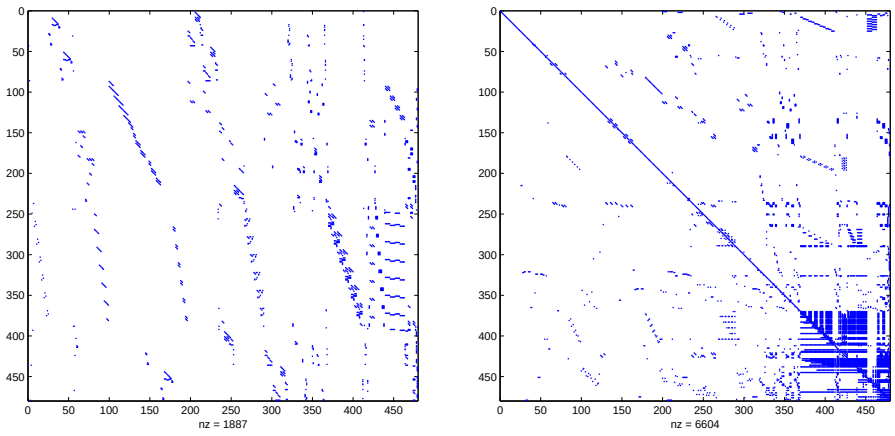


Figure 7.8.4. *Nonzero pattern of a matrix and its LU factors after reordering by increasing column count.*

An ordering that often performs even better is the so called column minimum degree ordering shown in Figure 7.8.5. The LU factors of the reordered matrix now containg 5904 nonzeros. This column ordering is obtained by using the symmetric minimum degree described in the next section on the matrix W^TW. MATLAB uses an implementation of this ordering algorithm that does not actually form the matrix W^TW. For the origin and details of this code we refer to Gilbert, Moler, and Schreiber [33].

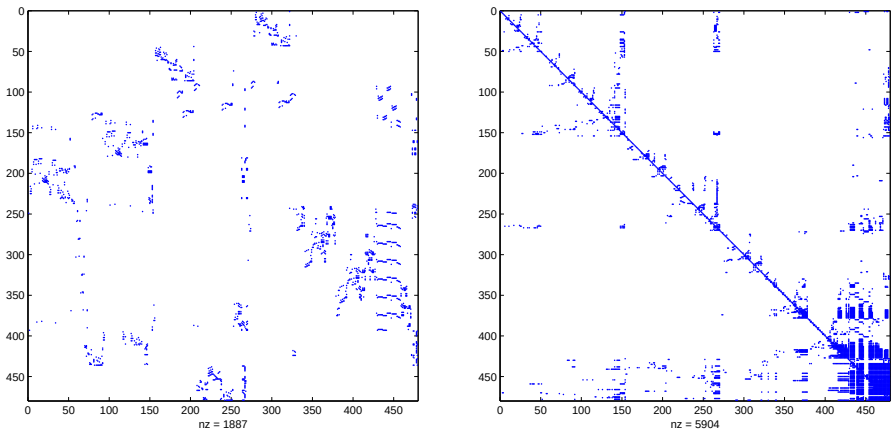


Figure 7.8.5. *Nonzero pattern of a matrix and its LU factors after minimum degree ordering.*

For unsymmetric systems some kind of stability check on the pivot elements must be performed during the numerical factorization. Therefore the storage structure for L and U cannot be predicted from the structure of A only. but must be determined dynamically during the numerical elimination phase.

MATLAB uses the column sweep method with partial pivoting due to Gilbert and Peierls [34] for computing the LU factorization a column of L and U at a time. In this the basic operation is to solve a series of sparse triangular system involving the already computed part of L. The column-oriented storage structure is set up dynamically as the factorization progresses. Note that the size of storage needed can not be predicted in advance. The total time for this LU factorization algorithm can be shown to be proportional to the number of arithmetic operations plus the size of the result.

Other sparse LU algorithms reorders both rows and columns before the numerical factorization. One of the most used ordering algorithm is the **Markowitz algorithm**. To motivate this suppose that Gaussian elimination has proceeded through k stages and let $A^{(k)}$ be the remaining active submatrix. Denote by r_i is the number of nonzero elements in the ith row and c_j is the number of nonzero elements in the jth column of $A^{(k)}$. In the Markowitz algorithm one performs a row and column interchange so that the product

$$(r_i - 1)(c_j - 1),$$

is minimized. (Some rules for tie-breaking are also needed.) This is equivalent to a *local minimization* of the fill-in at the next stage, assuming that all entries modified were zero beforehand. This choice also minimizes the number of multplications required for this stage.

With such an unsymmetric reordering there is a conflict with ordering for sparsity and for stability. The ordering for sparsity may not give pivotal elements which are acceptable from the point of numerical stability. Usually a **threshold pivoting** scheme is used to minimize the reorderings. This means that the chosen pivot is restricted by an inequality

$$|a_{ij}^{(k)}| \geq \tau \max_r |a_{rj}^{(k)}|, \tag{7.8.5}$$

where τ, $0 < \tau \leq 1$, is a predetermined threshold value. A value of $\tau = 0.1$ is usually recommended as a good compromise between sparsity and stability. (Note that the usual partial pivoting strategy is obtained for $\tau = 1$.) The condition (7.8.5) ensures that in any column that is modified in an elimination step the maximum element increases in size by at most a factor of $(1 + 1/\tau)$. Note that a column is only modified if the pivotal row has a nonzero element in that column. The total number of times a particular column is modified during the complete elimination is often quite small if the matrix is sparse. Furthermore, it is possible to monitor stability by, for example, computing the relative backward error, see Sec. 7.5.2.

7.8.6 Cholesky Factorization of Sparse Matrices

If A is symmetric and positive definite, then the Cholesky factorization is numerically stable for any choice of pivots along the diagonal. We need only consider

symmetric permutations PAP^T, where P can be chosen with regard only to sparsity. This, leads to a substantial increase in the efficiency of the sparse Cholesky algorithm since a static storage structure can be used.

We remark that the structure predicted for R from that of P^TAP by performing the Cholesky factor symbolically, is such that $R + R^T$ will be at least as full as PAP^T. In Figure 7.8.6 we show the nonzero pattern of the matrix $S = WW^T$, where W is the matrix west0479, and its Cholesky factor.

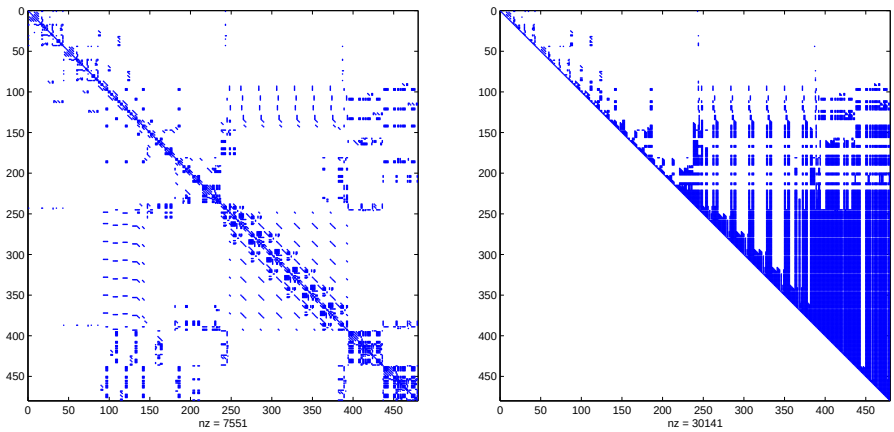


Figure 7.8.6. *Nonzero pattern of a matrix and its Cholesky factor.*

The Cholesky factorization of a sparse symmetric positive definite matrix A can be divided into four separate steps:

1. Determine a permutation P such that P^TAP has a sparse Cholesky factor L.

2. Perform a symbolic Cholesky factorization of PAP^T and generate a storage structure for R.

3. Form P^TAP and store in data structure for R.

4. Compute numerically the Cholesky factor R such that $P^TAP = R^TR$.

We stress that steps 1 and 2 are done symbolically, only working on the structure of A. The numerical computations take place in steps 3 and 4 a static storage scheme can be used.

Example 7.8.4.
To illustrate the symbolic factorization we use the sparse symmetric matrix A

with Cholesky factor R

$$A = \begin{pmatrix} \times & \times & & \times & \times & & \\ \times & \times & \times & & & & \\ & \times & \times & & & \times & \times \\ \times & & & \times & & & \\ \times & & & & \times & & \\ & & \times & & & \times & \\ & & \times & & & & \times \end{pmatrix}, \qquad R = \begin{pmatrix} \times & \times & & \times & \times & & \\ & \times & \times & + & + & & \\ & & \times & + & + & \times & \times \\ & & & \times & + & & \\ & & & & \times & & \\ & & & & & \times & \\ & & & & & & \times \end{pmatrix},$$

where $\times$ and $+$ denote a nonzero element. We show only the nonzero structure of A and R, not any numerical values. The five elements marked $+$ are the fill-in that occur in the Cholesky factorization.

Graph theory provides a powerful tool for the analysis and implementation of ordering algorithms. In the following we restrict ourselves to the case of a symmetric structure. Below is the ordered graph $G(A)$, of the matrix in Example 7.8.4.

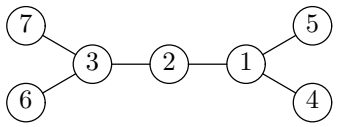


Figure 7.8.7. *The labeled graph of the matrix A.*

Example 7.8.5.

The labelled graph suggest that row and columns of the matrix in Example 7.8.5 is rearranged in order $4, 5, 7, 6, 3, 1, 2$. With this ordering the Cholesky factor of the matrix PAP^T will have no fill-in!

$$PAP^T = \begin{pmatrix} \times & & & & & \times & \\ & \times & & & & \times & \\ & & \times & & \times & & \\ & & & \times & \times & & \\ & & \times & \times & \times & & \times \\ \times & \times & & & & \times & \times \\ & & & & \times & \times & \times \end{pmatrix}, \qquad R = \begin{pmatrix} \times & & & & & \times & \\ & \times & & & & \times & \\ & & \times & & \times & & \\ & & & \times & \times & & \\ & & & & \times & & \times \\ & & & & & \times & \times \\ & & & & & & \times \end{pmatrix},$$

From the graph $G(A^TA)$ the structure of the Cholesky factor R can be predicted by using a graph model of Gaussian elimination. The fill-in under the factorization process can be analyzed by considering a sequence of **elimination graphs** that can be recursively formed as follows. We take $G_0 = G(A)$, and form G_i from $G_{(i-1)}$ by removing the node i and its incident edges and adding fill edges. The fill edges in eliminating node v in the graph G are

$$\{(j,k) \mid (j,k) \in \mathrm{Adj}_G(v), j \neq k\}.$$

Thus the fill edges correspond to the set of edges required to make the adjacent nodes of v pairwise adjacent. The filled graph $G_F(A)$ of A is a graph with n vertices and edges corresponding to all the elimination graphs G_i, $i = 0, \ldots, n-1$. The filled graph bounds the structure of the Cholesky factor R,

$$G(R^T + R) \subset G_F(A). \tag{7.8.6}$$

Under a no-cancellation assumption, the relation (7.8.6) holds with equality.

The following characterization of the filled graph describes how it can be computed directly from $G(A)$.

Theorem 7.8.4. *Let $G(A) = (X, E)$ be the undirected graph of A. Then (x_i, x_j) is an edge of the filled graph $G_F(A)$ if and only if $(x_i, x_j) \in E$, or there is a path in $G(A)$ from node i to node j passing only through nodes with numbers less than* $\min(i, j)$.

Consider the structure of the Cholesky factor $R = (r_{ij}$. For each row $i \leq n$ we define $\gamma(i)$ by

$$\gamma(i) = \min\{j > i \mid r_{ij} \neq 0\}, \tag{7.8.7}$$

that is $\gamma(i)$ is the column subscript of the first off-diagonal nonzero element in row i of R. If row i has no off-diagonal nonzero, then $\gamma(i) = i$. Clearly $\gamma(n) = n$. The quantities $\gamma(i)$, $i = 1 : n$ can be used to represent the structure of the sparse Cholesky factor R. For the matrix R in Example 7.8.4 we have

i	1	2	3	4	5	6	7
$\gamma(i)$	2	3	6	4	5	6	7

We now introduce the **elimination tree** corresponding to the structure of the Cholesky factor. The tree has n nodes, labelled form 1 to n. For each i if $\gamma(i) > j$, then node $\gamma(i)$ is the **parent** of node i in the elimination tree and node j is one of possible several **child** nodes of node $\gamma(i)$. If the matrix is irreducible then n is the only node with $\gamma(n) = n$ and is the **root** of the tree. There is exactly one path from node i to the root. If node j lies on the pathe from node i to the root, then node j is an ancestor to node i and node j is a descendant of node i.

The most widely used algorithm for envelope reduction for symmetric matrices is the **reverse Cuthill–McKee ordering**. This works on the graph $G(A)$ as follows:

1. Determine a starting node and label this 1.
2. For $i = 1 : n - 1$ find all unnumbered nodes adjacent to the node with label i, and number them in increasing order of degree.
3. The reverse ordering is obtained by reversing the ordering just determined.

The reversal of the Cuthill–McKee ordering in step 3 was suggested by Alan George, who noticed that it often was much superior to the original ordering produced by steps 1 and 2 above. In order for the algorithm to perform well it is

necessary to choose a good starting node; see George and Liu [31, Section 4.3.3]. In Fig. 7.8.2 we show the structure of the matrix from Fig. 7.8.1 and its Cholesky factor after reverse Cuthill–McKee reordering. The number of non-zero elements in the Cholesky factor is 23,866.

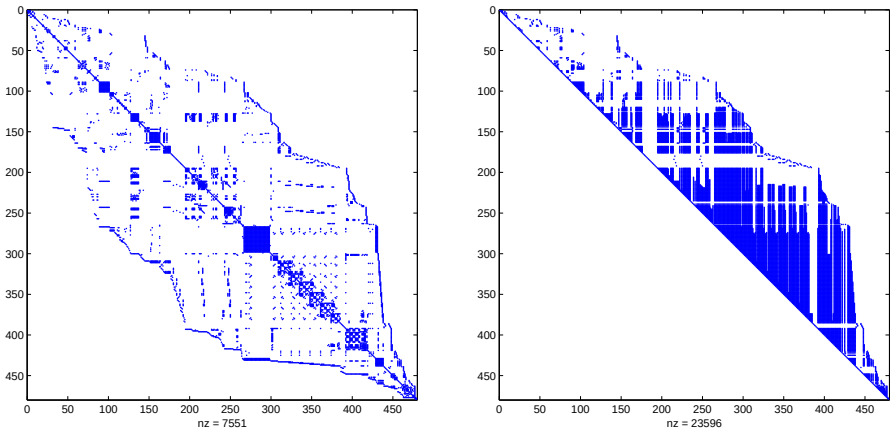


Figure 7.8.8. *Matrix and its Cholesky factor after reverse Cuthill–McKee reordering.*

As for unsymmetric matrices, the orderings that approximately minimize the total fill-in in the Cholesky factor tend to have their nonzeros scattered throughout the matrix. For some problems, such orderings can reduce fill-in by one or more orders of magnitude over the corresponding minimum bandwidth ordering.

In the symmetric case $r_i = c_i$ for the Markowitz ordering. It is then equivalent to minimizing r_i, and the resulting algorithm is called the **minimum-degree algorithm**. The minimum degree ordering can be determined using a graph model of the Cholesky factorization. At the same time the nonzero structure of the Cholesky factor R can be determined and a storage structure for R generated. The minimum-degree algorithm ordering algorithm has been subject to an extensive development. Very efficient implementations now exist. For details we refer to George and Liu [31, *Chapter* 5] and [32].

Figure 7.8.3 shows the structure of the matrix from Fig. 7.8.1 and its Cholesky factor after minimum-degree reordering. The number of non-zero elements in the Cholesky factor is reduced to 12,064. For nested dissection orderings, see George and Liu [31, Chapter 8].

Review Questions

1. Describe the coordinate form of storing a sparse matrix. Why is this not suitable for performing the numerical LU factorization?

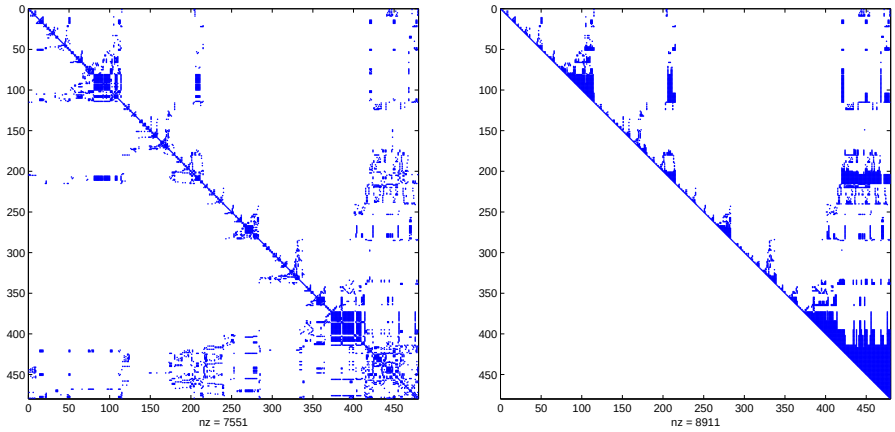


Figure 7.8.9. *Matrix and its Cholesky factor after minimum-degree reordering.*

2. Give an example of a sparse matrix A, which suffers extreme fill-in in Gaussian elimination..

3. Describe the Markowitz algorithm for ordering rows and columns of a non-symmetric matrix before factorization.

4. Describe threshold pivoting. Why is this used instead of partial pivoting in some schemes for LU factorization?

5. What does the reverse Cuthill–McKee ordering minimize?

Problems

1. Let $A, B \in \mathbf{R}^{n\times n}$ be sparse matrices. Show that the number of multiplications to compute the product $C = AB$ is $\sum_{i=1}^{n} \eta_i\theta_i$, where η_i denotes the number of nonzero elements in the ith *column* of A and θ_i the number of nonzeros in the ith *row* of B.

Hint: Use the outer product formulation $C = \sum_{i=1}^{n} a_{\cdot i}b_i^T$.

2. (a) It is often required to add a multiple a of a sparse vector x to another sparse vector y. Show that if the vector x is held in coordinate form as nx pairs of values and indices, and y is held in a full length array this operation may be expressed thus:

$$\begin{aligned}&\text{for } k = 1 : nx\\ &\quad y(\text{index}(k)) = a * x(k) + y(\text{index}(k));\end{aligned}$$

(b) Give an efficient algorithm for computing the inner product of two compressed vectors.

3. Consider a matrix with the symmetric structure

$$A = \begin{pmatrix} \times & & \times & & & \\ & \times & & \times & \times & \\ \times & & \times & \times & & \\ & \times & \times & & & \times \\ & \times & & & \times & \times \\ & & & \times & \times & \times \end{pmatrix}.$$

(a) What is the envelope of A? Where will fill-in occur during Gaussian elimination?

(b) Draw the undirected graph G, which represents the sparsity structure of A.

7.9 Structured Systems

The coefficient matrices in systems of linear equations arising from signal processing, control theory and linear prediction often have some special structure that can be taken advantage of. Several classes of such structured systems can be solved by fast methods in $O(n^2)$ operations, or by super-fast methods even in $O(n \log n)$ operations rather than $O(n^3)$ otherwise required by Gaussian elimination. This has important implications for many problems in signal restoration, acoustics, seismic exploration and many other application areas. Since the numerical stability properties of super-fast methods are generally either bad or unknown we consider only fast methods in the following.

7.9.1 Toeplitz and Hankel Matrices

Note: The following subsection are not yet complete and will be amended.

A **Toeplitz matrix** T is a matrix whose entries are constant along every diagonal; $T = (t_{i-j})_{1 \le i,j \le n}$,

$$T = \begin{pmatrix} t_0 & t_1 & \dots & t_{n-1} \\ t_{-1} & t_0 & \dots & t_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ t_{-n+1} & t_{-n+2} & \dots & t_0 \end{pmatrix} \in \mathbf{R}^{n \times n},$$

and is defined by the $2n-1$ values of $t_{-n+1}, \dots, t_0, \dots, t_{n-1}$. Toeplitz matrices arising in applications are often large, and dimensions of $10,000$ not uncommon. Consequently there is a need for special fast methods for solving Toeplitz systems. In large problems also storage requirements are important. The original matrix T only requires $2n-1$ storage. However, if standard factorization methods are used, at least $n(n+1)/2$ storage is needed.

A **Hankel matrix** is a matrix whose elements are constant along every an-

tidiagonal, i.e.,$H = (h_{i+j-2})_{1\le i,j\le n}$

$$H = \begin{pmatrix} h_0 & h_1 & \dots & h_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-2} & h_{n-1} & \dots & h_{2n-3} \\ h_{n-1} & h_n & \dots & h_{2n-2} \end{pmatrix} \in \mathbf{R}^{n\times n}.$$

Reversing the rows (or columns) of a Hankel matrix we get a Toeplitz matrix. Hence methods developed for solving Toeplitz systems apply also to Hankel systems.

7.9.2 Cauchy-Like Matrices

A **Cauchy matrix** is a matrix of the following form:

$$C = \left(\frac{1}{y_i - z_j}\right)_{1\le i,j\le n}, \qquad a_i, b_j \in \mathbf{R}^p. \tag{7.9.1}$$

where we assume that $y_i \neq z_j$ for $1 \le i, j \le n$.

Example 7.9.1. Consider the problem of finding the coefficients of a rational function

$$r(x) = \sum_{j=1}^{n} a_j \frac{1}{x - y_j},$$

which satisfies the interpolation conditions $r(x_i) = f_i$, $i = 1, \dots, n$. With $a = (a_1, \dots, a_n)$, $f = (f_1, \dots, f_n)$ this leads to the linear system $Ca = f$, where C is the Cauchy matrix in (7.9.1).

Cauchy gave in 1841 the following explicit expression for the determinant

$$\det(C) = \frac{\prod_{1\le i<j\le n} (y_j - y_i)(z_j - z_i)}{\prod_{1\le i\le j\le n} (y_j + z_i)}.$$

We note that any row or column permutation of a Cauchy matrix is again a Cauchy matrix. This property allows fast and stable version of Gaussian to be developed for Cauchy systems.

Many of these methods also apply in the more general case of **Loewner matrices** of the form

$$C = \left(\frac{a_i^T b_j}{y_i - z_j}\right)_{1\le i,j\le n}, \qquad a_i, b_j \in \mathbf{R}^p. \tag{7.9.2}$$

Example 7.9.2. The most famous example of a Cauchy matrix is the **Hilbert matrix**, which is obtained by taking $y_i = z_i = i - 1/2$:

$$H_n \in \mathbf{R}^{n\times n}, \quad h_{ij} = \frac{1}{i+j-1}.$$

For example,

$$H_4 = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{pmatrix}.$$

The Hilbert matrix is symmetric and positive definite Hankel matrix. It is also **totally positive** . The inverse of H_n is known explicitly and has integer elements. Hilbert matrices of high order are known to be very ill-conditioned; for large n it holds that $\kappa_2(H_n) \sim e^{3.5n}$.

7.9.3 Vandermonde systems

In Chapter 4 the problem of interpolating given function values $f(\alpha_i)$, $i = 1, \ldots, n$ at distinct points α_i with a polynomial of degree $\leq n-1$ was shown to lead to a linear system of equations with matrix $M = [p_j(\alpha_i)]_{i,j=1}^m$. In the case of the power basis $p_j(z) = z^{j-1}$, the matrix M equals V^T, where V is the **Vandermonde matrix**

$$V = [\alpha_j^{i-1}]_{i,j=1}^n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1^{n-1} & \alpha_2^{n-1} & \cdots & \alpha_n^{n-1} \end{pmatrix}. \tag{7.9.3}$$

Hence the unique polynomial $P(z)$ satisfying the interpolating conditions $P(\alpha_i) = f_i$, $i = 1, \ldots, n$ is given by

$$P(z) = (1, z, \ldots, z^{n-1})a,$$

where a is the solution of the dual Vandermonde system.

$$V^T a = f \tag{7.9.4}$$

One of the most efficient ways to determine $P(x)$ is by Newton's interpolation formula, which uses the basis polynomials

$$Q_1(z) = 1, \qquad Q_k(z) = (z-\alpha_1)\cdots(z-\alpha_{k-1}), \quad k = 2:n-1.$$

We write the polynomial in the form

$$P(z) = (Q_1(z), Q_2(z), \ldots, Q_n(z))c,$$

where c are the divided differences of $f_1 : f_n$. These divided differences can be recursively computed, see Section 4.?. This leads to the algorithm below for computing the coefficient vector a in the power basis. Note that the algorithm operates directly on the α_j's and the matrix V^T is never formed,

Algorithm 7.9.1 Dual Vandermonde System

Given distinct scalars $\alpha_1, \alpha_2, \ldots, \alpha_n$ and $f = (f_1, f_2, \ldots, f_n)^T$ the following algorithm solves the dual Vandermonde system $V^T a = f$:

$a = \text{dvand}(\alpha, f)$
  $a := f;$
  for $k = 1 : n-1$
    for $j = n : (-1) : k+1$
      $a_j := (a_j - a_{j-1})/(\alpha_j - \alpha_{j-k})$
    end
  end
  for $k = n-1 : (-1) : 1$
    for $j = k : n-1$
      $a_j := a_j - \alpha_k * a_{j+1}$
    end
  end

The accuracy of this algorithm depends on the ordering of the interpolation points α_i. Often the best ordering is the monotone ordering for which

$$\alpha_1 < \alpha_2 \cdots < \alpha_n.$$

If moreover $0 \leq \alpha_1$ this algorithm often gives remarkably accurate solutions.

To interpret the Newton interpolation algorithm in matrix terms we define lower bidiagonal matrices

$$L_k(\alpha) = \begin{pmatrix} I_{k-1} & 0 \\ 0 & B_{n-k+1}(\alpha) \end{pmatrix}, \quad k = 1, \ldots, n-1,$$

where

$$B_p(\alpha) = \begin{pmatrix} 1 & & & \\ -\alpha & 1 & & \\ & \ddots & \ddots & \\ & & -\alpha & 1 \end{pmatrix} \in \mathbf{R}^{p \times p}.$$

We further let

$$D_k = \text{diag}\,(1, \ldots, 1, (\alpha_{k+1} - \alpha_1), \ldots, (\alpha_n - \alpha_{n-k})).$$

Then we find that the dual Vandermonde algorithm can be written as

$$\begin{aligned} c &= U^T f, \qquad U^T = D_{n-1}^{-1} L_{n-1}(1) \cdots D_1^{-1} L_1(1), \\ a &= L^T c, \qquad L^T = L_1^T(\alpha_1) L_2^T(\alpha_2) \cdots L_{n-1}^T(\alpha_{n-1}). \end{aligned}$$

Systems of equations with Vandermonde matrix

$$Vx = b \tag{7.9.5}$$

are called primal Vandermonde systems and occur, e.g., in approximation of linear functionals (see Chapter 4). The matrix representation of the algorithm for the dual Vandermonde system allows us to derive an algorithm also for solving primal Vandermonde systems.

Since $a = V^{-T}f = L^T U^T f$, we have $V^{-T} = L^T U^T$. Transposing this relation and find

$$V^{-1} = UL,$$

Hence the solution to the primal system $Vx = b$ is given by $x = V^{-1}b = U(Lb)$ or

$$\begin{aligned} d &= Lb, \qquad L = L_{n-1}(\alpha_{n-1}) \cdots L_2(\alpha_2) L_1(\alpha_1) \\ x &= Uf, \qquad U = M_1^T D_1^{-1} \cdots M_{n-1}^T D_{n-1}^{-1} \end{aligned}$$

This gives rise to the following algorithm:

Algorithm 7.9.2 Primal Vandermonde System

Given distinct scalars $\alpha_1, \alpha_2, \ldots, \alpha_n$ and $b = (b_1, b_2, \ldots, b_n)^T$ the following algorithm solves the primal Vandermonde system $Vx = b$:

```
x = pvand(α, b)
  x := b;
  for k = 1 : n − 1
    for j = n : (−1) : k + 1
      x_j := x_j − α_k ∗ x_{j−1}
    end
  end
  for k = n − 1 : (−1) : 1
    for j = k + 1 : n
      x_j := x_j/(α_j − α_{j−k})
    end
    for j = k : n − 1
      x_j := x_j − x_{j+1}
    end
  end
```

This is the **Björck–Pereyra algorithm**. It solves primal Vandermonde systems with only $\frac{1}{2}n(n+1)(3A+2M)$ operations, where A and M denotes one floating point addition and multiplication, respectively. Note also that no extra storage is needed since a can overwrite f.

Notes

Although the history of Gaussian elimination goes back at least to Chinese mathematicians about 250 B.C., there was no practical experience of solving large linear

systems until the advent of computers in the 1940s. Gaussian elimination was the first numerical algorithm to be subjected to a rounding error analysis. In 1946 there was a mood of pessimism about the stability of Gaussian elimination. Hotelling [44] had produced bounds showing that the error in the solution would be proportional to 4^n, which suggested that it would be impossible to solve even systems of modest order. A few years later J. von Neumann and H. H. Goldstein published more relevant error bounds. In 1948 A. M. Turing wrote a remarkable paper [63], where he formulated the LU factorization and introduced matrix condition numbers. The more or less final form of error analysis of Gaussian elimination was given by J. H. Wilkinson [65]. For a more detailed historical perspective of Gaussian elimination we refer to N. J. Higham [41, Sec. 9.13].

Rook pivoting for nonsymmetric matrices was introduced by Neal and Poole in [51]. Related pivoting strategies for symmetric indefinite matrices were introduced earlier by Fletcher [26].

The idea of doing only half the elimination for symmetric systems, while preserving symmetry is probably due to Gauss, who first sketched his elimination algorithm in 1809. The Cholesky method is named after Andre-Louis Cholesky, who was a French military officer. He devised his method to solve symmetric, positive definite system arising in a geodetic survey in Crete and North Africa just before World War I.

The literature on linear algebra is very extensive. For a theoretical treatise a classical source is Gantmacher [28, 1959]. Several nonstandard topics are covered in depth in two excellent volumes by Horn and Johnson [42, 1985] and [43, 1991].

An interesting survey of classical numerical methods in linear algebra can be found in Faddeev and Faddeeva [25, 1963], but many of the methods treated are now dated. A compact, lucid and modern presentation is given in Householder [45, 1964]. Bellman [7, 1960] is an original and readable complementary text.

An up to date and indispensable book for of anyone interested in computational linear algebra is Golub and Van Loan [35, 1996]. The book by Higham [41, 2002] is a wonderful and useful source book for information about the accuracy and stability of algorithms in numerical linear algebra. Other excellent textbooks on matrix computation include Stewart [60, 1998]. For results on on perturbation theory and related topics a very complete reference book is Stewart and Sun [61, 1990]. In particular, an elegant treatise on norms and metrics is found in [61, Chapter II].

Direct methods for sparse symmetric positive definite systems are covered in George and Liu [31, 1981], while a more general treatise is given by Duff et al. [22, 1986].

Bauer [6] was the first to study componentwise perturbation theory. This did not catch on in English publications until Skeel took it up in two papers [56, 1979], and [57, 1980].

A gallery of test matrices is documented in N. J. Higham [40]. available from the Web,; see also [41, Appendix D].

Bibliography

[1] Jan Ole Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971.

[2] Edward Anderson, Zhaojun Bai, Christian Bischof, S. Blackford, J. Demmel, J. Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, A. McKenney, and Danny Sorensen, editors. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[3] Mario Arioli, James W. Demmel, and Iain S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.*, 10:165–190, 1989.

[4] Cleve Ashcraft, Roger G. Grimes, and John G. Lewis. Accurate symmetric indefinite linear system solvers. *SIAM J. Matrix Anal. Appl.*, 20:2:513–561, 1998.

[5] Edgar Asplund. Inverses of matrices $\{a_{ij}\}$ which satisfy $a_{ij} = 0$ for $j > i + p$. *Math. Scand.*, 7:57–60, 1959.

[6] F. L. Bauer. Genauigkeitsfragen bei der Lösung linearer Gleichungssysteme. *Z. Angew. Math. Mech.*, 46:7:409–421, 1966.

[7] Richard Bellman. *Introduction to Matrix Analysis*. SIAM, Philadelphia, PA, 1995.

[8] Åke Björck and Tommy Elfving. Algorithms for confluent Vandermonde systems. *Numer. Math.*, 21:130–137, 1973.

[9] Åke Björck and Victor Pereyra. Solution of Vandermonde system of equations. *Math. Comp.*, 24:893–903, 1970.

[10] Z. Bothe. Bounds for rounding errors in the Gaussian elimination for band systems. *J. Inst. Maths. Applics.*, 16:133–142, 1975.

[11] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.*, 31:163–179, 1977.

[12] Eleanor Chu and Alan George. A note on estimating the error in gaussian elimination without pivoting. *ACM SIGNUM Newsletter*, 20:2:2–7, 1985.

[13] Philippe G. Ciarlet. *Introduction to Numerical Linear Algebra and Optimization*. Cambridge University Press, Cambridge, UK, 1989.

[14] Alan K. Cline, Cleve B. Moler, George W. Stewart, and James H. Wilkinson. An estimate for the condition number of a matrix. *SIAM J. Numer. Anal.*, 16:368–375, 1979.

[15] Carl de Boor and Allan Pinkus. Backward error analysis for totally positive linear systems. *Numer. Math.*, 27:485–490, 1977.

[16] James W. Demmel, Nicholas J. Higham, and Robert S. Schreiber. Stability of block LU factorizations. *Numer. Linear Algebra Appl.*, 2:173–190, 1995.

[17] Inderjit S. Dhillon. Reliable computation of the condition number of a tridiagonal matrix in $O(n)$ time. *SIAM J. Matrix Anal. Appl.*, 19:3:776–796, 1998.

[18] J. J. Dongarra, James R. Bunch, Cleve B. Moler, and George W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, PA, 1979.

[19] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Software*, 16:1–17, 1988.

[20] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. A extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Software*, 14:1–17, 1988.

[21] Jeremy Du Croz and Nicholas J. Higham. Stability of methods for matrix inversion. *IMA J. Numer. Anal.*, 12:1–19, 1992.

[22] Iain S. Duff, A. M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, London, 1986.

[23] Iain S. Duff, Roger G. Grimes, and John G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15:1:1–14, 1989.

[24] Erik Elmroth, F. G. Gustavson, Isak Jonsson, and Bo Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46:1, 2004.

[25] D. K. Faddeev and V. N. Faddeeva. *Computational Methods of Linear Algebra*. W. H. Freeman, San Francisco, CA, 1963.

[26] Roger Fletcher. Factorizing symmetric indefinite matrices. *Linear Algebra Appl.*, 14:257–272, 1976.

[27] George E. Forsythe and Cleve B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[28] F. R. Gantmacher. *The Theory of Matrices. Vols. I and II*. Chelsea Publishing Co, New York, 1959.

[29] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and G. W. Stewart. *Matrix Eigensystems Routines: EISPACK Guide Extension*. Springer-Verlag, New York, 1977.

[30] Alan George, Kkakim D. Ikramov, and Andrey B. Kucherov. On the growth factor in Gaussian elimination for generalized Higham matrices. *Numer. Linear Algebra Appl.*, 9:107–114, 2002.

[31] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[32] Alan George and Joseph W. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[33] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 9:862–874, 1992.

[34] John R. Gilbert and Tim Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sc. Statist. Comput.*, 13:1:333–356, 1988.

[35] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[36] William W. Hager. Condition estimates. *SIAM J. Sci. Statist. Comput.*, 5:311–316, 1984.

[37] Eldon Hansen. *Topics in Interval Analysis*. Oxford University Press, Oxford, 1969.

[38] G. I. Hargreaves. Interval analysis in MATLAB. Numer. anal. report 418, Department of Mathematics, University of Manchester, 2002.

[39] Nicholas J. Higham. FORTRAN codes for estimating the one-norm of a real or complex matrix, with application to condition estimation. *ACM Trans. Math. Software*, 14:4:381–396, 1988.

[40] Nicholas J. Higham. The Matrix Computation Toolbox, 1995. `http://www.ma.man.ac.uk/~higham/mctoolbox`.

[41] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, second edition, 2002.

[42] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1985.

[43] Roger A. Horn and Charles R. Johnson. *Topics in Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1991.

[44] Harold Hotelling. Some new methods in matrix calculus. *Ann. Math. Statist.*, 14:1–34, 1943.

[45] Alston S. Householder. *The Theory of Matrices in Numerical Analysis*. Dover, New York, 1975.

[46] Yasuhiko Ikebe. On inverses of Hessenberg matrices. *Linear Algebra Appl.*, 24:93–97, 1979.

[47] W. M. Kahan. Numerical linear algebra. *Canad. Math. Bull.*, 9:757–801, 1966.

[48] Charles L. Lawson, Richard J. Hanson, D. R. Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Software*, 5:308–323, 1979.

[49] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, A. Kapur, M. C. Martin, T. Tung, and D. J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. LAPACK working note 149 Tech. Report CS-00-451, Department of Computer Science, University of Tennessee, Knoxville, TN, 2000.

[50] Jean Meinguet. Refined error analysis of Cholesky factorization. *SIAM J. Numer. Anal.*, 20:1243–1250, 1983.

[51] Larry Neal and George Poole. A geometric analysis of Gaussian elimination. II. *Linear Algebra Appl.*, 173:239–264, 1992.

[52] John von Neumann. In A. H. Taub, editor, *Collected Works*. Pergamon Press, New York, 1962.

[53] G. Peters and James H. Wilkinson. On the stability of Gauss–Jordan elimination with pivoting. *Comm. ACM*, 18:20–24, 1975.

[54] S. M. Rump. Fast and parallel interval arithmetic. *BIT*, 39:3:534–554, 1999.

[55] Siegfrid M. Rump. INTLAB—INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[56] Robert D. Skeel. Scaling for stability in Gaussian elimination. *J. Assoc. Comput. Mach.*, 26:494–526, 1979.

[57] Robert D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput.*, 35:817–832, 1980.

[58] B. T. Smith, J. M. Boyle, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystems Routines—EISPACK Guide*. Springer-Verlag, New York, second edition, 1976.

[59] Torsten Söderström and G. W. Stewart. On the numerical properties of an iterative method for computing the Moore–Penrose generalized inverse. *SIAM J. Numer. Anal.*, 11:61–74, 1974.

[60] George W. Stewart. *Matrix Algorithms Volume I: Basic Decompositions*. SIAM, Philadelphia, PA, 1998.

[61] George W. Stewart and Ji guang. Sun. *Matrix Perturbation Theory*. Academic Press, Boston, MA, 1990.

[62] Lloyd N. Trefethen and Robert S. Schreiber. Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11:335–360, 1990.

[63] A. M. Turing. Rounding-off errors in matrix processes. *Quart. J. Mech. Appl. Math.*, 1:287–308, 1948.

[64] James M. Varah. On the solution of block-tridiagonal systems arising from certain finite-difference equations. *Math. Comp.*, 26(120):859–869, 1972.

[65] James H. Wilkinson. Error analysis of direct methods of matrix inversion. *J. ACM*, 8:281–330, 1961.

[66] James H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.

[67] James H. Wilkinson. A priori error analysis of algebraic processes. In *Proceedings International Congress Math.*, pages 629–639. Izdat. Mir, Moscow, 1968.

[68] James H. Wilkinson and C. Reinsch, editors. *Handbook for Automatic Computation. Vol. II, Linear Algebra*. Springer-Verlag, New York, 1971.

[69] Max A. Woodbury. Inverting modified matrices. Memorandum Report 42, Statistical Research Group, Princeton, 1950.

Index

algorithm
  LDL^T, 57
  1-norm estimator, 90
  back-substitution, 30
    banded, 74
  band LU, 74
  band-Cholesky, 76
  block Cholesky, 112
  block LU factorization, 111
  block-Cholesky factorization, 112
  Cholesky factorization, 62
  forward-substitution
    banded, 74
  Gaussian elimination, 34
  recursive Cholesky factorization, 115
  Vandermonde system, 139
    dual, 138
antidiagonal, 7
arithmetic
  floating-point, 92
  standard model, 92
array operations, 5
arrowhead matrix, 127

back-substitution, 29
  banded, 74
banded matrix, 72
banded systems, 70–76
bandwidth
  lower, 7
  of LU factors, 73
  upper, 7
bidiagonal matrix, 8, 73
BLAS, 103
block diagonally dominant, 111
block triangular form, 125–127
  algorithm, 127
block triangular structure, 125
bordered matrix, 120
bordering method, 49

Cauchy matrix, 136
Cauchy–Schwarz inequality, 18
Cayley transform, 27
Cholesky factorization, 61–76
  backward error, 98, 99
  symbolic, 133
componentwise perturbation bound, 84
condition estimation, 88–91
  Hager's, 89
  LINPACK's, 89
condition number
  of matrix, 24
convergence
  of vectors and matrices, 22
Cramer's rule, 2, 9
Crout's algorithm, 49
cyclic reduction, 78

decomposition
  SVD, 15
diagonally dominant, 56
distance
  to singular matrices, 25
Doolittle's algorithm, 49
dual
  norm, 18
  vector, 18
dual norm, 18
Dulmage–Mendelsohn form, 126

element growth, 43, 45
  in complete pivoting, 94

in partial pivoting, 94
partial pivoting, 95
elimination
right-looking, 111
elliptic norm, 18
envelope
of LU factors, 123
of matrix, 123
error
floating point rounding, 93
error bounds
a posteriori, 86, 87
backward, 86
Euler expansion, 53
expansion
Euler, 53
Neumann, 53

fill-in, 122
flam, 30
flam count
Gaussian elimination, 34
triangular system, 35
flop count
LDL^T, 58
banded back-substitution, 74
banded LU, 74
condition estimation, 88
Gauss–Jordan elimination, 42
Hessenberg system, 75
inverse matrix, 53
tridiagonal system, 77
forward-substitution, 29
banded, 74

Gauss–Jordan elimination, 42
Gaussian elimination, 31–51
backward error, 44
block algorithms, 109–116
compact schemes, 49, 109
rounding error analysis, 92–99
scaling invariance, 99
GE, *see* Gaussian elimination
Gerschgorin's Theorem, 56
graph
bipartite, 126
elimination, 132
filled, 132
ordered, 131
undirected, 131
growth rate, 43
growth ratio, 43

Hölder inequality, 18
Hankel matrix, 136
Hessenberg matrix, 8, 75
Hilbert matrix, 137

ill-conditioned
artificial, 86
Inertia
of symmetric matrices, 64–65
inertia of matrix, 65
inner product, 6
accurate, 103
error analysis, 92
INTLAB, 108
inverse
approximative, 53
of band matrix, 80
product form of, 42
inverse matrix, 6
irreducible matrix, 126
iterative refinement
error bound, 104
of solutions, 102–105

Krawczyck's method, 108
Kronecker
product, 117
Kronecker product, 117
Kronecker symbol, 7

left-looking, 113
linear map, 4
linear system
consistent, 9
homogeneous, 9
ill-scaling, 101
scaling, 99–102
scaling rule, 101
linearly independent

vectors, 3
LU factorization, 35–38
Doolittle's algorithm, 48
theorem, 37

magnitude
of interval, 106
matrix
arrowhead, 127
banded, 72
bidiagonal, 8, 73
block, 10
bordered, 120
congruent, 64
diagonally dominant, 45–47, 111
elementary elimination, 39
Hermitian, 8
Hessenberg, 8, 75
indefinite, 55
inverse, 6, 51–54
permutation, 36
persymmetric, 8
positive definite, 45, 55–60
rank of, 33
semidefinite, 55
skew-Hermitian, 8
sparse, 121
symmetric, 55
totally positive, 98
trapezoidal form, 33
tridiagonal, 8, 73
variable-band, 123
well-conditioned, 26
matrix multiplication
error bound, 93
maximum matching, 126

Neumann expansion, 53
Newton–Schultz iteration, 54
no-cancellation assumption, 132
norm
consistent, 19
dual, 18
Frobenius, 20
matrix, 19
operator, 19
scaled, 18
spectral, 20
submultiplicative, 19
subordinate, 19
unitarily invariant, 21
vector, 17
weighted, 18
null space (of matrix), 16

odd-even reduction, 78
Oettli–Prager error bounds, 87
ordering
Markowitz, 129
minimum-degree, 133
reverse Cuthill–McKee, 128, 133
outer product, 6

packed storage, 63
partitioning
conformal, 10
partitioning (of matrix), 10
permutation
even, 8
odd, 8
sign of, 8
permutation matrix, 36
perturbation
of linear systems, 86
perturbation bound
for linear system, 24
component-wise, 85
pivotal elements, 31
pivoting
Bunch–Kaufman, 68
complete, 42
for sparsity, 127–133
partial, 42
rook, 44
positive semidefinite matrices, 64

range (of matrix), 16
rank
structural, 127
reducible matrix, 126
right-looking, 113
rook pivoting, 44

Schur
    complement, 12, 66
Schur–Banachiewicz formula, 13
Sherman–Morrison formula, 14
singular value, 15
singular value decomposition, 14–15
singular vector, 15
sparse matrix
    block triangular form, 125–127
    irreducible, 126
    reducible, 126
standard basis, 3
storage scheme
    compressed form, 124
    dynamic, 125
    static, 125
Strassen's algorithm, 116
submatrix, 10
    principal, 10
subspaces
    dimension, 3
    intersection of, 3
    sum of, 3
SVD, *see* singular value decomposition
    compact form, 16
sweep method, 50
Sylvester's
    criterion, 59
    law of inertia, 65
symmetric
    gauge functions, 21
    indefinite matrix, 66–69
    matrix, 55
    pivoting, 63

Toeplitz matrix, 136
totally positive, 137
transformation
    congruence, 64
transpose (of matrix), 5
transposition, 8
    matrix, 36
triangular
    factorization, *see* LU factorization
    matrix, 29
    systems of equations, 30
tridiagonal
    matrix, 8, 73
    systems, 81
tridiagonal matrix
    periodic, 79
    symmetric indefinite, 80

Woodbury formula, 13
wrapping effect, 105

Contents

8 Linear Least Squares Problems **1**

8.1 Preliminaries 1
8.1.1 The Least Squares Principle 1
8.1.2 Linear Models and the Gauss–Markoff Theorem . . 2
8.1.3 Generalized Inverses 4
8.1.4 Matrix Approximation and the SVD 7
8.1.5 Perturbation Analysis 11
8.1.6 Backward Error and Stability 13
Review Questions 15
Problems 15
8.2 The Method of Normal Equations 16
8.2.1 Characterization of Least Squares Solutions 16
8.2.2 Forming and Solving the Normal Equations 18
8.2.3 Stability and Accuracy with Normal Equations . . . 21
8.2.4 Scaling Least Squares Problems 23
8.2.5 Methods Based on Gaussian Elimination 25
Review Questions 27
Problems 28
8.3 Methods using Orthogonal Factorizations 31
8.3.1 Orthogonal and Oblique Projections 31
8.3.2 Gram–Schmidt Orthogonalization 34
8.3.3 Least Squares Problems by Gram–Schmidt 40
8.3.4 Householder and Givens Transformations 42
8.3.5 Householder QR Factorization 47
8.3.6 Least Squares Problems by QR Factorization 53
8.3.7 Condition and Error Estimation 56
Review Questions 57
Problems 58
8.4 Rank Deficient and Ill-Posed Problems 59
8.4.1 Regularized Least Squares Problems 59
8.4.2 QR Factorization and Rank Deficient Matrices . . . 63
8.4.3 Rank Revealing QR Factorization 65
8.4.4 The URV and ULV decompositions 67
8.4.5 Bidiagonal Decomposition and Least Squares 69

Review Questions . . . 77
Problems . . . 77
8.5 Some Structured Least Squares Problems . . . 79
8.5.1 Banded Least Squares Problems . . . 79
8.5.2 Two-Block Least Squares Problems . . . 82
8.5.3 Block Triangular Form of a Rectangular Matrix . . . 83
8.5.4 Block Angular Least Squares Problems . . . 85
8.5.5 Kronecker Product Problems . . . 87
Review Questions . . . 89
Problems . . . 89
8.6 Generalized Least Squares . . . 90
8.6.1 Generalized Least Squares . . . 90
8.6.2 Weighted Problems . . . 93
8.6.3 Generalized Orthogonal Decompositions . . . 96
8.6.4 Indefinite Least Squares . . . 96
8.6.5 Orthogonal Regression . . . 98
8.6.6 Linear Equality Constraints . . . 100
Review Questions . . . 102
Problems . . . 102
8.7 Total Least Squares . . . 103
8.7.1 The Total Least Squares Problem . . . 103
8.7.2 Total Least Squares Problem and the SVD . . . 104
8.7.3 Conditioning of the TLS Problem . . . 105
8.7.4 Bidiagonalization and TLS Problems. . . . 107
8.7.5 Some Generalized TLS Problems . . . 109
8.7.6 Iteratively Reweighted Least Squares. . . . 112
Review Questions . . . 113
Problems and Computer Exercises . . . 113

Bibliography **117**

Index **122**

Chapter 8

Linear Least Squares Problems

8.1 Preliminaries

8.1.1 The Least Squares Principle

A fundamental task in scientific computing is to estimate parameters in a mathematical model from collected data which are subject to errors. The influence of the errors can be reduced by using a greater number of data than the number of unknowns. If the model is linear, the resulting problem is then to "solve" an in general inconsistent linear system $Ax = b$, where $A \in \mathbf{R}^{m\times n}$ and $m \geq n$. In other words, we want to find a vector $x \in \mathbf{R}^n$ such that Ax is in some sense the "best" approximation to the known vector $b \in \mathbf{R}^m$.

There are many possible ways of defining the "best" solution to an inconsistent linear system. A choice which can often be motivated for statistical reasons (see Theorem 8.1.4) and also leads to a simple computational problem is the following: Let x be a vector which minimizes the Euclidian length of the **residual vector** $r = b - Ax$; i.e., a solution to the minimization problem

$$\min_x \|Ax - b\|_2, \tag{8.1.1}$$

where $\|\cdot\|_2$ denotes the Euclidian vector norm. Note that this problem is equivalent to minimizing the sum of squares of the residuals $\sum_{i=1}^m r_i^2$. Hence, we call (8.1.1) a **linear least squares problem** and any minimizer x a **least squares solution** of the system $Ax = b$. [1]

Example 8.1.1. Consider a model described by a scalar function $y(t) = f(x, t)$, where $x \in \mathbf{R}^n$ is a parameter vector to be determined from measurements (y_i, t_i), $i = 1, \ldots, m$, $m > n$. In particular let $f(x, t)$ be *linear* in x,

$$f(x, t) = \sum_{j=1}^n x_j \phi_j(t).$$

[1] This draft last revised 2003 10 31.

Then the equations $y_i = \sum_{j=1}^{n} x_j \phi_j(t_i)$, $i = 1, \ldots, m$ form an overdetermined system, which can be written in matrix form $Ax = b$, where $a_{ij} = \phi_j(t_i)$, and $b_i = y_i$.

We shall see that a least squares solution x is characterized by $r \perp \mathcal{R}(A)$, where $\mathcal{R}(A)$ the range space of A. The residual vector r is always uniquely determined and the solution x is unique if and only if $\text{rank}\,(A) = n$, i.e., when A has linearly independent columns. If $\text{rank}\,(A) < n$, we seek the unique least squares solution of minimum Euclidean norm.

When there are more variables than needed to match the observed data, then we have an **underdetermined problem**. In this case we can seek the **minimum norm solution** $y \in \mathbb{R}^m$ of a linear system, i.e. solve

$$\min \|y\|_2, \qquad A^T y = c, \tag{8.1.2}$$

where $c \in \mathbb{R}^n$ and $A^T y = c$ is assumed to be consistent.

8.1.2 Linear Models and the Gauss–Markoff Theorem

We first need to introduce some concepts from statistics. Let the probability that random variable $y \leq x$ be equal to $F(x)$, where $F(x)$ is nondecreasing, right continuous, and satisfies

$$0 \leq F(x) \leq 1, \quad F(-\infty) = 0, \quad F(\infty) = 1.$$

Then $F(x)$ is called the **distribution function** for y.

The **expected value** and the **variance** of y are defined as the Stieltjes integrals

$$\mathcal{E}(y) = \mu = \int_{-\infty}^{\infty} y dF(y), \qquad \mathcal{E}(y-\mu)^2 = \sigma^2 = \int_{-\infty}^{\infty} (y-\mu)^2 dF(y),$$

If $y = (y_1, \ldots, y_n)^T$ is a vector of random variables and $\mu = (\mu_1, \ldots, \mu_n)^T$, $\mu_i = \mathcal{E}(y_i)$, then we write $\mu = \mathcal{E}(y)$. If y_i and y_j have the joint distribution $F(y_i, y_j)$ the **covariance** between y_i and y_j is

$$\begin{aligned} \sigma_{ij} &= \mathcal{E}[(y_i - \mu_i)(y_j - \mu_j)] = \int_{-\infty}^{\infty} (y_i - \mu_i)(y_j - \mu_j) dF(y_i, y_j) \\ &= \mathcal{E}(y_i y_j) - \mu_i \mu_j. \end{aligned}$$

The covariance matrix $V \in \mathbf{R}^{n \times n}$ of y is defined by

$$V = \mathcal{V}(y) = \mathcal{E}[(y-\mu)(y-\mu)^T] = \mathcal{E}(yy^T) - \mu\mu^T.$$

where the diagonal element σii is the variance of y_i.

We now prove some properties which will be useful in the following.

Lemma 8.1.1.
Let $B \in \mathbf{R}^{r\times n}$ be a matrix and y a random vector with $\mathcal{E}(y) = \mu$ and covariance matrix V. Then

$$\mathcal{E}(By) = B\mu, \qquad \mathcal{V}(By) = BVB^T.$$

In the special case that $B = b^T$ is a row vector, $r = 1$, then $\mathcal{V}(b^T y) = \mu\|b\|_2^2$.

Proof. The first property follows directly from the definition of expected value. The second follows from the relation

$$\begin{aligned}\mathcal{V}(By) &= \mathcal{E}[(B(y-\mu)(y-\mu)^T B^T] \\ &= B\mathcal{E}[(y-\mu)(y-\mu)^T]B^T = BVB^T.\end{aligned}$$

□

In linear statistical models one assumes that the vector $b \in \mathbf{R}^m$ of observations is related to the unknown parameter vector $x \in \mathbf{R}^n$ by a linear relationship

$$Ax = b + \epsilon, \tag{8.1.3}$$

where $A \in \mathbf{R}^{m\times n}$ is a known matrix, and, ϵ is a vector of random errors. In the **standard case** ϵ has zero mean and covariance matrix $\sigma^2 I$, i.e.,

$$\mathcal{E}(\epsilon) = 0, \qquad \mathcal{V}(\epsilon) = \sigma^2 I.$$

We also assume that rank $(A) = n$, and make the following definitions:

Definition 8.1.2.
A function g of the random vector y is called unbiased estimate of a parameter θ if $\mathcal{E}(g(y)) = 0$. When such a function exists, then θ is called an estimable parameter.

Definition 8.1.3.
The linear function $g = c^T y$, where c is a constant vector, is a minimum variance (best) unbiased estimate of the parameter θ if $\mathcal{E}(g) = \theta$, and $\mathcal{V}(g)$ is minimized over all linear estimators.

Gauss gave the method of least squares a sound theoretical basis in [23, 1821], without any assumptions that the random variables follow a normal distribution. This contribution of Gauss was somewhat neglected until rediscovered by Markoff 1912. We state the relevant theorem without proof.

Theorem 8.1.4. The Gauss–Markoff theorem.
Consider the linear model (8.1.3)*, where $A \in \mathbf{R}^{m\times n}$ is a known matrix, and ϵ is a random vector with zero mean and covariance matrix $\mathcal{V}(\epsilon) = \sigma^2 I$. Let $\hat{x}$ be the least square estimator, obtained by minimizing over x the sum of squares*

$\|Ax-b\|_2^2$. *Then the best linear unbiased estimator of any linear function* $g = c^T x$ *is* $c^T\hat{x}$. *Furthermore, the covariance matrix of the estimate* $\hat{x}$ *equals*

$$\mathcal{V}(\hat{x}) = V = \sigma^2(A^TA)^{-1} \tag{8.1.4}$$

and $\mathcal{E}(s^2) = \sigma^2$, *where* s^2 *is the quadratic form*

$$s^2 = \frac{1}{m-n}\|b - A\hat{x}\|_2^2.$$

Proof. See Zelen [67]. □

In the next subsection we show that the residual vector $\hat{r} = \hat{b} - Ax$ satisfies $A^T\hat{r} = 0$. Hence there are n linear relations among the m components of $\hat{r}$. It can be shown that the residuals $\hat{r}$ and therefore also s^2 are uncorrelated with $\hat{x}$, i.e.,

$$\mathcal{V}(\hat{r}, \hat{x}) = 0, \qquad \mathcal{V}(s^2, \hat{x}) = 0.$$

In the **general univariate linear model** the covariance matrix equals $\mathcal{V}(\epsilon) = \sigma^2 W$, where $W \in \mathbf{R}^{m\times m}$ is a positive semidefinite symmetric matrix. For full column rank A and positive definite W the best unbiased linear estimate is the solution of

$$\min_x (Ax-b)^T W^{-1}(Ax-b). \tag{8.1.5}$$

In particular, if the errors are uncorrelated with variances $w_{ii} > 0$, $i = 1, \ldots, m$, then W is diagonal and the best estimate is obtained form the problem the **weighted least squares** problem

$$\min_x \|D^{-1}(Ax-b)\|_2, \quad D = \operatorname{diag}(\sqrt{w_{11}}, \ldots, \sqrt{w_{mm}}). \tag{8.1.6}$$

Hence if the ith equation is scaled by $1/\sqrt{w_{ii}}$ we get the standard case. This is consistent with the obvious observation that the larger the variance the smaller weight should be given to a particular equation. It is important to note that different scalings will give different solutions, unless the system is *consistent*, i.e., $b \in \mathcal{R}(A)$.

8.1.3 Generalized Inverses

IN

The SVD is a powerful tool both for analyzing and solving linear least squares problems. The reason for this is that the orthogonal matrices that transform A to diagonal form do not change the l_2-norm. We have the following fundamental result.

Theorem 8.1.5.
Let $A \in \mathbf{R}^{m\times n}$, $\operatorname{rank}(A) = r$, *and consider the general linear least squares problem*

$$\min_{x\in S}\|x\|_2, \qquad S = \{x \in \mathbf{R}^n \mid \|b - Ax\|_2 = \min\}. \tag{8.1.7}$$

This problem always has a unique solution, which in terms of the SVD of A can be written as

$$x = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T b, \tag{8.1.8}$$

Proof. Let

$$c = U^T b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix},$$

where $z_1, c_1 \in \mathbf{R}^r$. Using the orthogonal invariance of the l_2 norm we have

$$\begin{aligned} \|b - Ax\|_2 &= \|U^T(b - AVV^T x)\|_2 \\ &= \left\| \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \right\|_2 = \left\| \begin{pmatrix} c_1 - \Sigma_1 z_1 \\ c_2 \end{pmatrix} \right\|_2. \end{aligned}$$

The residual norm will attain its minimum value equal to $\|c_2\|_2$ for $z_1 = \Sigma_1^{-1} c_1$, z_2 arbitrary. Obviously the choice $z_2 = 0$ minimizes $\|x\|_2 = \|Vz\|_2 = \|z\|_2$. □

Note that problem (8.1.7) includes as special cases the solution of both overdetermined and underdetermined linear systems. We can write $x = A^\dagger b$, where

$$A^\dagger = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T \in \mathbf{R}^{n \times m} \tag{8.1.9}$$

is the unique **pseudo-inverse** of A and x is called the pseudo-inverse solution of $Ax = b$.

Methods for computing the SVD are described in Sec. 10.8. Note that for solving least squares problems we only need to compute the singular values, the matrix V_1 and vector $c = U_1^T b$, where we have partitioned $U = (U_1\ U_2)$ and $V = (V_1\ V_2)$ so that U_1 and V_1 have $r = \operatorname{rank}(A)$ columns. The pseudo-inverse solution (8.1.9) can then be written

$$x = V_1 \Sigma_1^{-1} U_1^T b = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} \cdot v_i, \quad r = \operatorname{rank}(A). \tag{8.1.10}$$

The matrix $A^\dagger$ is often called the **Moore–Penrose inverse**. Moore 1920 developed the concept of the general reciprocal in 1920. Penrose [1955], gave an elegant algebraic characterization and showed that $X = A^\dagger$ is uniquely determined by the four **Penrose conditions** :

$$(1) \quad AXA = A, \qquad (2) \quad XAX = X, \tag{8.1.11}$$

$$(3) \quad (AX)^T = AX, \qquad (4) \quad (XA)^T = XA. \tag{8.1.12}$$

It can be directly verified that $X = A^\dagger$ given by (8.1.9) satisfies these four conditions. In particular this shows that $A^\dagger$ does not depend on the particular choices of U and V in the SVD. (See also Problem 2.)

The orthogonal projections onto the four fundamental subspaces of A have the following simple expressions in terms of the pseudo-inverse :

$$P_{\mathcal{R}(A)} = AA^\dagger, \qquad P_{\mathcal{N}(A^T)} = I - AA^\dagger, \tag{8.1.13}$$
$$P_{\mathcal{R}(A^T)} = A^\dagger A, \qquad P_{\mathcal{N}(A)} = I - A^\dagger A.$$

These expressions are easily verified using the definition of an orthogonal projection and the Penrose conditions.

Another very useful characterization of the pseudo-inverse solution is the following:

Theorem 8.1.6. *The pseudo-inverse solution $x = A^\dagger b$ is uniquely characterized by the two geometrical conditions*

$$x \perp \mathcal{N}(A), \qquad Ax = P_{\mathcal{R}(A)}\, b. \tag{8.1.14}$$

Proof. These conditions are easily verified from (8.1.10). □

In the special case that $A \in \mathbf{R}^{m\times n}$ and $\operatorname{rank}(A) = n$ it holds that

$$A^\dagger = (A^TA)^{-1}A^T, \qquad (A^T)^\dagger = A(A^TA)^{-1} \tag{8.1.15}$$

These expressions follow from the normal equations (8.2.3) and (8.2.4). Some properties of the usual inverse can be extended to the pseudo-inverse, e.g., the relations

$$(A^\dagger)^\dagger = A, \qquad (A^T)^\dagger = (A^\dagger)^T,$$

easily follow form (8.1.9). In general $(AB)^\dagger \neq B^\dagger A^\dagger$. The following theorem gives a useful *sufficient* conditions for the relation $(AB)^\dagger = B^\dagger A^\dagger$ to hold.

Theorem 8.1.7.
If $A \in \mathbf{R}^{m\times r}$, $B \in \mathbf{R}^{r\times n}$, and $\operatorname{rank}(A) = \operatorname{rank}(B) = r$, then

$$(AB)^\dagger = B^\dagger A^\dagger = B^T(BB^T)^{-1}(A^TA)^{-1}A^T. \tag{8.1.16}$$

Proof. The last equality follows from (8.1.15). The first equality is verified by showing that the four Penrose conditions are satisfied. □

A matrix X which only satisfy some of the Penrose conditions is called a **generalized inverse**. A matrix X is called an **inner inverse** or $\{1\}$-inverse if it satisfies condition (1). Any matrix X which satisfies condition (2) is called an **outer inverse** or a $\{2\}$-inverse. A matrix which satisfies conditions (1) and (3), is called a $\{1,3\}$-inverse, etc.

Let X be a $\{1\}$-inverse of $A \in \mathbf{C}^{m\times n}$. Then for all b such that $Ax = b$ is consistent $x = Xb$ is a solution. The general solution can be written

$$x = Xb + (I - XA)y, \quad y \in \mathbf{C}^n.$$

Let $A \in \mathbf{R}^{m\times n}$ of rank r and X an $\{1\}$-inverse. Then $AXA = A$ and we have

$$(AX)^2 = AXAX = AX, \qquad (XA)^2 = XAXA = XA.$$

This shows that AX and XA are idempotent and therefore (in general oblique) projectors

$$AX = P_{\mathcal{R}(A),S}, \qquad XA = P_{T,\mathcal{N}(A)},$$

where S and T are some subspaces complementary to $\mathcal{R}(A)$ and $\mathcal{N}(A)$, respectively.

If A is a $\{1,3\}$-inverse, then AX is symmetric and therefore is the orthogonal projector onto $\mathcal{R}(A)$. Similarly, if A is a $\{1,4\}$-inverse, then XA is symmetric and therefore the orthogonal projector orthogonal to $\mathcal{N}(A)$.

Theorem 8.1.8.

Let $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$. Then $\|Ax - b\|_2$ is the smallest when $x = Xb$, where X is a $\{1,3\}$-inverse.

Conversely, if $X \in \mathbf{R}^{n\times m}$ has the property that for all b, $\|Ax-b\|_2$ is smallest when $x = Xb$, then X is a $\{1,3\}$-inverse.

Theorem 8.1.9.

Let $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$. If $Ax = b$ has a solution, the unique solution for which $\|x\|_2$ is smallest is given by $x = Xb$, where X is a $\{1,4\}$-inverse.

Conversely, if $X \in \mathbf{R}^{n\times m}$ is such that, whenever $Ax = b$ has a solution, $x = Xb$ is the solution of smallest norm, then X is a $\{1,4\}$-inverse.

8.1.4 Matrix Approximation and the SVD

A useful relationship between the SVD and a symmetric eigenvalue problem is given in the following theorem.

Theorem 8.1.10. *Let the SVD of $A \in \mathbf{R}^{m\times n}$ be $A = U\Sigma V^T$, where $U = \mathbf{R}^{m\times m}$ and $V \in \mathbf{R}^{n\times n}$ are orthogonal. Let $r = \operatorname{rank}(A) \le \min(m,n)$ and $\Sigma_1 = \operatorname{diag}(\sigma_1,\dots,\sigma_r) > 0$. Then it holds that*

$$C = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} = Q \begin{pmatrix} \Sigma & 0 & 0 \\ 0 & -\Sigma & 0 \\ 0 & 0 & 0 \end{pmatrix} Q^T, \tag{8.1.17}$$

where

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} U_1 & U_1 & \sqrt{2}\,U_2 & 0 \\ V_1 & -V_1 & 0 & \sqrt{2}\,V_2 \end{pmatrix}. \tag{8.1.18}$$

and U and V have been partitioned conformally. Hence the eigenvalues of C are $\pm\sigma_1, \pm\sigma_2, \dots, \pm\sigma_r$, and zero repeated $(m+n-2r)$ times.

Proof. Form the product on the right hand side of (8.1.17) and note that $A = U_1\Sigma_1 V_1^T$ and $A^T = V_1\Sigma_1 U_1^T$. □

The singular values have the following important extremal property, the **minimax characterization**.

Theorem 8.1.11.
Let $A \in \mathbf{R}^{m\times n}$ have singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, $p = \min(m,n)$, and S be a linear subspace of $\mathbf{R}^n$ of dimension $\dim(S)$. Then

$$\sigma_i = \min_{dim(S)=n-i+1} \max_{\substack{x\in S \\ x\neq 0}} \frac{\|Ax\|_2}{\|x\|_2}. \tag{8.1.19}$$

Proof. The result is established in almost the same way as for the corresponding eigenvalue theorem, Theorem 10.3.9 (Fischer's theorem). □

The minimax characterization of the singular values may be used to establish the following relations between the singular values of two matrices A and B.

Theorem 8.1.12.
Let $A, B \in \mathbf{R}^{m\times n}$ have singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p$ and $\tau_1 \geq \tau_2 \geq \ldots \geq \tau_p$ respectively, where $p = \min(m,n)$. Then

$$\max_i |\sigma_i - \tau_i| \leq \|A - B\|_2, \tag{8.1.20}$$

$$\sum_{i=1}^{p} |\sigma_i - \tau_i|^2 \leq \|A - B\|_F^2. \tag{8.1.21}$$

Proof. See Stewart [1973, pp. 321-322]. □

Hence perturbations of the elements of a matrix A result in perturbations of the same, or smaller, magnitude in the singular values. This result is important for the use of the SVD to determine the "numerical rank" of a matrix; see below.

The eigenvalues of the leading principal minor of order $n-1$ of a Hermitian matrix C can be shown to interlace the eigenvalues of C, see Theorem 10.3.8. From the relation (8.1.17) corresponding results can be derived for the singular values of a matrix A.

Theorem 8.1.13.
Let

$$\hat{A} = (A, u) \in \mathbf{R}^{m\times n}, \quad m \geq n, \quad u \in \mathbf{R}^m.$$

Then the ordered singular values σ_i of A interlace the ordered singular values $\hat{\sigma}_i$ of $\hat{A}$ as follows

$$\hat{\sigma}_1 \geq \sigma_1 \geq \hat{\sigma}_2 \geq \sigma_2 \ldots \geq \hat{\sigma}_{n-1} \geq \sigma_{n-1} \geq \hat{\sigma}_n.$$

Similarly, if A is bordered by a row,

$$\hat{A} = \begin{pmatrix} A \\ v^* \end{pmatrix} \in \mathbf{R}^{m\times n}, \quad m > n, \quad v \in \mathbf{R}^n,$$

then

$$\hat{\sigma}_1 \geq \sigma_1 \geq \hat{\sigma}_2 \geq \sigma_2 \ldots \geq \hat{\sigma}_{n-1} \geq \sigma_{n-1} \geq \hat{\sigma}_n \geq \sigma_n.$$

The SVD plays an important role in a number of matrix approximation problems. In the theorem below we consider the approximation of one matrix by another of lower rank.

Theorem 8.1.14. *Let $\mathcal{M}_k^{m\times n}$ denote the set of matrices in $\mathbf{R}^{m\times n}$ of rank k. Assume that $A \in \mathcal{M}_r^{m\times n}$ and consider the problem*

$$\min_{X\in\mathcal{M}_k^{m\times n}} \|A - X\|, \quad k < r.$$

Then the SVD expansion of A truncated to k terms $X = B = \sum_{i=1}^{k} \sigma_i u_i v_i^T$, solves this problem both for the l_2 norm and the Frobenius norm. Further, the minimum distance is given by

$$\|A - B\|_2 = \sigma_{k+1}, \qquad \|A - B\|_F = (\sigma_{k+1}^2 + \ldots + \sigma_r^2)^{1/2}.$$

The solution is unique for the Frobenius norm but not always for the l_2 norm.

Proof. See Mirsky [42] for the l_2, norm and Eckhard and Young[20] for the Frobenius norm. □

According to this theorem σ_i equals the distance in l_2 norm to the nearest matrix of rank $i-1$, $i \leq \min(m,n)$. In particular $\sigma_1 = \|A\|_2$.

Inaccuracy of data and rounding errors made during the computation usually perturb the ideal matrix A. In this situation the *mathematical* notion of rank may not be appropriate. For example, let A be a matrix of rank $r < n$, whose elements are perturbed by a matrix E of small random errors. Then it is most likely that the perturbed matrix $A + E$ has full rank n. However, $A + E$ is close to a rank deficient matrix, and should be considered as *numerically rank deficient*.

Clearly the **numerical rank** assigned to a matrix should depend on some tolerance δ, which reflects the error level in the data and/or the precision of the floating point arithmetic used. A useful definition is the following:

Definition 8.1.15.
A matrix $A \in \mathbf{R}^{m\times n}$ has numerical δ-rank equal to k ($k \leq \min\{m,n\}$) if

$$\sigma_1 \geq \ldots \geq \sigma_k > \delta \geq \sigma_{k+1} \geq \ldots \geq \sigma_n,$$

where σ_i, $i = 1, 2, \ldots, n$ are the singular values of A. If we write

$$A = U\Sigma V^T = U_1\Sigma_1 V_1^T + U_2\Sigma_2 V_2^T,$$

where $\Sigma_2 = \operatorname{diag}(\sigma_{k+1}, \ldots, \sigma_n)$ then $\mathcal{R}(V_2) = \operatorname{span}\{v_{k+1}, \ldots, v_n\}$ is called the **numerical nullspace** *of A.*

It follows from Theorem 8.1.12, that if the numerical δ-rank of A equals k, then $\operatorname{rank}(A+E) \geq k$ for all perturbations such that $\|E\|_2 \leq \delta$, i.e., such perturbations cannot *lower* the rank. Definition 8.1.15 is only useful when there is a well defined gap between σ_{k+1} and σ_k. This should be the case if the exact matrix A is rank deficient but well-conditioned. However, it may occur that there does not exist a gap for any k, e.g., if $\sigma_k = 1/k$. In such a case the numerical rank of A is not well defined!

If $r < n$ then the system is *numerically underdetermined*. Note that this can be the case even when $m > n$.

Let $A \in \mathbf{R}^{m\times n}$, be a matrix of rank n with the "thin" SVD $A = U_1 \Sigma V^T$. Since $A = U_1\Sigma V^T = U_1 \Sigma U_1^T U_1 V^T$ we have

$$A = PH, \qquad P = U_1 V^T, \qquad H = V\Sigma V^T, \tag{8.1.22}$$

where $P \in \mathbf{R}^{m\times n}$ has orthogonal columns, and $H \in \mathbf{R}^{n\times n}$ is symmetric, positive semidefinite. The decomposition (8.1.22) is called the **polar decomposition** of A, since it can be regarded as a generalization to matrices of the complex number representation $z = re^{i\theta}$, $r \geq 0$.

The significance of the factor P in the polar decomposition is that it is the closest matrix with orthogonal columns to A.

Theorem 8.1.16.

Let $\mathcal{M}_{m\times n}$ denote the set of all matrices in $\mathbf{R}^{m\times n}$ with orthogonal columns. Let $A \in \mathbf{R}^{m\times n}$ be a given matrix and $A = PH$ its polar decomposition, where $P \in \mathcal{M}_{m\times n}$ and H is symmetric positive semidefinite. Then for any matrix $Q \in \mathcal{M}_{m\times n}$,

$$\|A - Q\|_F \geq \|A - P\|_F.$$

Proof. This theorem was proved for $m = n$ and general unitarily invariant norms by Fan and Hoffman [21]. The generalization to $m > n$ follows from the additive property of the Frobenius norm. □

An generalization of Theorem 8.1.16 has important application in factor analysis in statistics.

Theorem 8.1.17.

Let $\mathcal{M}_{m\times n}$ denote the set of all matrices in $\mathbf{R}^{m\times n}$ with orthogonal columns. Let A and B be given matrices in $\mathbf{R}^{m\times n}$. If $B^T A = PH$ is the polar decomposition then for any matrix $Q \in \mathcal{M}_{m\times n}$ it holds that

$$\|A - BQ\|_F \geq \|A - BP\|_F.$$

Proof. See P. Schönemann [54]. □

8.1.5 Perturbation Analysis

We now consider the effect of perturbations of A and b on the least squares solution x. In this analysis the condition number of the matrix $A \in \mathbf{R}^{m\times n}$ will play a significant role. The following definition generalizes the condition number (6.6.3) of a square nonsingular matrix.

Definition 8.1.18.
Let $A \in \mathbf{R}^{m\times n}$ have rank $r > 0$ and singular values equal to $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. Then the condition number of A is

$$\kappa(A) = \|A\|_2\|A^\dagger\|_2 = \sigma_1/\sigma_r,$$

where the last equality follows from the relations $\|A\|_2 = \sigma_1$, $\|A^\dagger\|_2 = \sigma_r^{-1}$.

Using the singular value decomposition $A = U\Sigma V^T$ we obtain

$$A^TA = V\Sigma^T(U^TU)\Sigma V^T = V\begin{pmatrix}\Sigma_r^2 & 0\\ 0 & 0\end{pmatrix}V^T. \tag{8.1.23}$$

Hence, $\sigma_i(A^TA) = \sigma_i^2(A)$, and it follows that

$$\kappa(A^TA) = \kappa^2(A).$$

This shows that *the matrix of the normal equations has a condition number which is the square of the condition number of A.*

We now give a first order perturbation analysis for the least squares problem when $\operatorname{rank}(A) = n$. Denote the perturbed data $A+\delta A$ and $b+\delta b$ and assume that δA sufficiently small so that we have $\operatorname{rank}(A+\delta A) = n$. Let the perturbed solution be $x + \delta x$ and $r + \delta r$, where $r = b - Ax$ is the residual vector. Then, neglecting second order perturbations, we have

$$\delta r = \delta b - (A + \delta A)(x + \delta x) = (\delta b - \delta A\, x) - A\delta x.$$

The perturbed solution satisfies

$$(A + \delta A)^T\big((A + \delta A)(x + \delta x) - (b + \delta b)\big) = 0.$$

Subtracting $A^T(Ax - b) = 0$ and neglecting second order perturbations, we get

$$\delta x = (A^TA)^{-1}A^T(\delta b - \delta A\, x) + (A^TA)^{-1}\delta A^T r, \tag{8.1.24}$$
$$\delta r = (I - A(A^TA)^{-1}A^T)(\delta b - \delta Ax) - A(A^TA)^{-1}\delta A^T\, r, \tag{8.1.25}$$

Here we can identify

$$(A^TA)^{-1}A^T = A^\dagger, \qquad A(A^TA)^{-1} = (A^\dagger)^T,$$
$$I - A(A^TA)^{-1}A^T = I - AA^\dagger = P_{\mathcal{N}(A^T)}.$$

Using (8.1.9) and (8.1.23) it follows that

$$\|A^\dagger\|_2 = \|(A^\dagger)^T\|_2 = 1/\sigma_n, \qquad \|(A^TA)^{-1}\|_2 = 1/\sigma_n^2, \qquad \|P_{\mathcal{N}(A^T)}\|_2 = 1.$$

Hence, taking norms in (8.1.25) and (8.1.25) we obtain

$$\|\delta x\|_2 \lessapprox \frac{1}{\sigma_n}\|\delta b\|_2 + \frac{1}{\sigma_n}\|\delta A\|_2\left(\|x\|_2 + \frac{1}{\sigma_n}\|r\|_2\right), \tag{8.1.26}$$

$$\|\delta r\|_2 \lessapprox \|\delta b\|_2 + \|\delta A\|_2\left(\|x\|_2 + \frac{1}{\sigma_n}\|r\|_2\right), \tag{8.1.27}$$

A more refined perturbation analysis (see Wedin [65]) shows that if

$$\eta = \|A^\dagger\|_2\|\delta A\|_2 \ll 1.$$

then $\operatorname{rank}(A+\delta A) = n$, and there are perturbations δA and δb such that these upper bounds are almost attained.

Assuming that $x \neq 0$ and setting $\delta b = 0$, we get a bound for the normwise relative perturbation

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq \kappa(A)\frac{\|\delta A\|_2}{\|A\|_2}\Big(1 + \frac{\|r\|_2}{\sigma_n^2\|x\|_2}\Big) \tag{8.1.28}$$

Note that if the system $Ax = b$ is consistent, then $r = 0$ and the bound is identical to that obtained for a square nonsingular linear system. Otherwise, there is a second term present in the perturbation bound.

An upper bound for the condition number for x in the least squares problems with respect to A is

$$\kappa_{LS} = \kappa(A)\Big(1 + \frac{\|r\|_2}{\sigma_n\|x\|_2}\Big) \tag{8.1.29}$$

The two following facts should be noted:

- κ_{LS} depends not only on A but also on r and therefore on b;
- If $\|r\|_2 \ll \sigma_n\|x\|_2$ then $\kappa_{LS} \approx \kappa(A)$, but if $\|r\|_2 > \sigma_n\|x\|_2$ the second term in (8.1.29) will dominate,

Example 8.1.2. The following simple example illustrates the perturbation analysis above. Consider a least squares problem with

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \delta \\ 0 & 0 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 0 \\ \alpha \end{pmatrix}, \qquad \delta A = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \delta/2 \end{pmatrix}.$$

and $\kappa(A) = 1/\delta \gg 1$. If $\alpha = 1$ then

$$x = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \delta x = \frac{2}{5\delta}\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \qquad r = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \quad \delta r = -\frac{1}{5}\begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}.$$

For this right hand side $\|x\|_2 = \|r\|_2$ and $\kappa_{LS} = 1/\delta + 1/\delta^2 \approx \kappa^2(A)$. This is reflected in the size of δx.

If instead we take $\alpha = \delta$, then a short calculation shows that $\|r\|_2/\|x\|_2 = \delta$ and $\kappa_{LS} = 2/\delta$. The same perturbation δA now gives

$$\delta x = \frac{2}{5}\begin{pmatrix}0\\1\end{pmatrix}, \qquad \delta r = -\frac{\delta}{5}\begin{pmatrix}0\\2\\1\end{pmatrix}.$$

It should be stressed that in order for the perturbation analysis above to be useful, the matrix A and vector b should be scaled so that perturbations are "well defined" by bounds on $\|\delta A\|_2$ and $\|b\|_2$. If the columns in $A = (a_1, a_2, \ldots, a_n)$ have widely differing norms, then a much better estimate may often be obtained by applying (8.1.28) to the scaled problem $\min_{\tilde{x}} \|\tilde{A}\tilde{x} - b\|_2$, chosen so that $\tilde{A}$ has columns of unit length, i.e.,

$$\tilde{A} = AD^{-1}, \qquad \tilde{x} = Dx, \qquad D = \operatorname{diag}(\|a_1\|_2, \ldots, \|a_n\|_2).$$

By Theorem 8.2.5 this column scaling approximately minimizes $\kappa(AD^{-1})$ over $D > 0$. Note however that scaling the columns also changes the norm in which the error in x is measured.

If the *rows* in A differ widely in norm, then (8.1.28) may also considerably overestimate the perturbation in x. As remarked above, we cannot scale the rows in A without changing the least squares solution.

Perturbation bounds with better scaling properties can be obtained by considering component-wise perturbations.

$$|\delta A| \le \omega E, \qquad |\delta b| \le \omega f. \tag{8.1.30}$$

Substituting in (8.1.25)–(8.1.25) yields the bounds

$$|\delta x| \lessapprox \omega \left(|A^\dagger|(f + E|x|) + |(A^TA)^{-1}|E^T|r| \right), \tag{8.1.31}$$

$$|\delta r| \lessapprox \omega \left(|I - AA^\dagger|(f + E|x|) + |(A^\dagger)^T|E^T|r| \right). \tag{8.1.32}$$

where terms of order $O(\omega^2)$ have been neglected. In particular, if $E = |A|$, $f = |b|$, we obtain taking norms

$$\|\delta x\| \lessapprox \omega \left(\| \, |A^\dagger|(|b| + |A||x|)\| + \| \, |(A^TA)^{-1}||A|^T|r| \, \| \right), \tag{8.1.33}$$

$$\|\delta r\| \lessapprox \omega \left(\| \, |I - AA^\dagger|(|A||x| + |b|)\| + \| \, |(A^\dagger)^T||A|^T|r| \, \| \right). \tag{8.1.34}$$

8.1.6 Backward Error and Stability

An algorithm for solving the linear least squares problem is said to numerically stable if for any data A and b, there exist small perturbation matrices and vectors δA and δb, such that the computed solution $\bar{x}$ is the *exact* solution to

$$\min_x \|(A + \delta A)x - (b + \delta b)\|_2, \tag{8.1.35}$$

where $\|\delta A\| \leq \tau$, $\|\delta b\| \leq \tau$, with τ being a small multiple of the unit roundoff u. Methods which explicitly form the normal equations are not backward stable. However, many methods based on orthogonal factorizations have been proved to be numerically stable with $\delta b = 0$.

Any computed solution $\bar{x}$ is called a stable solution if it satisfies (8.1.35). This does not mean that $\bar{x}$ is close to the exact solution x. If the least squares problem is ill-conditioned then a stable solution can be very different from x. For a stable solution the error $\|x - \bar{x}\|$ can be estimated using the perturbation results given in Section 8.1.5.

Many special fast methods exist for solving structured least squares problems, e.g., where A is a Toeplitz matrix. These methods cannot be proved to be backward stable, which is one reason why a solution to the following problem is of interest:

Given an alleged solution $\tilde{x}$, find the smallest backward error, i.e. a perturbation δA of smallest norm such that $\tilde{x}$ is the exact solution to the perturbed problem

$$\min_x \|(b + \delta b) - (A + \delta A)x\|_2. \tag{8.1.36}$$

If we could find the backward error of smallest norm, this could be used to verify numerically the stability properties of an algorithm. There is not much loss in assuming that $\delta b = 0$ in (8.1.37). Then the optimal backward error in the Frobenius norm is

$$\eta_F(\tilde{x}) = \min\{\|\delta A\|_F \mid \tilde{x} \text{ solves } \min_x \|b - (A + \delta A)x\|_2\}. \tag{8.1.37}$$

This the optimal backward error can be found by characterizing the set of all backward perturbations and then finding an optimal bound, which minimizes the Frobenius norm.

Theorem 8.1.19. *Let $\tilde{x}$ be an alleged solution and $\tilde{r} = b - A\tilde{x} \neq 0$. The optimal backward error in the Frobenius norm is*

$$\eta_F(\tilde{x}) = \begin{cases} \|A^T\tilde{r}\|_2/\|\tilde{r}\|_2, & \text{if } \tilde{x} = 0, \\ \min\{\eta, \sigma_{min}([A \quad C])\} & \text{otherwise.} \end{cases} \tag{8.1.38}$$

where

$$\eta = \|\tilde{r}\|_2/\|\tilde{x}\|_2, \qquad C = I - (\tilde{r}\tilde{r}^T)/\|\tilde{r}\|_2^2$$

and $\sigma_{min}([A \quad C])$ denotes the smallest (nonzero) singular value of the matrix $[A \quad C] \in \mathbb{R}^{m\times(n+m)}$.

The task of computing $\eta_F(\tilde{x})$ is thus reduced to that of computing $\sigma_{min}(\mathcal{A})$. Since this is expensive, approximations that are accurate and less costly have been derived. If a QR factorization of A is available lower and upper bounds for $\eta_F(\tilde{x})$ can be computed in only $\mathcal{O}(mn)$ operations. Let $r_1 = P_{\mathcal{R}(A)}\tilde{r}$ be the orthogonal projection of $\tilde{r}$ onto the range of A. If $\|r_1\|_2 \leq \alpha\|r\|_2$ it holds that

$$\frac{\sqrt{5}-1}{2}\,\tilde{\sigma}_1 \leq \eta_F(\tilde{x}) \leq \sqrt{1+\alpha^2}\,\tilde{\sigma}_1, \tag{8.1.39}$$

where

$$\tilde{\sigma}_1 = \left\|(A^TA + \eta I)^{-1/2}A^T\tilde{r}\right\|_2 / \|\tilde{x}\|_2. \tag{8.1.40}$$

Since $\alpha \to 0$ for small perturbations $\tilde{\sigma}_1$ is an asymptotic upper bound.

Review Questions

1. State the Gauss–Markov theorem.

2. Assume that A has full column rank. Show that the matrix $P = A(A^TA)^{-1}A^T$ is symmetric and satisfies the condition $P^2 = P$.

3. (a) Give conditions for a matrix P to be the orthogonal projector onto a subspace $S \in \mathbf{R}^n$.

(b) Define the orthogonal complement of S in $\mathbf{R}^n$.

4. (a) Which are the four fundamental subspaces of a matrix? Which relations hold between them? Express the orthogonal projections onto the fundamental subspaces in terms of the SVD.

(b) Give two geometric conditions which are necessary and sufficient conditions for x to be the pseudo-inverse solution of $Ax = b$.

5. Which of the following relations are universally correct?

(a) $\mathcal{N}(B) \subseteq \mathcal{N}(AB)$. (b) $\mathcal{N}(A) \subseteq \mathcal{N}(AB)$. (c) $\mathcal{N}(AB) \subseteq \mathcal{N}(A)$.

(d) $\mathcal{R}(AB) \subseteq \mathcal{R}(B)$. (e) $\mathcal{R}(AB) \subseteq \mathcal{R}(A)$. (f) $\mathcal{R}(B) \subseteq \mathcal{R}(AB)$.

6. (a) What are the four Penrose conditions for X to be the pseudo-inverse of A?

(b)A matrix X is said to be a **left-inverse** if $XA = I$. Show that a left-inverse is an $\{1,2,3\}$-inverse, i.e. satisfies the Penrose conditions (1), (2), and (3). Similarly show that a **right-inverse** is an $\{1,2,4\}$-inverse.

7. Let the singular values of $A \in \mathbf{R}^{m\times n}$ be $\sigma_1 \geq \cdots \geq \sigma_n$. What relations are satisfied between these and the singular values of

$$\tilde{A} = (A, u), \qquad \hat{A} = \begin{pmatrix} A \\ v^T \end{pmatrix}?$$

8. (a) Show that $A^\dagger = A^{-1}$ when A is a nonsingular matrix.

(b) Construct an example where $G \neq A^\dagger$ despite the fact that $GA = I$.

Problems

1. (a) Compute the pseudo-inverse $x^\dagger$ of a column vector x.

(b) Take $A = (\,1 \quad 0\,)$, $B = (\,1 \quad 1\,)^T$, and show that $1 = (AB)^\dagger \neq B^\dagger A^\dagger = 1/2$.

2. (a) Verify that the Penrose conditions uniquely defines the matrix X. Do it first for $A = \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$, and then transform the result to a general matrix A.

3 (a) Show that if $w \in \mathbf{R}^n$ and $w^T w = 1$, then the matrix $P(w) = I - 2ww^T$ is both symmetric and orthogonal.

(b) Given two vectors $x, y \in \mathbf{R}^n$, $x \neq y$, $\|x\|_2 = \|y\|_2$, then

$$P(w)x = y, \qquad w = (y - x)/\|y - x\|_2.$$

4. Let $S \subseteq \mathbf{R}^n$ be a subspace, P_1 and P_2 be orthogonal projections onto $S = \mathcal{R}(P_1) = \mathcal{R}(P_2)$. Show that $P_1 = P_2$, i.e., the orthogonal projection onto S is unique.

Hint: Show that for any $z \in \mathbf{R}^n$

$$\|(P_1 - P_2)z\|_2^2 = (P_1 z)^T (I - P_2)z + (P_2 z)^T (I - P_1)z = 0.$$

5. (R. E. Cline) Let A and B be any matrices for which the product AB is defined, and set

$$B_1 = A^\dagger AB, \qquad A_1 = AB_1 B_1^\dagger.$$

Show that $AB = AB_1 = A_1 B_1$ and that $(AB)^\dagger = B_1^\dagger A_1^\dagger$.

Hint: Use the Penrose conditions.

6. (a) Show that the matrix $A \in \mathbf{R}^{m\times n}$ has a **left inverse** $A^L \in \mathbf{R}^{n\times m}$, i.e., $A^L A = I$, if and only if $\operatorname{rank}(A) = n$. Although in this case $Ax = b \in \mathcal{R}(A)$ has a unique solution, the left inverse is not unique. Find the general form of Σ^L and generalize the result to A^L.

(b) Discuss the **right inverse** A^R in a similar way.

7. Show that $A^\dagger$ minimizes $\|AX - I\|_F$.

8. Prove *Bjerhammar's characterization* : Let A have full column rank and let B be any matrix such that $A^T B = 0$ and $(\,A \quad B\,)$ is nonsingular. Then $A^\dagger = X^T$ where

$$\begin{pmatrix} X^T \\ Y^T \end{pmatrix} = (\,A \quad B\,)^{-1}.$$

8.2 The Method of Normal Equations

8.2.1 Characterization of Least Squares Solutions

We now show a necessary condition for a vector x to minimize $\|b - Ax\|_2$.

Theorem 8.2.1.

Given the matrix $A \in \mathbf{R}^{m\times n}$ and a vector $b \in \mathbf{R}^m$. The vector x minimizes $\|b - Ax\|_2$ if and only if the residual vector $r = b - Ax$ is orthogonal to $\mathcal{R}(A)$, or equivalently

$$A^T(b - Ax) = 0. \tag{8.2.1}$$

Proof. Let x be a vector for which $A^T(b - Ax) = 0$. Then for any $y \in \mathbf{R}^n$ $b - Ay = (b - Ax) + A(x - y)$. Squaring this and using (8.2.1) we obtain

$$\|b - Ay\|_2^2 = \|b - Ax\|_2^2 + \|A(x - y)\|_2^2 \geq \|b - Ax\|_2^2.$$

On the other hand assume that $A^T(b - Ax) = z \neq 0$. Then if $x - y = -\epsilon z$ we have for sufficiently small $\epsilon \neq 0$,

$$\|b - Ay\|_2^2 = \|b - Ax\|_2^2 - 2\epsilon\|z\|_2^2 + \epsilon^2\|Az\|_2^2 < \|b - Ax\|_2^2$$

so x does not minimize $\|b - Ax\|_2$. □

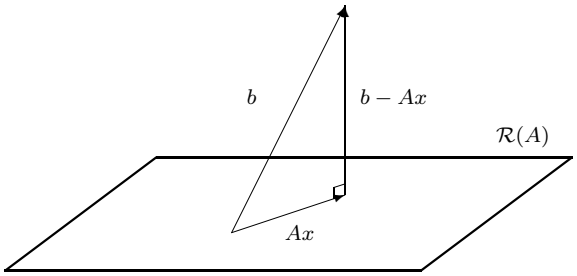


Figure 8.2.1. *Geometric characterization of the least squares solution.*

Here $A^TA \in \mathbf{R}^{n\times n}$ is a symmetric matrix and since

$$x^TA^TAx = \|Ax\|_2^2 \geq 0,$$

also positive semidefinite. The normal equations $A^TAx = A^Tb$ are *consistent* since

$$A^Tb \in \mathcal{R}(A^T) = \mathcal{R}(A^TA),$$

and therefore a least squares solution always exists.

By Theorem 8.2.1 any least squares solution x will decompose the right hand side b into two orthogonal components

$$b = Ax + r, \quad r \perp Ax. \tag{8.2.2}$$

Here $Ax = P_{\mathcal{R}(A)}b$ is the orthogonal projection (see Sec. 8.3.1) onto $\mathcal{R}(A)$ and $r \in \mathcal{N}(A^T)$ (cf. Fig. 8.2.1). Any solution to the (always consistent) normal equations (8.2.1) is a least squares solution. Note that although the least squares solution x may not be unique the decomposition in (8.2.2) always is unique.

Theorem 8.2.2.

The matrix A^TA is positive definite if and only if the columns of A are linearly independent, i.e., when $\operatorname{rank}(A) = n$. *In this case the least squares solution x is unique and given by*

$$x = (A^TA)^{-1}A^Tb. \tag{8.2.3}$$

Proof. If the columns of A are linearly independent, then $x \neq 0 \Rightarrow Ax \neq 0$. Therefore $x \neq 0 \Rightarrow x^TA^TAx = \|Ax\|_2^2 > 0$, and hence A^TA is positive definite. On

the other hand, if the columns are linearly dependent, then for some $x_0 \neq 0$ we have $Ax_0 = 0$. Then $x_0^T A^T A x_0 = 0$, and therefore $A^T A$ is not positive definite. When $A^T A$ is positive definite it is also nonsingular and (8.2.3) follows. □

For the minimum norm problem (8.1.2) let y be any solution of $A^T y = c$, and write $y = y_1 + y_2$, where $y_1 \in \mathcal{R}(A)$. $y_2 \in \mathcal{N}(A^T)$. Then $A^T y_2 = 0$ and hence y_1 is also a solution. Since $y_1 \perp y_2$ we have

$$\|y_1\|_2^2 = \|y\|_2^2 - \|y_2\|_2^2 \leq \|y\|_2^2,$$

with equality only if $y_2 = 0$. Hence the minimum norm solution lies in $\mathcal{R}(A)$ and we can write $y = Az$, for some z. Then we have $A^T y = A^T A z = c$. If A^T has linearly independent rows the inverse of $A^T A$ exists and the minimum norm solution $y \in \mathbb{R}^m$ satisfies the normal equations of second kind

$$y = A(A^T A)^{-1} c. \tag{8.2.4}$$

8.2.2 Forming and Solving the Normal Equations

From the time of Gauss until the computer age the basic computational tool for solving (8.1.1) was to form $A^T A$ and $A^T b$ and solve the normal equations by symmetric Gaussian elimination (which Gauss did), or later by the Cholesky factorization [7]. We now discuss the numerical implementation of this method. We defer treatment of rank deficient problems to later and assume throughout this section that rank $(A) = n$.

The first step is to compute the elements of the symmetric matrix $C = A^T A$ and the vector $d = A^T b$. If $A = (a_1, a_2, \ldots, a_n)$ has been partitioned by columns, we can use the inner product formulation

$$c_{jk} = (A^T A)_{jk} = a_j^T a_k, \qquad d_j = (A^T b)_j = a_j^T b, \quad 1 \leq j \leq k \leq n. \tag{8.2.5}$$

Since C is symmetric it is only necessary to compute and store its lower (or upper) triangular which requires $\frac{1}{2}mn(n+1)$ multiplications. Note that if $m \gg n$, then the number of elements $\frac{1}{2}n(n+1)$ in the upper triangular part of $A^T A$ is much smaller than the number mn of elements in A. Hence in this case the formation of $A^T A$ and $A^T b$ can be viewed as a *data compression*!

The formulas in (8.2.5) may not be suitable for large problems, where the matrix A is held in secondary storage, since each column needs to be accessed many times. An alternative row oriented outer product algorithm only needs *one pass* through the data (A, b). Denoting by $\tilde{a}_i^T$, the ith row of A, $i = 1, \ldots, m$, we have

$$C = A^T A = \sum_{i=1}^{m} \tilde{a}_i \tilde{a}_i^T, \qquad d = A^T b = \sum_{i=1}^{m} b_i \tilde{a}_i. \tag{8.2.6}$$

This is an form, where $A^T A$ is expressed as the sum of m matrices of rank one and $A^T b$ as a linear combination of the transposed rows of A. Using this alternative no

more storage is needed than that for A^TA and A^Tb. This outer product form is also preferable if the matrix A is sparse; see the hint to Problem 7.6.1. Note that both formulas can be combined if we adjoin b to A and form

$$(A,b)^T(A,b) = \begin{pmatrix} A^TA & A^Tb \\ b^TA & b^Tb \end{pmatrix}.$$

The matrix $C = A^TA$ is symmetric, and if $\operatorname{rank}(A) = n$ also positive definite. Gauss solved the normal equations by symmetric Gaussian elimination, but computing the Cholesky factorization

$$C = A^TA = R^TR, \quad R \in \mathbf{R}^{n\times n}, \tag{8.2.7}$$

is now the standard approach. The Cholesky factor R is upper triangular and nonsingular and can be computed by one of the algorithms given in Sec. 7.4.2. The least squares solution is then obtained by solving the two triangular systems

$$R^Tz = d, \qquad Rx = z. \tag{8.2.8}$$

Forming and solving the normal equations requires (neglecting lower order terms) about $\frac{1}{2}mn^2 + \frac{1}{6}n^3$ flops. If we have several right hand sides b_i, $i = 1:p$, then the Cholesky factorization need only be computed once. To solve for each new right hand side then only needs $mn + n^2$ additional flops.

Example 8.2.1.

Linear regression is the problem of fitting a linear model $y = \alpha + \beta x$ to a set of given points (x_i, y_i), $i = 1:m$. This leads to a overdetermined linear system

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

Forming the normal equations we get

$$\begin{pmatrix} m & \sum_{i=1}^m x_i \\ \sum_{i=1}^m x_i & \sum_{i=1}^m x_i^2 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^m y_i \\ \sum_{i=1}^m y_ix_i \end{pmatrix}. \tag{8.2.9}$$

Eliminating α we obtain the "classical" formulas

$$\beta = \Big(\sum_{i=1}^m y_ix_i - m\bar{y}\bar{x}\Big) \Big/ \Big(\sum_{i=1}^m x_i^2 - m\bar{x}^2\Big),$$

where

$$\bar{y} = \frac{1}{m}\sum_{i=1}^m y_i, \qquad \bar{x} = \frac{1}{m}\sum_{i=1}^m x_i. \tag{8.2.10}$$

are the mean values. The first equation in (8.2.9) gives

$$\bar{y} = \alpha + \beta\bar{x}. \tag{8.2.11}$$

which shows that $(\bar{y}, \bar{x})$ lies on the fitted line. This determines $\alpha = \bar{y} - \beta\bar{x}$.

A more accurate formula for β is obtained by first subtracting out the mean values from the data. We have

$$(y - \bar{y}) = \beta(x - \bar{x})$$

In the new variables the matrix of normal equation is *diagonal.* and we find

$$\beta = \textstyle\sum_{i=1}^{m}(y_i - \bar{y})(x_i - \bar{x})_i \Big/ \sum_{i=1}^{m}(x_i - \bar{x})^2. \tag{8.2.12}$$

A drawback of this formula is that it requires two passes through the data.

In many least squares problems the matrix A has the property that in each row all nonzero elements in A are contained in a narrow band. For banded rectangular matrix A we define:

Definition 8.2.3.

For $A \in \mathbf{R}^{m\times n}$ let f_i and l_i be the column subscripts of the first and last nonzero in the ith row of A, i.e.,

$$f_i = \min\{j \mid a_{ij} \neq 0\}, \qquad l_i = \max\{j \mid a_{ij} \neq 0\}. \tag{8.2.13}$$

Then the matrix A is said to have row bandwidth w, where

$$w = \max_{1\le i\le m} w_i, \qquad w_i = (l_i - f_i + 1). \tag{8.2.14}$$

Alternatively w is the smallest number for which it holds that

$$a_{ij}a_{ik} = 0, \quad \textit{if} \quad |j - k| \geq w. \tag{8.2.15}$$

For this structure to have practical significance we need to have $w \ll n$. Matrices of small row bandwidth often occur naturally, since they correspond to a situation where only variables "close" to each other are coupled by observations. We now prove a relation between the row bandwidth of the matrix A and the bandwidth of the corresponding matrix of normal equations A^TA.

Theorem 8.2.4.

Assume that the matrix $A \in \mathbf{R}^{m\times n}$ has row bandwidth w. Then the symmetric matrix A^TA has bandwidth $r \leq w - 1$.

Proof. From the Definition 8.2.3 it follows that $a_{ij}a_{ik} \neq 0 \Rightarrow |j - k| < w$. Hence,

$$|j - k| \geq w \Rightarrow (A^TA)_{jk} = \sum_{i=1}^{m} a_{ij}a_{ik} = 0.$$

□

If the matrix A also has full column rank it follows that we can use the band Cholesky Algorithm 6.4.6 to solve the normal equations.

The covariance matrix estimate in (8.1.4) can be expressed in terms of the Cholesky factor as

$$V = \sigma^2(A^TA)^{-1} = \sigma^2(R^TR)^{-1} = \sigma^2R^{-1}R^{-T}.$$

In order to assess the accuracy of the computed least squares estimate of x it is often required to compute the matrix V, or part of it. The matrix $S = R^{-1}$, which is also upper triangular, can be computed from the triangular system $RS = I$ by back-substitution. Often just the diagonal elements v_{ii} of $V = \sigma^2SS^T$ are required, which are the variances of the components of the least squares solution x. These elements are the 2-norms squared of the rows of S,

$$v_{ii} = \sigma^2 \sum_{j=i}^{n} s_{ij}^2, \quad i = 1, 2, \ldots, n.$$

In many situations the matrix V only occurs as an intermediate quantity in a formula. For example the variance of a linear functional $\varphi = f^T\hat{x}$ is equal to $\sigma^2 v$, where

$$v = f^TVf = f^TR^{-1}R^{-T}f = z^Tz, \quad z = R^{-T}f.$$

Thus to compute v we only need to solve the triangular system $R^Tz = f$ and form z^Tz. This is a more stable and efficient approach than using the expression f^TVf.

We have $r - \hat{r} = -A(A^TA)^{-1}A^T\epsilon$, where $\hat{r} = b - A\hat{x}$ is the least squares residual and ϵ the random error in the model. Hence $r - \hat{r}$ has covariance matrix

$$V_r = \sigma^2(A(A^TA)^{-1}A^T)^2 = \sigma^2A(A^TA)^{-1}A^T = \sigma^2P_{\mathcal{R}(A)}.$$

Note that the orthogonal projector $P_{\mathcal{R}(A)}$ can be computed from

$$P_{\mathcal{R}(A)} = A(R^TR)^{-1}A^T = QQ^T, \qquad Q = AR^{-1}.$$

The **normalized residuals** are defined by

$$\tilde{r} = (\operatorname{diag}(V_r))^{-1/2}\hat{r}.$$

Large components in $\tilde{r}$ can be assumed to correspond to "bad" data.

8.2.3 Stability and Accuracy with Normal Equations

We now turn to a discussion of the accuracy of the method of normal equations for least squares problems. First we consider rounding errors in the formation of the system of normal equations. Using the standard model for floating point computation we get for the elements $\bar{c}_{ij}$ in the computed matrix $\bar{C} = fl(A^TA)$

$$\bar{c}_{ij} = fl\Big(\sum_{k=1}^{m} a_{ik}a_{jk}\Big) = \sum_{k=1}^{m} a_{ik}a_{jk}(1 + \delta_k),$$

where (see (2.4.4)) $|\delta_k| < 1.06(m+2-k)u$ (u is the machine unit). It follows that the computed matrix satisfies

$$\bar{C} = A^T A + E, \qquad |e_{ij}| < 1.06um \sum_{k=1}^{m} |a_{ik}||a_{jk}|. \tag{8.2.16}$$

A similar estimate holds for the rounding errors in the computed vector $A^T b$. Note that it is *not* possible to show that $\bar{C} = (A+E)^T(A+E)$ for some small error matrix E, i.e., the rounding errors in forming the matrix $A^T A$ are not in general equivalent to small perturbations of the initial data matrix A. From this we can deduce that *the method of normal equations is not backwards stable.* The following example illustrates that when $A^T A$ is ill-conditioned, *it might be necessary to use double precision in forming and solving the normal equations in order to avoid loss of significant information.*

Example 8.2.2. (LÄUCHLI) Consider the system $Ax = b$, where

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \epsilon & & \\ & \epsilon & \\ & & \epsilon \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad |\epsilon| \ll 1.$$

We have, exactly

$$A^T A = \begin{pmatrix} 1+\epsilon^2 & 1 & 1 \\ 1 & 1+\epsilon^2 & 1 \\ 1 & 1 & 1+\epsilon^2 \end{pmatrix}, \qquad A^T b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},$$

$$x = \frac{1}{3+\epsilon^2}\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T, \qquad r = \frac{1}{3+\epsilon^2}\begin{pmatrix} \epsilon^2 & -1 & -1 & -1 \end{pmatrix}^T.$$

Now assume that $\epsilon = 10^{-4}$, and that we use eight-digit decimal floating point arithmetic. Then $1+\epsilon^2 = 1.00000001$ rounds to 1, and the computed matrix $A^T A$ will be singular. We have lost all information contained in the last three rows of A! Note that the residual in the first equation is $O(\epsilon^2)$ but $O(1)$ in the others.

Least squares problems of this form occur when the error in some equations (here $x_1 + x_2 + x_3 = 1$) have a much smaller variance than in the others; see Sec. 8.6.2.

To assess the error in the least squares solution $\bar{x}$ computed by the method of normal equations, we must also account for rounding errors in the Cholesky factorization and in solving the triangular systems. Using Theorem 6.6.6 and the perturbation bound in Theorem 6.6.2 it can be shown that provided that $2n^{3/2}u\kappa(A^T A) < 0.1$, the error in the computed solution $\bar{x}$ satisfies

$$\|\bar{x} - x\|_2 \leq 2.5n^{3/2}u\kappa(A^T A)\|x\|_2. \tag{8.2.17}$$

As seen in Sec. 8.1.5, for "small" residual least squares problem the true condition number is approximately $\kappa(A) = \kappa^{1/2}(A^T A)$. In this case *the system of normal*

equations can be much worse conditioned than the least squares problem from which it originated.

Sometimes ill-conditioning is caused by an unsuitable formulation of the problem. Then a different choice of parameterization can significantly reduce the condition number. For example, in approximation problems one should try to use orthogonal, or nearly orthogonal, base functions. In case the elements in A and b are the original data the ill-conditioning cannot be avoided in this way.

In statistics the linear least squares problem $\min_x \|b - Ax\|_2$ derives from a **multiple linear regression** problem, where the vector b is a response variable and the columns of A contain the values of the explanatory variables.

In Secs. 8.3 and 8.4 we consider methods for solving least squares problems based on orthogonalization. These methods work directly with A and b and are backwards stable.

8.2.4 Scaling Least Squares Problems

In Sec. 7.7.7 we discussed how the scaling of rows and columns of a linear system $Ax = b$ influenced the solution computed by Gaussian elimination. For a least squares problem $\min_x \|Ax - b\|_2$ a row scaling of (A, b) is not allowed since such a scaling would change the exact solution. However, we can scale the columns of A. If we take $x = Dx'$, the normal equations will change into

$$(AD)^T(AD)x' = D(A^TA)Dx' = DA^Tb.$$

Hence this corresponds to a *symmetric scaling* of rows and columns in A^TA. It is important to note that if the Cholesky algorithm is carried out without pivoting the computed solution is *not* affected by such a scaling, cf. Theorem 7.5.6. This means that even if no explicit scaling is carried out, the rounding error estimate (8.2.17) for the computed solution $\bar{x}$ holds for *all* D,

$$\|D(\bar{x} - x)\|_2 \leq 2.5n^{3/2}u\kappa(DA^TAD)\|Dx\|_2.$$

(Note, however, that scaling the columns changes the norm in which the error in x is measured.)

Denote the *minimum* condition number under a symmetric scaling with a positive diagonal matrix by

$$\kappa'(A^TA) = \min_{D>0} \kappa(DA^TAD). \tag{8.2.18}$$

The following result by van der Sluis [1969] shows the scaling where D is chosen so that in $D(A^TA)D$ all column norms are equal, i.e. $D = \text{diag}(\|a_1\|_2, \ldots, \|a_n\|_2)^{-1}$, comes within a factor of n of the minimum value.

Theorem 8.2.5. *Let $C \in \mathbf{R}^{n\times n}$ be a symmetric and positive definite matrix, and denote by $\mathcal{D}$ the set of $n \times n$ nonsingular diagonal matrices. Then if in C all diagonal elements are equal, and C has at most q nonzero elements in any row, it holds that*

$$\kappa(C) \leq q \min_{D\in\mathcal{D}} \kappa(DCD).$$

As the following example shows, this scaling can reduce the condition number considerably. In cases where the method of normal equations gives surprisingly accurate solution to a seemingly very ill-conditioned problem, the explanation often is that the condition number of the scaled problem is quite small!

Example 8.2.3. The matrix $A \in R^{21\times 6}$ with elements

$$a_{ij} = (i-1)^{j-1}, \qquad 1 \le i \le 21, \quad 1 \le j \le 6$$

arises when fitting a fifth degree polynomial $p(t) = x_0 + x_1 t + x_2 t^2 + \ldots + x_5 t^5$ to observations at points $x_i = 0, 1, \ldots, 20$. The condition numbers are

$$\kappa(A^T A) = 4.10 \cdot 10^{13}, \qquad \kappa(DA^T AD) = 4.93 \cdot 10^{6}.$$

where D is the column scaling in Theorem 8.2.5. Thus, the condition number of the matrix of normal equations is reduced by about seven orders of magnitude by this scaling!

A simple way to improve the accuracy of a solution $\bar{x}$ computed by the method of normal equations is by fixed precision iterative refinement, see Sec. 7.7.8. This requires that the data matrix A is saved and used to compute the residual vector $b - A\bar{x}$. In this way information lost when $A^T A$ was formed can be recovered. If also the corrections are computed from the normal equations we obtain the following algorithm:

Iterative Refinement with Normal Equations:

Set $x_1 = \bar{x}$, and for $s = 1, 2, \ldots$ until convergence do

$$\begin{aligned} r_s &:= b - Ax_s, \qquad R^T R \delta x_s = A^T r_s, \\ x_{s+1} &:= x_s + \delta x_s. \end{aligned}$$

Here R is computed by Cholesky factorization of the matrix of normal equation $A^T A$. This algorithm only requires one matrix-vector multiplication each with A and A^T and the solution of two triangular systems. Note that the first step, i.e., for $i = 0$, is identical to solving the normal equations. It can be shown that initially the errors will be reduced with rate of convergence equal to

$$\bar{\rho} = cu\kappa'(A^T A), \tag{8.2.19}$$

where c is a constant depending on the dimensions m, n. Several steps of the refinement may be needed to get good accuracy. (Note that $\bar{\rho}$ is proportional to $\kappa'(A^T A)$ even when no scaling of the normal equations has been performed!)

Example 8.2.4. If $\kappa'(A^T A) = \kappa(A^T A)$ and $c \approx 1$ the error will be reduced to a backward stable level in p steps if $\kappa^{1/2}(A^T A) \le u^{-p/(2p+1)}$. (As remarked before $\kappa^{1/2}(A^T A)$ is the condition number for a small residual problem.) For example, with $u = 10^{-16}$, the maximum value of $\kappa^{1/2}(A^T A)$ for different values of p are:

$$10^{5.3}, 10^{6.4}, 10^{8}, \qquad p = 1, 2, \infty.$$

For moderately ill-conditioned problems the normal equations combined with iterative refinement can give very good accuracy. For more ill-conditioned problems the methods based QR factorization described in Secs. 8.3 and 8.4 are usually to be preferred.

8.2.5 Methods Based on Gaussian Elimination

The pseudo-inverse of a matrix can also be computed using a LU factorization with complete pivoting. Usually it will be sufficient to use partial pivoting with a linear independence check. Let $\tilde{a}_{q,p+1}$ be the element of largest magnitude in column $p+1$. If $|\tilde{a}_{q,p+1}| < tol$, column $p+1$ is considered to be linearly dependent and is placed last. We then look for a pivot element in column $p+2$, etc.

Assume now that we have computed the LU factorization

$$\Pi_1 A \Pi_2 = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} (U_{11} \quad U_{12}), \tag{8.2.20}$$

where $L_{11}, U_{11} \in \mathbf{R}^{r\times r}$ are triangular and nonsingular. Then by Theorem 8.1.7 we have

$$\begin{aligned} A^\dagger &= \Pi_2 (U_{11} \quad U_{12})^\dagger \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix}^\dagger \Pi_1 \\ &= \Pi_2 (I_r \quad S)^\dagger U_{11}^{-1} L_{11}^{-1} \begin{pmatrix} I_r \\ T \end{pmatrix}^\dagger \Pi_1, \end{aligned}$$

where

$$T = L_{21} L_{11}^{-1}, \qquad S = U_{11}^{-1} U_{12},$$

Note the symmetry in the treatment of the L and U factors!

Standard algorithms for solving nonsymmetric linear systems $Ax = b$ are usually based on LU factorization with partial pivoting. Therefore it seems natural to consider such factorizations also for least squares problems which are only slightly overdetermined, i.e., where $m - n \ll n$.

A rectangular matrix $A \in \mathbf{R}^{m\times n}$, $m \geq n$, can be reduced by Gaussian elimination with partial pivoting to an upper triangular form U. In general, column interchanges are needed to ensure numerical stability. In the full rank case, $\operatorname{rank}(A) = n$, the resulting LDU factorization becomes

$$\Pi_1 A \Pi_2 = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = LDU = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} DU, \tag{8.2.21}$$

where $L_1 \in \mathbf{R}^{n\times n}$ is unit lower triangular, D diagonal, and $U \in \mathbf{R}^{n\times n}$ is unit upper triangular and nonsingular. Thus the matrix L has the same dimensions as A and a lower trapezoidal structure. Computing this factorization requires $\frac{1}{2}n^2(m - \frac{1}{3}n)$ flops.

Using the LU factorization (8.2.21) and setting $\tilde{x} = \Pi_2^T x$, $\tilde{b} = \Pi_1 b$, the least squares problem $\min_x \|Ax - b\|_2$ is reduced to

$$\min_y \|Ly - \tilde{b}\|_2, \qquad DU\tilde{x} = y. \tag{8.2.22}$$

If partial pivoting by rows is used in the factorization (8.2.21), then L is usually a well-conditioned matrix. In this case the solution to the least squares problem (8.2.22) can be computed from the normal equations

$$L^T L y = L^T \tilde{b},$$

without substantial loss of accuracy. This is the approach taken by Peters and Wilkinson [49, 1970]. The following example shows that this is a more stable method than using the normal equation $A^T Ax = A^T b$.

Example 8.2.5. (Noble [43, 1976])
Consider the matrix A and its pseudo-inverse

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1+\epsilon^{-1} \\ 1 & 1-\epsilon^{-1} \end{pmatrix}, \quad A^\dagger = \frac{1}{6}\begin{pmatrix} 2 & 2-3\epsilon^{-1} & 2+3\epsilon^{-1} \\ 0 & 3\epsilon^{-1} & -3\epsilon^{-1} \end{pmatrix}.$$

The (exact) matrix of normal equations is

$$A^T A = \begin{pmatrix} 3 & 3 \\ 3 & 3+2\epsilon^2 \end{pmatrix}.$$

If $\epsilon \leq \sqrt{u}$, then in floating point computation $fl(3+2\epsilon^2) = 3$, and the computed matrix $fl(A^T A)$ has rank one. However, the LU factorization of A is

$$A = LDU = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix},$$

where L and U are well-conditioned. The correct pseudo-inverse is now obtained from

$$A^\dagger = U^{-1}D^{-1}(L^T L)^{-1}L^T = \begin{pmatrix} 1 & -\epsilon \\ 0 & \epsilon \end{pmatrix} \begin{pmatrix} 1/3 & 0 \\ 0 & 1/2 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \end{pmatrix}.$$

and now there is no cancellation. □

Forming the symmetric matrix $L^T L$ requires $\frac{1}{2}n^2(m - \frac{2}{3}n)$ flops, and computing its Cholesky factorization takes $n^3/6$ flops. Hence, neglecting terms of order n^2, the total number of flops to compute the least squares solution by the Peters–Wilkinson method is $n^2(m - \frac{1}{3}n)$. This is always more expensive than the method of normal equations applied to $A^T A$.

When $m - n < n$ an algebraic reformulation is advantageous. If we let $T = L_2 L_1^{-1}$ and $L_1 y = z$, problem (8.2.22) becomes

$$\min_z \left\| \begin{pmatrix} I_n \\ T \end{pmatrix} z - \begin{pmatrix} \tilde{b}_1 \\ \tilde{b}_2 \end{pmatrix} \right\|_2 .$$

The solution z can be computed from

$$\begin{aligned} z &= (I_n + T^T T)^{-1}(\tilde{b}_1 + T^T \tilde{b}_2) \\ &= \tilde{b}_1 + (I_n + T^T T)^{-1} T^T (\tilde{b}_2 - T\tilde{b}_1) \\ &= \tilde{b}_1 + T^T (I_{m-n} + TT^T)^{-1}(\tilde{b}_2 - T\tilde{b}_1). \end{aligned} \tag{8.2.23}$$

The last expression can be evaluated more efficiently if $m - n < n$ and leads to the most efficient method for solving slightly overdetermined least squares problems. (Note that for $m = n + 1$ the inversion in (8.2.23) is reduced to a scalar division.)

Methods based on the factorization (8.2.21) for solving the minimum norm problem $\min \|y\|_2$, subject to $A^T y = c$ can be similarly developed. Setting $\tilde{c} = \Pi_2^T c$ and $\tilde{y} = \Pi_1 y$, we have

$$\tilde{y} = (U^T L^T)^\dagger \tilde{c} = L(L^T L)^{-1} U^{-T} \tilde{c}.$$

For the case $m - n < n$ we note that from $U^T L^T \tilde{y} = \tilde{c}$ we have

$$\tilde{y}_1 = L_1^{-T} U^{-T} \tilde{c} - (L_2 L_1^{-1})^T \tilde{y}_2 = e - T^T \tilde{y}_2. \tag{8.2.24}$$

Hence $\tilde{y}_2$ can be obtained as the solution to the least squares problem

$$\min_{\tilde{y}_2} \left\| \begin{pmatrix} T^T \\ I_{m-n} \end{pmatrix} \tilde{y}_2 - \begin{pmatrix} e \\ 0 \end{pmatrix} \right\|_2 ,$$

or using the normal equations,

$$\tilde{y}_2 = (I_{m-n} + TT^T)^{-1} Te. \tag{8.2.25}$$

The reformulation used above for the almost square case follows from a useful identity, which holds for any matrix S of dimension $r \times (n - r)$ of rank r:

$$(I_r + S^T S)^{-1} S^T = S^T (I_{n-r} + SS^T)^{-1}. \tag{8.2.26}$$

This identity is easily proved using the Woodbury formula (3.1.6). It reduces the computation of the pseudo-inverse of a matrix of rank r to the computation of the pseudo-inverse of a matrix of rank $(n - r)$. If $n - r \ll r$, there is a great gain in efficiency.

Review Questions

1. Give a necessary and sufficient condition for x to be a solution to $\min_x \|Ax - b\|_2$, and interpret this geometrically. When is is the least squares solution x unique? When is $r = b - Ax$ unique?

2. What are the advantages and drawbacks with the method of normal equations for computing the least squares solution of $Ax = b$? Give a simple example, which shows that loss of information can occur in forming the normal equations.

3. Discuss how the accuracy of the method of normal equations can be improved by (a) scaling the columns of A, (b) iterative refinement.

4. Show that the more accurate formula in Example 8.2.1 can be interpreted as a special case of the method (8.5.5)–(8.5.6) for partitioned least squares problems.

5. (a) Let $A \in \mathbf{R}^{m\times n}$ with $m < n$. Show that A^TA is singular.
(b) Show, using the SVD, that $\operatorname{rank}(A^TA) = \operatorname{rank}(AA^T) = \operatorname{rank}(A)$.

6. Define the condition number $\kappa(A)$ of a rectangular matrix A. What terms in the perturbation of a least squares solution depend on κ and κ^2, respectively?

Problems

1. In order to estimate the height above sea level for three points, A,B, and C, the difference in altitude was measured between these points and points D,E, and F at sea level. The measurements obtained form a linear system in the heights x_A, x_B, and x_C of A,B, and C,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ x_B \\ x_C \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 1 \end{pmatrix}.$$

Show that the least squares solution and residual vector are

$$x = \frac{1}{4}(5, 7, 12)^T, \qquad r = \frac{1}{4}(-1, 1, 0, 2, 3, -3)^T.$$

and verify that the residual vector is orthogonal to all columns in A.

2. (a) Consider the linear regression problem of fitting $y(t) = \alpha + \beta(t - c)$ by the method of least squares to the data

t	1	3	4	6	7
$f(t)$	-2.1	-0.9	-0.6	0.6	0.9

With the (unsuitable) choice $c = 1,000$ the normal equations

$$\begin{pmatrix} 5 & 4979 \\ 4979 & 4958111 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} -2.1 \\ -2097.3 \end{pmatrix}$$

become very ill-conditioned. Show that if the element 4958111 is rounded to $4958 \cdot 10^3$ then β is perturbed from its correct value 0.5053 to -0.1306!

(b) As shown in Example 8.2.1, a much better choice of base functions is shifting with the mean value of t, i.e., taking $c = 4.2$. However, it is not necessary to shift with the *exact* mean; Show that shifting with 4, the midpoint of the interval $(1, 7)$, leads to a very well-conditioned system of normal equations.

3. Denote by x_w the solution to the weighted least squares problem (8.1.6) and let x be the solution to the corresponding unweighted problem ($W = I$). Using the normal equations show that

$$x_w - x = (A^T W^{-1} A)^{-1} A^T (W^{-1} - I)(b - Ax). \tag{8.2.27}$$

Conclude that weighting the rows affects the solution if $b \notin \mathcal{R}(A)$.

4. Assume that $\operatorname{rank}(A) = n$, and put $\bar{A} = (A, b) \in \mathbf{R}^{m\times(n+1)}$. Let the corresponding cross product matrix, and its Cholesky factor be

$$\bar{C} = \bar{A}^T \bar{A} = \begin{pmatrix} C & d \\ d^T & b^T b \end{pmatrix}, \qquad \bar{R} = \begin{pmatrix} R & z \\ 0 & \rho \end{pmatrix}.$$

Show that the solution x and the residual norm ρ to the linear least squares problem $\min_x \|b - Ax\|_2$ is given by

$$Rx = z, \qquad \|b - Ax\|_2 = \rho.$$

5. Let $A \in \mathbf{R}^{m\times n}$ and $\operatorname{rank}(A) = n$. Show that the minimum norm solution of the underdetermined system $A^T y = c$ can be computed as follows:

(i) Form the matrix $A^T A$, and compute its Cholesky factorization $A^T A = R^T R$.

(ii) Solve the two triangular systems $R^T z = c$, $Rx = z$, and compute $y = Ax$.

6. Compute the solution x using the LDU factorization in Example 8.6.2. Compare with the exact solution given in Example 8.2.2.

7. (B. Noble 1976) Consider the matrix A and its generalized inverse

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 + \epsilon^{-1} \\ 1 & 1 - \epsilon^{-1} \end{pmatrix}.$$

(a) Show that The (exact) matrix of normal equations is

$$A^T A = \begin{pmatrix} 3 & 3 \\ 3 & 3 + 2\epsilon^2 \end{pmatrix}.$$

Hence if $\epsilon \leq \sqrt{u}$, then in floating point computation $fl(3 + 2\epsilon^2) = 3$, and the computed matrix $fl(A^T A)$ has rank one.

(b) An LU factorization of A is

$$A = LU = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & \epsilon \end{pmatrix}.$$

Show that here L is well-conditioned. and that the pseudo-inverse can be stably computed from $A^\dagger = U^{-1}(L^T L)^{-1} L^T$.

8. (S. M. Stiegler [60].) In 1793 the French decided to base the new metric system upon a unit, the meter, equal to one 10,000,000th part of the distance from the the north pole to the equator along a meridian arc through Paris. The following famous data obtained in a 1795 survey consist of four measured subsections of an arc from Dunkirk to Barcelona. For each subsection the length of the arc S (in modules), the degrees d of latitude and the latitude L of the midpoint (determined by the astronomical observations) are given.

Segment	Arc length S	latitude d	Midpoint L
Dunkirk to Pantheon	62472.59	2.18910°	49° 56′ 30″
Pantheon to Evaux	76145.74	2.66868°	47° 30′ 46″
Evaux to Carcassone	84424.55	2.96336°	44° 41′ 48″
Carcassone to Barcelona	52749.48	1.85266°	42° 17′ 20″

If the earth is ellipsoidal, then to a good approximation it holds

$$z + y\sin^2(L) = S/d,$$

where z and y are unknown parameters. The meridian quadrant then equals $M = 90(z + y/2)$ and the eccentricity is e is found from $1/e = 3(z/y + 1/2)$. Use least squares to determine z and y and then M and $1/e$.

9. Consider the least squares problem $\min_x \|Ax - b\|_2^2$, where A has full column rank. Partition the problem as

$$\min_{x_1,x_2} \left\| (A_1 \ A_2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - b \right\|_2^2.$$

By a geometric argument show that the solution can be obtained as follows. First compute x_2 as solution to the problem

$$\min_{x_2} \|P_{A_1}^{\perp}(A_2x_2 - b)\|_2^2,$$

where $P_{A_1}^{\perp} = I - P_{A_1}$ is the orthogonal projector onto $\mathcal{N}(A_1^T)$. Then compute x_2 as solution to the problem

$$\min_{x_1} \|A_1x_1 - (b - A_2x_2)\|_2^2.$$

10. Show that if $A, B \in \mathbf{R}^{m\times n}$ and rank $(B) \neq$ rank (A) then it is not possible to bound the difference between $A^{\dagger}$ and $B^{\dagger}$ in terms of the difference $B - A$. *Hint*: Use the following example. Let $\epsilon \neq 0$, $\sigma \neq 0$, take

$$A = \begin{pmatrix} \sigma & 0 \\ 0 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} \sigma & \epsilon \\ \epsilon & 0 \end{pmatrix},$$

and show that $\|B - A\|_2 = \epsilon$, $\|B^{\dagger} - A^{\dagger}\|_2 > 1/\epsilon$.

11. Show that for any matrix A it holds

$$A^\dagger = \lim_{\mu\to 0}(A^TA + \mu^2 I)^{-1}A^T = \lim_{\mu\to 0} A^T(AA^T + \mu^2 I)^{-1}. \tag{8.2.28}$$

12. (a) Let $A = (a_1, a_2)$, where $a_1^T a_2 = \cos\gamma$, $\|a_1\|_2 = \|a_2\|_2 = 1$. Hence γ is the angle between the vectors a_1 and a_2. Determine the singular values and right singular vectors v_1, v_2 of A by solving the eigenvalue problem for

$$A^TA = \begin{pmatrix} 1 & \cos\gamma \\ \cos\gamma & 1 \end{pmatrix}.$$

Then determine the left singular vectors u_1, u_2 from (7.1.33).
(b) Show that if $\gamma \ll 1$, then $\sigma_1 \approx \sqrt{2}$ and $\sigma_2 \approx \gamma/\sqrt{2}$ and

$$u_1 \approx (a_1 + a_2)/2, \qquad u_2 \approx (a_1 - a_2)/\gamma.$$

8.3 Methods using Orthogonal Factorizations

Orthogonality plays a key role in least squares problems. By Theorem 8.2.2, in the full rank case, $\operatorname{rank}(A) = n$, the residual $r = b - Ax$ can be written

$$r = P_{\mathcal{N}(A^T)}b, \qquad P_{\mathcal{N}(A^T)} = I - A(A^TA)^{-1}A^T, \tag{8.3.1}$$

which gives an expression for $P_{\mathcal{R}(A)}$, the orthogonal projector onto $\mathcal{R}(A)$, the range space of A. It follows that any solution to the consistent linear system

$$Ax = P_{\mathcal{R}(A)}b \tag{8.3.2}$$

is a least squares solution. In the next section we survey the theory of orthogonal and oblique projection.

8.3.1 Orthogonal and Oblique Projections

Recall that two vectors v and w in $\mathbf{R}^n$ are said to be **orthogonal** if $(v, w) = 0$. A set of vectors $v_1, \ldots, v_k$ in $\mathbf{R}^n$ is called orthogonal with respect to the Euclidian inner product if

$$v_i^T v_j = 0, \quad i \neq j,$$

and **orthonormal** if also $v_i^T v_i = 1, i = 1 : k$. An orthogonal set of vectors is linearly independent. More generally, a collection of subspaces $S_1, \ldots, S_k$ of $\mathbf{R}^n$ are mutually orthogonal if

$$x^Ty = 0, \quad \forall\, x \in S_i, \quad \forall\, y \in S_j, \quad i \neq j.$$

The **orthogonal complement** $S^\perp$ of a subspace $S \in \mathbf{R}^n$ is defined by

$$S^\perp = \{y \in \mathbf{R}^n |\ x^Ty = 0,\ x \in S\}.$$

Let $q_1, \ldots, q_k$ form an orthonormal basis for a subspace $S \subset \mathbf{R}^n$. Such a basis can always be extended to a full orthonormal basis $q_1, \ldots, q_n$ for $\mathbf{R}^n$, and then $S^\perp = \operatorname{span}\{q_{k+1}, \ldots, q_n\}$.

Let $q_1, \ldots, q_n \in \mathbf{R}^m$ be orthonormal Then the matrix $Q = (q_1, \ldots, q_n) \in \mathbf{R}^{m\times n}$, $m \geq n$, is called an **orthogonal matrix** and $Q^TQ = I_n$. If Q is square $(m = n)$ then $Q^{-1} = Q^T$, and hence also $QQ^T = I_n$. Further,

$$1 = \det(Q^TQ) = \det(Q^T)\det(Q) = (\det(Q))^2,$$

and it follows that $\det(Q) = \pm 1$.

In the complex case, $A = (a_{ij}) \in \mathbf{C}^{m\times n}$ the Hermitian inner product leads to modifications in the definition of symmetric and orthogonal matrices. Two vectors x and y in $\mathbf{C}^n$ are called orthogonal if $x^Hy = 0$. A square matrix U for which $U^HU = I$ is called **unitary**. Then

$$(Ux)^HUy = x^HU^HUy = x^Hy,$$

and hence unitary matrices have the property that they preserve the Hermitian inner product. In particular the Euclidian length of a vector is invariant under unitary transformations, i.e., $\|Ux\|_2^2 = \|x\|_2^2$. Note that when the vectors and matrices are real the definitions for the complex case are consistent with those made for the real case.

Any square matrix $P \in \mathbf{R}^{n\times n}$ such that

$$P^2 = P. \tag{8.3.3}$$

is called **idempotent** and a **projector**. An arbitrary vector $v \in \mathbf{R}^n$ can be decomposed in a unique way as

$$v = Pv + (I - P)v = v_1 + v_2. \tag{8.3.4}$$

Here $v_1 = Pv \in S$ is a projection of v onto $\mathcal{R}(P)$, the range space of P. Since $Pv_2 = (P - P^2)v = 0$ it follows that $(I - P)$ is a projection onto $\mathcal{N}(P)$, the null space of P.

If P is symmetric, $P^T = P$, then

$$v_1^Tv_2 = (Pv)^T(I - P)v = v^TP(I - P)v = v^T(P - P^2)v = 0.$$

It follows that $v_2 \perp S$, i.e., v_2 lies in the orthogonal complement $S^\perp$ of S; In this case P is the **orthogonal projector** onto S and $I - P$ the orthogonal projector onto $S^\perp$. It can be shown that the orthogonal projector P onto a given subspace S is unique, see Problem 1.

Example 8.3.1.

Let $Q = (q_1, \ldots, q_n) \in \mathbf{R}^{m\times n}$, $m \geq n$, where $q_1, \ldots, q_n \in \mathbf{R}^m$ are orthonormal vectors. Then the orthogonal projector onto the orthogonal complement of $\mathcal{R}(Q)$.

$$P = I_m - QQ^T, \tag{8.3.5}$$

If $n = 1$ then $P = I_m - q_1 q_1^T$ and the null space $\mathcal{N}(P) = \text{span}\,(q_1)$ has dimension one. P is then called an **elementary orthogonal projection**.

A projector P such that $P \neq P^T$ is called an **oblique projector**. We now briefly review oblique projections and their matrix representations. If λ is an eigenvalue of P then from $P^2 = P$ it follows that $\lambda^2 = \lambda$. Hence the eigenvalues of P are either 1 or 0 and we can write the eigendecomposition

$$P = (U_1\, U_2) \begin{pmatrix} I_k & 0 \\ 0 & 0_{n-k} \end{pmatrix} \begin{pmatrix} \hat{Y}_1^T \\ \hat{Y}_2^T \end{pmatrix}, \quad \begin{pmatrix} \hat{Y}_1^T \\ \hat{Y}_2^T \end{pmatrix} = (U_1\, U_2)^{-1}, \tag{8.3.6}$$

where $k = \text{trace}\,(P)$ is the rank of P and

$$\text{span}\,(U_1) = \mathcal{R}(P), \qquad \text{span}\,(U_2) = \mathcal{N}(P).$$

The matrices $U_1 \in \mathbf{R}^{n\times n_1}$ and $U_2 \in \mathbf{R}^{n\times n_2}$ $(n_1 + n_2 = n)$, can be chosen as orthogonal bases for the invariant subspaces corresponding to the eigenvalues 1 or 0, respectively. In terms of this eigendecomposition (8.3.4) can be written

$$v = (U_1\ U_2) \begin{pmatrix} \hat{Y}_1^T \\ \hat{Y}_2^T \end{pmatrix} v = (U_1 \hat{Y}_1^T)v + (U_2 \hat{Y}_2^T)v = v_1 + v_2, \tag{8.3.7}$$

that is

$$P = U_1 \hat{Y}_1^T, \qquad I - P = U_2 \hat{Y}_2^T. \tag{8.3.8}$$

If $P^T = P$ then P is an orthogonal projector and in (8.3.6) we can take $U = (U_1\, U_2)$ orthogonal and $\hat{Y}_1 = U_1$ and $\hat{Y}_2 = U_2$. The projectors (8.3.7) then take the form

$$P = U_1 U_1^T, \qquad I - P = U_2 U_2^T; \tag{8.3.9}$$

For an orthogonal projector we have

$$\|Pv\|_2 = \|U_1^T v\|_2 \leq \|v\|_2 \ \ \forall \ \ v \in \mathbf{R}^m, \tag{8.3.10}$$

where equality holds for all vectors in $\mathcal{R}(U_1)$. From this it follows that for an orthogonal projector $\|P\|_2 = 1$. The conversion is also true; P is an orthogonal projection only if (8.3.10) holds.

When P is not symmetric we call $v_1 = Pv$ the **oblique projection** of v onto $\mathcal{R}(U_1)$ along $\mathcal{R}(U_2)$, and the matrix $P = U_1 \hat{Y}_1^T$ is the corresponding oblique projector. Similarly $I - P = U_2 \hat{Y}_2^T$ is the oblique projector onto $\mathcal{R}(U_2)$ along $\mathcal{R}(U_1)$.

From (8.3.6) we have

$$\begin{pmatrix} \hat{Y}_1^T \\ \hat{Y}_2^T \end{pmatrix} (U_1\ U_2) = \begin{pmatrix} \hat{Y}_1^T U_1 & \hat{Y}_1^T U_2 \\ \hat{Y}_2^T U_1 & \hat{Y}_2^T U_2 \end{pmatrix} = \begin{pmatrix} I_k & 0 \\ 0 & I_{n-k} \end{pmatrix}. \tag{8.3.11}$$

In particular we have $\hat{Y}_1^T U_2 = 0$ and $\hat{Y}_2^T U_1 = 0$. Hence the columns of $\hat{Y}_1$ form a basis of the orthogonal complement of $\mathcal{R}(U_2)$ and, similarly, the columns of $\hat{Y}_2$ form a basis of the orthogonal complement of $\mathcal{R}(U_1)$.

Let Y_1 be an orthogonal matrix whose columns span $\mathcal{R}(\hat{Y}_1)$. Then there is a nonsingular matrix G_1 such that $\hat{Y}_1 = Y_1 G$. From (8.3.11) it follows that $G^T Y_1^T U_1 = I_k$, and hence $G^T = (Y_1^T U_1)^{-1}$. Similarly $Y_2 = (Y_2^T U_2)^{-1} Y_2$ is an orthogonal matrix whose columns span $\mathcal{R}(\hat{Y}_2)$. Hence using (8.3.8) the projectors can be written

$$P = U_1 (Y_1^T U_1)^{-1} Y_1^T, \qquad I - P = U_2 (Y_2^T U_2)^{-1} Y_2^T. \tag{8.3.12}$$

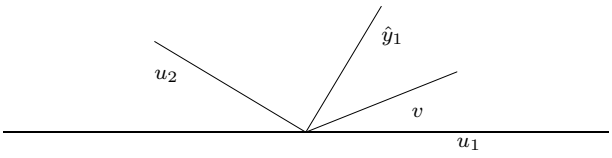


Figure 8.3.1. *The oblique projection of v on u_1 along u_2.*

Example 8.3.2.

We illustrate the case when $n = 2$ and $n_1 = 1$. Let the vectors u_1 and y_1 be normalized so that $\|u_1\|_2 = \|y_1\|_2 = 1$ and let $y_1^T u_1 = \cos\theta$, where θ is the angle between u_1 and y_1, see Fig. 8.4.1. Since

$$P = u_1 (y_1^T u_1)^{-1} y_1^T = \frac{1}{\cos\theta} u_1 y_1^T.$$

Hence $\|P\|_2 = 1/\cos\theta \geq 1$, and $\|P\|_2$ becomes very large when y_1 is almost orthogonal to u_1. When $y_1 = u_1$ we have $\theta = 0$ and P is an orthogonal projection.

8.3.2 Gram–Schmidt Orthogonalization

Gram–Schmidt orthogonalization is one of the fundamental algorithms in numerical linear algebra. Given a sequence of linearly independent vectors $a_1, a_2, \ldots, a_n$ Gram–Schmidt orthogonalization computes orthonormal vectors $q_1, q_2, \ldots, q_n$. such that

$$\text{span}\,[a_1, \ldots, a_k] = \text{span}\,[q_1, \ldots, q_k], \quad k = 1 : n. \tag{8.3.13}$$

Algorithm 8.3.1 Classical Gram–Schmidt (CGS).

for $k = 1 : n$

(i) If $k = 1$ then set $\hat{q}_1 = a_1$ else orthogonalize a_k against $q_1, \ldots, q_{k-1}$:

$$\hat{q}_k = a_k - \sum_{i=1}^{k-1} r_{ik} q_i, \qquad r_{ik} = q_i^T a_k, \quad i = 1 : k-1; \tag{8.3.14}$$

(ii) Normalize $\hat{q}_k$

$$r_{kk} = \|\hat{q}_k\|_2, \qquad q_k = \hat{q}_k / r_{kk}. \tag{8.3.15}$$

end;

Note that $\hat{q}_k \neq 0$, since otherwise a_k is a linear combination of the vectors $a_1, \ldots, a_{k-1}$, which contradicts the assumption. The CGS algorithm requires approximately mn^2 multiplications and can be interpreted in matrix terms as follows:

Theorem 8.3.1. *The QR Factorization*

Let the matrix $A = (a_1, a_2, \ldots, a_n) \in \mathbf{R}^{m\times n}$ have linearly independent columns. Then the Gram–Schmidt algorithm computes unique matrices $Q_1 \in \mathbf{R}^{m\times n}$ with orthonormal columns and an upper triangular $R \in \mathbf{R}^{n\times n}$ with positive diagonal elements, such that

$$A = (a_1, a_2, \ldots, a_n) = (q_1, q_2, \ldots, q_n) \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{2n} \end{pmatrix} \equiv Q_1 R. \quad (8.3.16)$$

Proof. Combining (8.3.14) and (8.3.15) we obtain

$$a_k = r_{kk} q_k + \sum_{i=1}^{k-1} r_{ik} q_i = \sum_{i=1}^{k} r_{ik} q_i, \quad k = 1 : n,$$

which is equivalent with (8.3.16). Since the vectors q_k are mutually orthogonal by construction the theorem follows. □

Corollary 8.3.2. *The factor R in the factorization* (8.3.16) *equals the Cholesky factor of $A^T A$. Hence the GS algorithm computes the Cholesky factor directly from A.*

Proof. The Cholesky factor R of a nonsingular matrix $A^T A$ is uniquely determined provided R is normalized to have a positive diagonal. From (8.3.16) we have $A^T A = R^T Q_1^T Q_1 R = R^T R$, and the result follows. □

For the *numerical* GS factorization of a matrix A a small reordering of the above algorithm gives the **modified Gram–Schmidt** method (MGS). Although mathematically equivalent to the classical algorithm MGS has greatly superior numerical properties, and is therefore usually to be preferred.

The modified Gram–Schmidt (MGS) algorithm employs a sequence of elementary orthogonal projections. At the beginning of step k, we have computed

$$(q_1, \ldots, q_{k-1}, a_k^{(k)}, \ldots, a_n^{(k)}),$$

where we have put $a_j = a_j^{(1)}$, $j = 1 : n$. Here $a_k^{(k)}, \ldots, a_n^{(k)}$ have already been made orthogonal to $q_1, \ldots, q_{k-1}$, which are final columns in Q_1. In the kth step q_k is obtained by normalizing the vector $a_k^{(k)}$,

$$\tilde{q}_k = a_k^{(k)}, \qquad r_{kk} = \|\tilde{q}_k\|_2, \qquad q_k = \tilde{q}_k / r_{kk}, \quad (8.3.17)$$

and then $a_{k+1}^{(k)}, \ldots, a_n^{(k)}$ are orthogonalized against q_k

$$a_j^{(k+1)} = (I_m - q_k q_k^T) a_j^{(k)} = a_j^{(k)} - r_{kj} q_k, \quad r_{kj} = q_k^T a_j^{(k)}, \quad j = k+1 : n. \tag{8.3.18}$$

After n steps we have obtained the factorization (8.3.16). Note that for $n = 2$ MGS and CGS are identical.

Algorithm 8.3.2 Modified Gram–Schmidt.
Given $A \in R^{m \times n}$ with $\text{rank}(A) = n$ the following algorithm computes the factorization $A = Q_1 R$:

```
for k = 1 : n
    q̂_k = a_k^(k);   r_kk = ‖q̂_k‖_2;
    q_k = q̂_k / r_kk;
    for j = k + 1 : n
        r_kj = q_k^T a_j^(k);
        a_j^(k+1) = a_j^(k) − r_kj q_k;
    end
end
```

The operations in Algorithm 8.3.2 can be sequenced so that the elements in R are computed in a column-wise fashion. However, the row-wise version given above is more suitable if column pivoting is to be performed; see Sec. 8.4.2.

There is also a **square root free** version of the modified Gram–Schmidt orthogonalization method, which results if the normalization of the vectors $\tilde{q}_k$ is omitted. In this version one computes scaled factors $\tilde{Q}_1 = (\tilde{q}_1, \ldots, \tilde{q}_n)$ and $\tilde{R}$ so that

$$A = \tilde{Q}_1 \tilde{R},$$

where $\tilde{R}$ is **unit** upper triangular. We take $\tilde{r}_{kk} = 1$, $d_k = \tilde{q}_k^T \tilde{q}_k$, and change (8.3.18) to

$$a_j^{(k+1)} = a_j^{(k)} - \tilde{r}_{kj} \tilde{q}_k, \qquad \tilde{r}_{kj} = \tilde{q}_k^T a_j^{(k)} / d_k, \qquad j = k+1, \ldots, n. \tag{8.3.19}$$

The unnormalized vector $\tilde{q}_k$ is just the orthogonal projection of a_k onto the complement of $\text{span}[a_1, a_2, \ldots, a_{k-1}] = \text{span}[q_1, q_2, \ldots, q_{k-1}]$.

In CGS the orthogonalization of a_k in step (8.3.14) can be written

$$\hat{q}_k = (I - Q_{k-1} Q_{k-1}^T) a_k, \quad Q_{k-1} = (q_1, \ldots, q_{k-1}).$$

In MGS *the projections $r_{ik} q_i$ are subtracted from a_k as soon as they are computed*, which corresponds to computing

$$\hat{q}_k = (I - q_{k-1} q_{k-1}^T) \cdots (I - q_1 q_1^T) a_k.$$

For $k > 2$ these two expressions are identical only if the $q_1, \ldots, q_{k-1}$ are accurately orthogonal. However, due to round-off there will be a gradual (sometimes catastrophic) loss of orthogonality. In this respect CGS and MGS behave very differently.

In MGS the loss of orthogonality occurs in a predictable manner proportional to the $\kappa(A)$. This is not the case for CGS.

Loss of orthogonality will occur in orthogonalization whenever cancellation takes place in subtracting the orthogonal projection on q_i from $a_k^{(i)}$, that is when

$$a_j^{(k+1)} = (I - q_k q_k^T) a_j^{(k)}, \qquad \|a_k^{(i+1)}\|_2 \ll \alpha \|a_k^{(i)}\|_2. \tag{8.3.20}$$

Consider the case of orthogonalizing *two* vectors. Given a vector a_2, we want to orthogonalize it against a vector q_1, $\|q_1\|_2 = 1$, by computing

$$\hat{q}_2 = a_2 - r_{12} q_1, \quad r_{12} = q_1^T a_2. \tag{8.3.21}$$

We use the standard model for floating point computation, and the basic results in Sec. 2.3.2 to analyze the rounding errors. For the *computed* scalar product $\bar{r}_{12} = fl(q_1^T a_2)$ we get

$$|\bar{r}_{12} - r_{12}| < \gamma_m \|a_2\|_2, \quad \gamma_m = \frac{mu}{1 - mu/2},$$

where u is the unit roundoff. Using $|r_{12}| \leq \|a_2\|_2$ we obtain for $\bar{q}_2 = fl(a_2 - fl(\bar{r}_{12} q_1))$

$$\|\bar{q}_2 - \hat{q}_2\|_2 < \gamma_{m+2} \|a_2\|_2.$$

Since $q_1^T \hat{q}_2 = 0$, it follows that $|q_1^T \bar{q}_2| < \gamma_{m+2} \|a_2\|_2$ and the loss of orthogonality

$$\frac{|q_1^T \bar{q}_2|}{\|\bar{q}_2\|_2} \approx \frac{|q_1^T \bar{q}_2|}{\|\hat{q}_2\|_2} < \gamma_{m+2} \frac{\|a_2\|_2}{\|\tilde{q}_2\|_2} = \frac{\gamma_{m+2}}{\sin \phi(q_1, a_2)}, \tag{8.3.22}$$

is proportional to $\phi(q_1, a_2)$, the angle between q_1 and a_2.

Example 8.3.3. As an illustration consider the matrix

$$A = (a_1, a_2) = \begin{pmatrix} 1.2969 & 0.8648 \\ 0.2161 & 0.1441 \end{pmatrix}.$$

Using the Gram–Schmidt algorithm and IEEE double precision we get

$$q_1 = \begin{pmatrix} 0.98640009002732 \\ 0.16436198585466 \end{pmatrix},$$

$r_{12} = q_1^T a_2 = 0.87672336001729$,

$$\hat{q}_2 = a_2 - r_{12} q_1 = \begin{pmatrix} -0.12501091273265 \\ 0.75023914025696 \end{pmatrix} 10^{-8},$$

$$q_2 = \begin{pmatrix} -0.16436196071471 \\ 0.98640009421635 \end{pmatrix},$$

and

$$R = \begin{pmatrix} 1.31478090189963 & 0.87672336001729 \\ 0 & 0.00000000760583 \end{pmatrix}.$$

Severe cancellation has taken place when computing $\hat{q}_2$, which leads to a serious loss of orthogonality between q_1 and q_2:

$$q_1^T q_2 = 2.5486557 \cdot 10^{-8},$$

which should be compared with the unit roundoff $1.11 \cdot 10^{-16}$. We note that the loss of orthogonality is roughly equal to a factor 10^{-8}.

Reorthogonalizing the computed vector $a_2^{(2)}$ against q_1 we obtain

$$q_1^T q_2 = 2.5486557 \cdot 10^{-8}, \quad \tilde{q}_2 = \begin{pmatrix} -0.16436198585466 \\ 0.98640009002732 \end{pmatrix}.$$

The vector $\tilde{q}_2$ is exactly orthogonal to q_1.

For MGS the loss of orthogonality can be bounded in terms of the condition number $\kappa(A)$ also for $n > 2$. (Note that for $n = 2$ MGS and CGS are the same.) In can be shown that if $c_2 \kappa u < 1$, then

$$\|I - \bar{Q}_1^T \bar{Q}_1\|_2 \leq \frac{c_1}{1 - c_2 \kappa u} \kappa u.$$

where c_1 and c_2 denote constants depending on m, n, and the details of the arithmetic. In contrast, the computed vectors q_k from CGS may depart from orthogonality to an almost arbitrary extent. The more gradual loss of orthogonality in the computed vectors q_i for MGS is illustrated in the example below; see also Problem 1.

Example 8.3.4. A matrix $A \in \mathbf{R}^{50 \times 10}$ was generated by computing

$$A = U \mathrm{diag}\,(1, 10^{-1}, \ldots, 10^{-9}) V^T$$

where U and V are orthonormal matrices. Hence A has singular values $\sigma_i = 10^{-i+1}$, $i = 1 : 10$, and $\kappa(A) = 10^9$. Fig. 8.5.1 shows the condition number of $A_k = (a_1, \ldots, a_k)$ and the loss of orthogonality in CGS and MGS after k steps as measured by $\|I_k - Q_k^T Q_k\|_2$.

For MGS the loss of orthogonality is more gradual and proportional to $\kappa(A_k)$, whereas for CGS the loss of orthogonality is roughly proportional to $\kappa^2(A_k)$,

In some applications it is important not only that the computed $\bar{Q}_1$ and $\bar{R}$ are such that $\bar{Q}_1 \bar{R}$ accurately represents A, but also that $\bar{Q}_1$ is accurately orthogonal. We call this the **orthogonal basis problem**. It can be show that for MGS it holds that

$$A + E = \bar{Q}_1 \bar{R}, \quad \|E\|_2 \leq c_0 u \|A\|_2.$$

However, to satisfy the second condition it is necessary to **reorthogonalize** the computed vectors in the Gram–Schmidt algorithm, whenever (8.3.20) is satisfied for some suitably chosen parameter $\alpha < 1$ typically chosen in the range $[0.1, 1/\sqrt{2}]$. In a sense to be made more precise below, *one reorthogonalization will always suffice*. Hence reorthogonalization will at most double the cost of the Gram–Schmidt factorization.

Table 8.3.1. *Loss of orthogonality and CGS and MGS.*

k	$\kappa(A_k)$	$\|I_k - Q_C^T Q_C\|_2$	$\|I_k - Q_M^T Q_M\|_2$
1	1.000e+00	1.110e-16	1.110e-16
2	1.335e+01	2.880e-16	2.880e-16
3	1.676e+02	7.295e-15	8.108e-15
4	1.126e+03	2.835e-13	4.411e-14
5	4.853e+05	1.973e-09	2.911e-11
6	5.070e+05	5.951e-08	3.087e-11
7	1.713e+06	2.002e-07	1.084e-10
8	1.158e+07	1.682e-04	6.367e-10
9	1.013e+08	3.330e-02	8.779e-09
10	1.000e+09	5.446e-01	4.563e-08

For the case $n = 2$ the following result is known:

Algorithm 8.3.3 Kahan–Parlett algorithm Parlett [48, Sec. 6.9].

Suppose that for any given z the expression $\bar{p} := \text{orthog}(a_1, z)$ computes an approximation to the (exact) orthogonal complement $p = z - a_1(a_1^T z)/\|a_1\|_2^2$ of z to a_1, such that the error satisfies $\|\bar{p} - p\|_2 \leq \epsilon\|z\|_2$ for some tiny positive ϵ. Then, given $A = (a_1, a_2)$, the following algorithm computes a vector $\bar{q}_2$, which satisfies

$$\|\bar{q}_2 - q_2\|_2 \leq (1+\alpha)\epsilon\|a_2\|_2, \qquad \|a_1^T \bar{q}_2\| \leq \epsilon\alpha^{-1}\|\bar{q}_2\|_2\|a_1\|_2, \tag{8.3.23}$$

where q_2 is the exact complement of a_2 orthogonal to a_1. The first inequality implies that $\bar{q}_2$ is close to a linear combination of a_1 and a_2. The second says that $\bar{q}_2$ is nearly orthogonal to a_1.

$\bar{q}_2 := \text{orthog}(a_1, a_2)$;
if $\|\bar{q}_2\|_2 < \alpha\|a_2\|_2$
  $\check{q}_2 := \text{orthog}(a_1, \bar{q}_2)$; (reorthogonalize$\bar{q}_2$)
  if $\|\check{q}_2\|_2 \geq \alpha\|\bar{q}_2\|_2$ $\check{q}_2$;
  else $\bar{q}_2 = \bar{q}_2 := 0$; (numerically singular case)
  end
end

Note that if $\|\check{q}_2\|_2 < \alpha\|\bar{q}_2\|_2$ we conclude that the given vectors (a_1, a_2) are linearly dependent and and signal this by setting $\bar{q}_2 := 0$.
When α is large, say $\alpha \geq 1/\sqrt{2}$, then the bounds in (8.3.23) are very good but reorthogonalization will occur more frequently. If α is small, reorthogonalization will be rarer, but the bound on orthogonality less good. For larger n there seems to be a good case for recommending the stringent value $\alpha = 1/\sqrt{2}$ or *always* perform one step of reorthogonalization ($\alpha = 1$).

Now consider the case $n > 2$. Assume we are given a matrix $Q_1 = (q_1, \ldots, q_{k-1})$

with $\|q_1\|_2 = \ldots = \|q_{k-1}\|_2 = 1$. Adding the new vector a_k, we want to compute a vector $\hat{q}_k$ such that

$$\hat{q}_k \in \text{span}(Q_1, a_k) \perp \text{span}(Q_1).$$

The solution equals $\hat{q}_k = a_k - Q_1 r_k$, where r_k solves the least squares problem

$$\min_{r_k} \|a_k - Q_1 r_k\|_2.$$

We first assume that Q_1 is accurately orthogonal. Then it can be rigorously proved that it suffices to run MGS twice on the matrix (Q_1, a_k). This generalizes the result by Kahan–Parlett to $n > 2$.

To solve the problem, when the columns of Q_1 are not accurately orthogonal, we can use **iterated** Gram–Schmidt methods. In the iterated CGS algorithm we put $\hat{q}_k^{(0)} := a_k$, $r_k^{(0)} := 0$, and for $p = 0, 1, \ldots$ compute

$$s_k^{(p)} := Q_1^T \hat{q}_k^{(p)}, \quad \hat{q}_k^{(p+1)} := \hat{q}_k^{(p)} - Q_1 s_k^{(p)}, \quad r_k^{(p+1)} := r_k^{(p)} + s_k^{(p)}.$$

The first step of this algorithm is the usual CGS algorithm, and each step is a reorthogonalization. The iterated MGS algorithm is similar, except that each projection is subtracted as soon as it computed: As in the Kahan–Parlett algorithm, the iterations can be stopped when $\|\hat{q}_k^{(p+1)}\|_2 > \alpha \|\hat{q}_k^{p}\|_2$.

The iterated Gram–Schmidt algorithm can be used recursively, adding one column a_k at a time, to compute the factorization $A = Q_1 R$. If A has full numerical column rank, then with $\alpha = 1/\sqrt{2}$ both iterated CGS and MGS computes a factor Q_1, which is orthogonal to almost full working precision, *using at most one reorthogonalization.* Hence in this case iterated CGS is *not* inferior to the iterated MGS.

8.3.3 Least Squares Problems by Gram–Schmidt

We now consider the use of the Modified Gram–Schmidt algorithm for solving linear least squares problem. It is important to note that because of the loss of orthogonality in Q_1 *computing* x *by forming* $c_1 = Q_1^T b$ *and then solving* $Rx = c_1$ *will not in general give an accurate solution.* Using the MGS factorization in this way seems to have contributed to an undeserved bad reputation of the method. Used correctly, as described below, the MGS factorization will give as accurate results as any competing method.

To solve a least squares problems the MGS algorithm is applied to the augmented matrix (A, b). If we skip the normalization of the $(n+1)$st column we obtain a factorization

$$(A,\, b) = (Q_1,\, r) \begin{pmatrix} R & z \\ 0 & 1 \end{pmatrix}, \tag{8.3.24}$$

where r is the residual vector. We have

$$\|Ax - b\|_2 = \left\| (A, b) \begin{pmatrix} x \\ -1 \end{pmatrix} \right\|_2 = \|Q_1(Rx - z) - r\|_2.$$

Let us assume that $q_{n+1} = r/\|r\|_2$ is orthogonal to Q_1. Then the minimum of the last expression occurs when $Rx - z = 0$ and the least squares residual equals r. Although this assumptions is not true to machine precision, we note that it is not necessary to assume that Q_1 is accurately orthogonal for the conclusion to hold. This heuristic argument leads to the following algorithm for solving linear least squares problems by MGS, which can be proved to be backward stable for computing the solution x:

Algorithm 8.3.4 Linear Least Squares Solution by MGS.
Carry out MGS on $A \in R^{m\times n}$, $\text{rank}(A) = n$, to give $Q_1 = (q_1, \ldots, q_n)$ and R, and put $b^{(1)} = b$. Compute the vector $z = (z_1, \ldots, z_n)^T$ by

$$\begin{aligned}
&\text{for } k = 1, 2, \ldots, n \\
&\quad z_k = q_k^T b^{(k)}; \quad b^{(k+1)} = b^{(k)} - z_k q_k; \\
&\text{end} \\
&r = b^{(n+1)}; \\
&\text{solve } Rx = z;
\end{aligned}$$

If implemented as above MGS gives very accurate results. Unfortunately, a common error can still found in some textbooks. This is to compute R by MGS, but in the final step solve $Rx = Q_1^T b$. *This destroys the accuracy and may be one reason the MGS method is not widely used.*

In some applications it is important to use an algorithm which is backwards stable for the computed residual $\bar{r}$, i.e. we want a relation

$$(A + E)^T \bar{r} = 0, \quad \|E\|_2 \le cu\|A\|_2, \tag{8.3.25}$$

to hold for some constant c. This implies that $A^T\bar{r} = -E^T\bar{r}$, and

$$\|A^T\bar{r}\|_2 \le cu\|\bar{r}\|_2\|A\|_2. \tag{8.3.26}$$

Note that this is much better than if we compute

$$\bar{r} = fl(b - fl(Ax)) = fl\left(\begin{pmatrix} b & A \end{pmatrix}\begin{pmatrix} 1 \\ -x \end{pmatrix}\right)$$

even when x is the *exact* least squares solution. We obtain using (2.3.13) and $A^T r = 0$

$$|A^T\bar{r}| < \gamma_{n+1}|A^T|(|b| + |A||x|).$$

From this we get the norm-wise bound

$$\|A^T\bar{r}\|_2 \le n^{1/2}\gamma_{n+1}\|A\|_2(\|b\|_2 + n^{1/2}\|A\|_2\|x\|_2),$$

which is a much weaker than (8.3.26) when, as is often the case, $\|\bar{r}\|_2 \ll \|b\|_2$!

An obvious remedy seems to be to reorthogonalize the computed residual r against $Q_1 = (q_1, q_2, \ldots, q_n)$. However, to obtain a backward stable algorithm for r this should be done in *reverse order*!

Algorithm 8.3.5 Orthogonal projection by MGS.
To make Algorithm 8.3.3 backward stable for r it suffices to add a loop where the vector $b^{(n+1)}$ is orthogonalized against $q_n, q_{n-1}, \ldots, q_1$ (*note the order*):

$$\begin{aligned}
&\text{for } k = n, n-1, \ldots, 1\\
&\quad z_k = q_k^T b^{(k+1)}; \quad b^{(k)} = b^{(k+1)} - z_k q_k;\\
&\text{end}\\
&r = b^{(1)};
\end{aligned}$$

It can be proved that this step "magically" compensates for the lack of orthogonality of Q_1 and the $\bar{r}$ computed by Algorithm 8.3.3 satisfies (8.3.25).

A similar idea is used to construct a backward stable algorithm for the minimum norm problem

$$\min \|y\|_2, \qquad A^T y = c.$$

Algorithm 8.3.6 Minimum Norm Solution by MGS.
Carry out MGS on $A^T \in R^{m\times n}$, with $\operatorname{rank}(A) = n$ to give $Q_1 = (q_1, \ldots, q_n)$ and R. Then the minimum norm solution $y = y^{(0)}$ is obtained from

$$\begin{aligned}
&R^T(\zeta_1, \ldots, \zeta_n)^T = c;\\
&y^{(n)} = 0;\\
&\text{for } k = n, \ldots, 2, 1\\
&\quad \omega_k = q_k^T y^{(k)}; \quad y^{(k-1)} = y^{(k)} - (\omega_k - \zeta_k) q_k;\\
&\text{end}
\end{aligned}$$

If the columns of Q_1 were orthogonal to working accuracy, then $\omega_k = 0$, $k = m, \ldots, 1$. Hence ω compensates for the lack of orthogonality to make this algorithm backwards stable!

8.3.4 Householder and Givens Transformations

Orthogonal matrices which are equal to the unit matrix modified by a matrix of rank one are called **elementary orthogonal matrices**. Such matrices are flexible and useful tools for constructing algorithms for solving a variety of problems in linear algebra. They are attractive since multiplication of a vector with an orthogonal matrix preserves the Euclidean length and hence there use leads to numerically stable algorithms.

Recall that a square matrix $Q \in \mathbf{R}^{m\times m}$, is called orthogonal if $Q^T Q = I$. Then $Q^{-1} = Q^T$, and hence $QQ^T = I$. Taking the determinant of both sides

$$\det(Q^T Q) = \det(Q^T)\det(Q) = \det(Q)^2 = 1.$$

and hence $\det(Q) = \pm 1$. A very important class of orthogonal transformations are matrices of the form

$$H = I - \beta uu^T, \quad \beta = 2/(u^T u). \tag{8.3.27}$$

By construction H is symmetric $H^T = H$, and using (8.3.27) we have

$$H^T H = H^2 = I - 2\beta uu^T + \beta^2 u(u^T u)u^T = I.$$

Hence H is orthogonal, and $H^2 = I$. The product Ha where a is a given vector can be computed without explicitly forming H itself using

$$Ha = (I - \beta uu^T)a = a - \beta u(u^T a).$$

Note that $Ha \in \text{span}[a, u]$. We have $Hu = -u$, i.e., H reverses u. Further $Ha = a$, for $a \perp u$. Hence H has $m-1$ eigenvalues equal to $+1$ and one equal to -1, and thus $\det(H) = -1$. The effect of the transformation Ha for a general vector a is to reflect a in the $(m-1)$ dimensional hyperplane characterized by the normal vector u, see Fig. 8.5.1. Therefore, H is called an **elementary reflector**. The use of elementary reflectors in numerical linear algebra was initiated by A. S. Householder. Matrices of the form (8.3.27) are therefore often called **Householder reflectors** and the vector u is called a **Householder vector**.

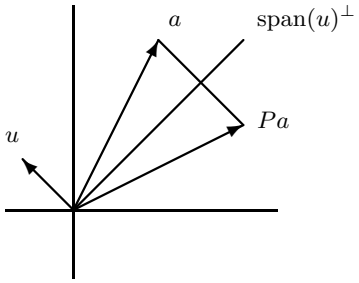


Figure 8.3.2.

Fig. 8.4.1 shows the vector a mapped into Ha by a reflection in the plane with normal vector u. Note that this is equivalent to subtracting twice the orthogonal projection onto u. Further the normal u is parallel to the difference $(a - Pa)$. Given $a \neq 0 \in \mathbf{R}^m$, we consider the problem of constructing a plane reflection $H \in \mathbf{R}^{m\times m}$ such that *multiplication by H zeros all components except the first in a*, i.e.,

$$Ha = \pm\sigma e_1, \quad \sigma = \|a\|_2. \tag{8.3.28}$$

Multiplying (8.3.28) from the left by H and using $H^2 = I$ it follows that $y = Ue_1$ satisfies $U^T y = e_1$ or

$$He_1 = \pm a/\sigma.$$

Hence (8.3.28) is equivalent to finding a square orthogonal matrix H with its first column proportional to $\pm a/\sigma$. It is easily seen that (8.3.28) is satisfied if we take

$$u = a \mp \sigma e_1 = \begin{pmatrix} \alpha_1 \mp \sigma \\ a_2 \end{pmatrix}, \quad a = \begin{pmatrix} \alpha_1 \\ a_2 \end{pmatrix}. \tag{8.3.29}$$

Note that u differs from a only in its first component. A short calculation shows that

$$1/\beta = \tfrac{1}{2}u^T u = \tfrac{1}{2}(a \mp \sigma e_1)^T(a \mp \sigma e_1) = \tfrac{1}{2}(\sigma^2 \mp 2\sigma\alpha_1 + \sigma^2) = \sigma(\sigma \mp \alpha_1).$$

If a is close to a multiple of e_1, then $\sigma \approx |\alpha_1|$ and cancellation may lead to a large relative error in β. To avoid this we take

$$u = a + \operatorname{sign}(\alpha_1)\sigma e_1, \qquad 1/\beta = \sigma(\sigma + |\alpha_1|), \tag{8.3.30}$$

which gives

$$Ha = -\operatorname{sign}(\alpha_1)\sigma e_1 = \hat{\sigma} e_1.$$

Note that with this choice of sign the vector $a = e_1$ will be mapped onto $-e_1$. (It is possible to rewrite the formula in (8.3.30) for β so that the other choice of sign does not give rise to numerical cancellation; for details see Parlett [48, pp. 91].)

The Householder transformation in (8.3.27) does not depend on the scaling of u. It is often more convenient to scale u so that its first component equals 1. If we write

$$H = I - \beta u u^T, \quad u = \begin{pmatrix} 1 \\ u_2 \end{pmatrix},$$

then

$$\beta = 1 + |\alpha_1|/\sigma. \qquad u_2 = \rho a_2, \quad \rho = \operatorname{sign}(\alpha_1)/(\sigma + |\alpha_1|), \tag{8.3.31}$$

This has the advantage that we can stably reconstruct β from u_2 using

$$\beta = 2/(u^T u) = 2/(1 + u_2^T u_2).$$

Algorithm 8.3.7 Let $a \in \mathbf{R}^m$ be a vector with $\|a\|_2 = \sigma$ and $a^T e_1 = \alpha_1$. The following algorithm constructs a Householder transformation $H = I - \beta u u^T$, where $u^T e_1 = 1$, such that $Ha = -sign(\alpha_1)\hat{\sigma} e_1$, where $\hat{\sigma} = -sign(\alpha_1)\sigma$.

$$\begin{aligned}
&[u, \beta, \hat{\sigma}] = \text{house}(a)\\
&\quad \alpha_1 = a(1);\\
&\quad \sigma = \|a\|_2;\\
&\quad \beta = 1 + |\alpha_1|/\sigma;\\
&\quad \hat{\sigma} = -\operatorname{sign}(\alpha_1)\sigma;\\
&\quad \rho = -1/(\hat{\sigma}\beta);\\
&\quad u = [1; \rho \cdot a(2:m)];
\end{aligned}$$

If a matrix $A = (a_1, \ldots, a_n) \in \mathbf{R}^{m\times n}$ is *premultiplied* by H the product can be computed in $2mn$ multiplications as

$$HA = (Ha_1, \ldots, Ha_n), \qquad Ha_j = a_j - \beta(u^T a_j)u. \tag{8.3.32}$$

An analogous formula, exists for *postmultiplying* A with H, where H now acts on the *rows* of A. Writing the products HA and AH as

$$HA = A - \beta u(u^T A), \qquad AH = A - \beta(Au)u^T,$$

shows that in both cases is A altered by a matrix of rank one.

Another useful class of orthogonal transformations are the matrices representing **plane rotations**, which are also called **Givens rotations** after Wallace Givens, who popularized their use for numerical computations. In $\mathbf{R}^2$ the matrix representing a rotation clockwise through an angle θ is

$$G(\theta) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \qquad c = \cos\theta, \quad s = \sin\theta. \tag{8.3.33}$$

Note that $G^{-1}(\theta) = G(-\theta)$, and $\det G(\theta) = +1$.

In $\mathbf{R}^m$ the matrix representing a rotation in the plane spanned by the unit vectors e_i and e_j, $i < j$, is the following rank two modification of the unit matrix I_m

$$G_{ij}(\theta) = \begin{array}{c} \\ \\ i \\ \\ j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccc} & & i & & j & & \end{array} \\ \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & & s & & \\ & & & \ddots & & & \\ & & -s & & c & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix} \end{array}. \tag{8.3.34}$$

Premultiplying a vector $a = (\alpha_1, \ldots, \alpha_m)^T$ by $G_{ij}(\theta)$ we get

$$G_{ij}(\theta)a = (\tilde{\alpha}_1, \ldots, \tilde{\alpha}_m)^T, \qquad \tilde{\alpha}_k = \begin{cases} \alpha_k, & k \neq i, j; \\ c\alpha_i + s\alpha_j, & k = i; \\ -s\alpha_i + c\alpha_j, & k = j. \end{cases} \tag{8.3.35}$$

Thus a plane rotation may be multiplied into a vector at a cost of two additions and four multiplications. We can determine the rotation $G_{ij}(\theta)$ so that $\tilde{\alpha}_j$ becomes zero by taking

$$c = \alpha_i/\sigma, \qquad s = \alpha_j/\sigma, \qquad \sigma = (\alpha_i^2 + \alpha_j^2)^{1/2} \neq 0. \tag{8.3.36}$$

Note that $-G(\theta)$ also zeros $\tilde{\alpha}_j$ so c and s are only determined up to a factor ± 1.

To guard against possible overflow, the Givens rotation should be computed as in the following procedure:

Algorithm 8.3.8 Given $(\alpha, \beta)^T \neq 0$ the algorithm constructs c, s, σ such that $s^2 + c^2 = 1$ and

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \end{pmatrix}:$$

$$
\begin{aligned}
&[c, s, \sigma] = \text{givrot}(\alpha, \beta) \\
&\quad \text{if } |\alpha| > |\beta| \\
&\qquad t = \beta/\alpha; \quad c = 1/\sqrt{1+t^2}; \\
&\qquad s = tc; \quad \sigma = \alpha/c; \\
&\quad \text{else} \\
&\qquad t = \alpha/\beta; \quad s = 1/\sqrt{1+t^2}; \\
&\qquad c = ts; \quad \sigma = \beta/s; \\
&\quad \text{end}
\end{aligned}
$$

Premultiplication of a matrix $A \in R^{m\times n}$ with a Givens rotation G_{ij} will only affect the two *rows* i and j in A, which are transformed according to

$$a_{ik} := ca_{ik} + sa_{jk}, \tag{8.3.37}$$

$$a_{jk} := -sa_{ik} + ca_{jk}, \quad k = 1, 2, \ldots, n. \tag{8.3.38}$$

The product requires $4n$ multiplications. An analogous algorithm, which only affects *columns* i and j, exists for postmultiplying A with G_{ij}.

Givens rotations can be used in several different ways to construct an orthogonal matrix U, which satisfies (8.3.28). Let G_{1k}, $k = 2, \ldots, m$ be a sequence of Givens rotations, where G_{1k} is determined to zero the kth component in the vector a,

$$G_{1m} \ldots G_{13} G_{12} a = \sigma e_1.$$

Note that G_{1k} will not destroy previously introduced zeros. Another possible sequence is $G_{k-1,k}$, $k = m, m-1, \ldots, 2$, where $G_{k-1,k}$ is chosen to zero the kth component. This demonstrates the flexibility of Givens rotations compared to reflectors.

It is essential to note that the matrix G_{ij} is never explicitly formed, but represented by (i, j) and the two numbers c and s. When a large number of rotations need to be stored it is more economical to store just a single number, from which c and s can be retrieved in a numerically stable way. Since the formula $\sqrt{1-x^2}$ is poor if $|x|$ is close to unity a slightly more complicated method than storing just c or s is needed. In a scheme devised by G. W. Stewart one stores the number c or s of smallest magnitude. To distinguish between the two cases one stores the reciprocal of c. More precisely, if $c \neq 0$ we store

$$\rho = \begin{cases} s, & \text{if } |s| < |c|; \\ 1/c, & \text{if } |c| \leq |s| \end{cases}.$$

In case $c = 0$ we put $\rho = 1$, a value that cannot appear otherwise.

To reconstruct the Givens rotation, if $\rho = 1$, we take $s = 1$, $c = 0$, and

$$\rho = \begin{cases} s = \rho, & c = \sqrt{1-s^2}, & \text{if } |\rho| < 1; \\ c = 1/\rho, & s = \sqrt{1-c^2}, & \text{if } |\rho| > 1; \end{cases}$$

It is possible to rearrange the Givens rotations so that it uses only two instead of four multiplications per element and no square root. These modified transformations called "fast" Givens transformations, and are described in Golub and Van Loan [29, 1996, Sec. 5.1.13].

8.3.5 Householder QR Factorization

Methods for solving the linear least squares problem which, like the SVD, are based on orthogonal transformations avoid the squaring of the condition number that results from forming the normal equations. In this section we first develop algorithms using elementary orthogonal transformations to factor a matrix $A \in \mathbf{R}^{m\times n}$ $(m \geq n)$ into the product of a *square* orthogonal matrix $Q \in \mathbf{R}^{m\times m}$ and an upper triangular matrix $R \in \mathbf{R}^{m\times n}$. We then show how to use this **full QR factorization** for solving linear least squares problems.

Theorem 8.3.3. *The Full QR Factorization*

Let $A \in \mathbf{R}^{m\times n}$ with $\operatorname{rank}(A) = n$. Then there is an orthogonal matrix $Q \in \mathbf{R}^{m\times m}$ and an upper triangular matrix R with positive diagonal elements such that

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{8.3.39}$$

Proof. A constructive proof will be given in Sec. 8.4.3. □

Since Q is orthogonal the singular values of R equal those of A and $\kappa(R) = \kappa(A)$. Indeed, to compute the SVD of A one can first compute the QR factorization and then the SVD of R.

The QR factorization can be written

$$A = (Q_1, Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R. \tag{8.3.40}$$

where Q has been partitioned as $Q = (Q_1, Q_2)$, $Q_1 \in \mathbf{R}^{m\times n}$, $Q_2 \in \mathbf{R}^{m\times(m-n)}$. This is the the factorization computed by the Gram–Schmidt algorithm. From (8.3.40) it follows that the columns of Q_1 and Q_2 form orthonormal bases for the range space of A and its orthogonal complement,

$$\mathcal{R}(A) = \mathcal{R}(Q_1), \qquad \mathcal{N}(A^T) = \mathcal{R}(Q_2), \tag{8.3.41}$$

and the corresponding orthogonal projections are

$$P_{\mathcal{R}(A)} = Q_1 Q_1^T, \qquad P_{\mathcal{N}(A^T)} = Q_2 Q_2^T. \tag{8.3.42}$$

Note that although the matrix Q_1 in (8.3.40) is uniquely determined, Q_2 can be any orthogonal matrix with range $\mathcal{N}(A^T)$.

In contrast to the Gram–Schmidt algorithm for computing the QR factorization, the methods we now consider represents Q *implicitly* as a product of Householder or Givens matrices. This elegantly avoids the problem with loss of orthogonality in Q!

The QR factorization of a matrix $A \in \mathbf{R}^{m\times n}$ of rank n can be computed using a sequence of n Householder reflectors. Let $A = (a_1, a_2, \ldots, a_n)$, $\sigma_1 = \|a_1\|_2$, and choose $H_1 = I - \beta_1 u_1 u_1^T$, so that

$$H_1 a_1 = H_1 \begin{pmatrix} \alpha_1 \\ \hat{a}_1 \end{pmatrix} = \begin{pmatrix} r_{11} \\ 0 \end{pmatrix}, \qquad r_{11} = -\text{sign}\,(\alpha_1)\sigma_1.$$

By (8.3.30) we achieve this by choosing $\beta_1 = 1 + |\alpha_1|/\sigma_1$,

$$u_1 = \begin{pmatrix} 1 \\ \hat{u_1} \end{pmatrix}, \qquad \hat{u_1} = \text{sign}\,(\alpha_1)\hat{a}_1/\rho_1, \qquad \rho_1 = \sigma_1\beta_1.$$

H_1 is then applied to the remaining columns $a_2, \ldots, a_n$, giving

$$A^{(2)} = H_1 A = \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1n} \\ 0 & \tilde{a}_{22} & \ldots & \tilde{a}_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & \tilde{a}_{n2} & \ldots & \tilde{a}_{nn} \end{pmatrix}.$$

Here the first column has the desired form and, as indicated by the notation, the first row is the final first row in R. In the next step the $(m-1)\times(n-1)$ block in the lower right corner is transformed. All remaining steps, $k = 2, \ldots, n$ are similar to the first. Before the kth step we have computed a matrix of the form

$$A^{(k)} = \begin{matrix} \\ {\scriptstyle k-1} \end{matrix} \begin{matrix} \overset{k-1}{} \\ \end{matrix} \begin{pmatrix} R_{11}^{(k)} & R_{12}^{(k)} \\ 0 & \hat{A}^{(k)} \end{pmatrix}, \tag{8.3.43}$$

where the first $k-1$ rows of $A^{(k)}$ are rows in the final matrix R, and $R_{11}^{(k)}$ is upper triangular. In step k the matrix $a^{(k)}$ is transformed,

$$A^{(k+1)} = H_k A^{(k)}, \qquad H_k = \begin{pmatrix} I_k & 0 \\ 0 & \tilde{H}_k \end{pmatrix}. \tag{8.3.44}$$

Here $\tilde{H}_k = I - \beta_k u_k u_k^T$ is chosen to zero the elements below the main diagonal in the first column of the submatrix

$$\hat{A}^{(k)} = (a_k^{(k)}, \ldots, a_n^{(k)}) \in \mathbf{R}^{(m-k+1)\times(n-k+1)},$$

i.e. $\tilde{H}_k a_k^{(k)} = r_{kk} e_1$. With $\sigma_k = \|a_k^{(k)}\|_2$, using (8.3.29), we get $r_{kk} = -\text{sign}\,(a_{kk}^{(k)})\sigma_k$, and

$$\hat{u}_k = \text{sign}\,(\alpha_k^{(k)})\hat{a}_k^{(k)}/\rho_k, \qquad \beta_k = 1 + |a_{kk}|/\sigma_k. \tag{8.3.45}$$

where $\rho_k = \sigma_k\beta_k$. After n steps we have obtained the QR factorization of A, where

$$R = R_{11}^{(n+1)}, \qquad Q = H_1 H_2 \cdots H_n. \tag{8.3.46}$$

Note that the diagonal elements r_{kk} will be positive if $a_k^{(kk)}$ is negative and negative otherwise. Negative diagonal elements may be removed by multiplying the corresponding rows of R and columns of Q by -1. [2]

[2]The difference between the Householder and Gram–Schmidt QR algorithms has been aptly summarized by Trefethen, who calls Gram–Schmidt triangular orthogonalization as opposed to Householder which is orthogonal triangularization.

Algorithm 8.3.9 Householder QR Factorization.
Given a matrix $A^{(1)} = A \in \mathbf{R}^{m\times n}$ of rank n, the following algorithm computes R and Householder matrices:

$$H_k = \text{diag}\,(I_{k-1}, \tilde{H}_k), \qquad \tilde{H}_k = I - \beta_k u_k u_k^T, \quad k = 1, 2, \ldots, n, \tag{8.3.47}$$

so that $Q = H_1 H_2 \cdots H_n$.

```
for k = 1, 2, ..., n
    [u_k, β_k, r_kk] = house(a_k^(k));
    for j = k + 1, ..., n
        γ_jk = β_k u_k^T a_j^(k);
        r_kj = a_kj^(k) − γ_jk;
        a_j^(k+1) = â_j^(k) − γ_jk û_k;
    end
end
```

The vectors $\hat{u}_k$ can overwrite the elements in the strictly lower trapezoidal part of A. Thus, all information associated with the factors Q and R can be overwritten A. The vector $(\beta_1, \ldots, \beta_n)$ of length n can be recomputed from

$$\beta_k = \tfrac{1}{2}(1 + \|\hat{u}_k\|_2^2)^{1/2},$$

and therefore need not be saved. The algorithm requires $(mn^2 - n^3/3)$ multiplications, or $n^3/3$ less than for the MGS method. Note that in the special case that $m = n$ it would be possible to skip the last step which just computes $\tilde{H}_n = -1$ and $r_{nn} = -a_{nn}^{(n)}$.

Following Higham [Eq. (3.8)][33]) we will in the following frequently make use of the notation

$$\bar{\gamma}_k = \frac{cku}{1 - cku/2}, \tag{8.3.48}$$

where c denotes a small integer constant.

Theorem 8.3.4.

*Let $\bar{R}$ denote the upper triangular matrix R computed by the Householder QR algorithm. Then there exists an exactly orthogonal matrix $\hat{Q} \in \mathbf{R}^{m\times m}$ (*not *the matrix corresponding to exact computation throughout) such that*

$$A + E = \hat{Q} \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix},$$

where

$$\|e_j\|_2 \le \bar{\gamma}_n \|a_j\|_2, \quad j = 1 : n,$$

As have been stressed before it is usually not advisable to compute the matrix Q in the QR factorization explicitly, even when it is to be used in later calculations. In the rare case that the $Q = H_1 H_2 \cdots H_n$ from the Householder algorithm is explicitly required it should be accumulated in backward order by taking $Q^{(n)} = I_m$, and computing $Q = Q^{(0)}$ in $2(m^2 n - mn^2 + n^3/3)$ flops by the recursion

$$Q^{(k-1)} = H_k Q^{(k)}, \quad k = n : -1 : 1.$$

Note that by setting

$$Q^{(n)} = \begin{pmatrix} I_n \\ 0 \end{pmatrix}, \quad \text{or} \quad Q^{(n)} = \begin{pmatrix} 0 \\ I_{m-n} \end{pmatrix},$$

we can similarly accumulate Q_1 or Q_2 separately. $mn^2 - n^3/3$ and $2m^2 n - 3mn^2 + n^3$ flops, respectively; see Problem 3.

It is often advantageous to use **column pivoting** in the QR factorization and compute

$$AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{8.3.49}$$

where P is a permutation matrix. The following simple pivoting strategy, first suggested by Golub, has been shown to work well in practice. Assume that after k steps in the Householder Algorithm 7.3.3 we have computed the partial QR factorization

$$A^{(k+1)} = (H_k \cdots H_1) A (\Pi_1 \cdots \Pi_k) = \begin{pmatrix} R_{11}^{(k+1)} & R_{12}^{(k+1)} \\ 0 & \tilde{A}^{(k+1)} \end{pmatrix}, \tag{8.3.50}$$

Then the pivot column in the next step is chosen as a column of largest norm in the submatrix

$$\tilde{A}^{(k+1)} = (\tilde{a}_{k+1}^{(k+1)}, \ldots, \tilde{a}_n^{(k+1)}) \in \mathbf{R}^{(m-k)\times(n-k)},$$

i.e., Π_{k+1} is chosen to interchange columns p and $k+1$, where p is the smallest index such that

$$s_p^{(k+1)} \geq s_j^{(k+1)}, \qquad s_j^{(k+1)} = \|\tilde{a}_j^{(k+1)}\|_2^2, \qquad j = k+1, \ldots, n. \tag{8.3.51}$$

If $s_p^{(k+1)} = 0$ then the algorithm terminates with $\tilde{A}^{(k+1)} = 0$ in (8.3.50). This pivoting strategy can be viewed as choosing a remaining column of largest distance to the subspace spanned by the previously chosen columns. This is equivalent to maximizing the diagonal element $r_{k+1,k+1}$.

If the column norms in $\tilde{a}^{(k)}$ were recomputed at each stage, then column pivoting would increase the operation count by 50%. Instead the norms of the columns of A can be computed initially, and recursively updated as the factorization proceeds. This reduces the overhead of column pivoting to $0(mn)$ operations. This pivoting strategy can also be implemented in the Cholesky and modified Gram–Schmidt algorithms.

Since column norms are preserved by left orthogonal transformations it is easily shown that the elements in R, computed by QR factorization with pivoting, satisfy

$$r_{kk}^2 \geq \sum_{i=k}^{j} r_{ij}^2, \quad j = k+1, \ldots, n. \tag{8.3.52}$$

This implies in particular that $|r_{kk}| \geq |r_{kj}|$, $j > k$ and that the diagonal elements form a non-increasing sequence,

$$|r_{11}| \geq |r_{22}| \geq \cdots \geq |r_{nn}|. \tag{8.3.53}$$

To obtain near-peak performance for large dense matrix computations on current computing architectures requires code that is dominated by matrix-matrix operations since these involve less data movement per floating point computation. The QR factorization should therefore be organized in partitioned or blocked form, where the operations have been reordered and grouped into matrix operations.

For the QR factorization $A \in \mathbf{R}^{m\times n}$ $(m \geq n)$ is partitioned as

$$A = (A_1,\ A_2), \qquad A_1 \in \mathbf{R}^{m\times nb}, \tag{8.3.54}$$

where nb is a suitable block size and the QR factorization

$$Q_1^T A_1 = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, \qquad Q_1 = H_1 H_2 \cdots H_{nb}, \tag{8.3.55}$$

is computed, where $H_i = I - u_i u_i^T$ are Householder reflections. Then the remaining columns A_2 are are updated

$$Q_1^T A_2 = Q_1^T \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} R_{12} \\ \tilde{A}_{22} \end{pmatrix}. \tag{8.3.56}$$

In the next step we partition $\tilde{A}_{22} = (B_1, B_2)$, and compute the QR factorization of $B_1 \in \mathbf{R}^{(m-r)\times r}$. Then B_2 is updated as above, and we continue in this way until the columns in A are exhausted.

A major part of the computation in spent in the updating step (8.3.56). As written this step cannot use BLAS-3, which slows down the execution. To achieve better performance it is essential that this part is sped up. The solution is to aggregate the Householder transformations so that their application can be expressed as matrix operations. For use in the next subsection, we give a slightly more general result.

Lemma 8.3.5.

Let $H_1, H_2, \ldots, H_r$ be a sequence of Householder transformations. Set $r = r_1 + r_2$, and assume that

$$Q_1 = H_1 \cdots H_{r_1} = I - Y_1 T_1 Y_1^T, \qquad Q_2 = H_{r_1+1} \cdots H_r = I - Y_2 T_2 Y_2^T,$$

where $T_1, T_2 \in \mathbf{R}^{r\times r}$ are upper triangular matrices. Then for the product $Q_1 Q_2$ we have

$$Q = Q_1 Q_2 = (I - Y_1 T_1 Y_1^T)(I - Y_2 T_2 Y_2^T) = (I - YTY^T) \tag{8.3.57}$$

where

$$\hat{Y} = (Y_1,\ Y_2), \qquad \hat{T} = \begin{pmatrix} T_1 & -(T_1 Y_1^T)(Y_2 T_2) \\ 0 & T_2 \end{pmatrix}. \tag{8.3.58}$$

Note that Y is formed by concatenation, but computing the off-diagonal block in T requires extra operations.

For the partitioned algorithm we use the special case when $r_2 = 1$ to aggregate the Householder transformations for each processed block. Starting with $Q_1 = I - \tau_1 u_1 u_1^T$, we set $Y = u_1$, $T = \tau_1$ and update

$$Y := (Y,\ u_{k+1}), \qquad T := \begin{pmatrix} T & -\tau_k T Y^T u_k \\ 0 & \tau_k \end{pmatrix}, \quad k = 2 : nb. \tag{8.3.59}$$

Note that Y will have a trapezoidal form and thus the matrices Y and R can overwrite the matrix A. With the representation $Q = (I - YTY^T)$ the updating of A_2 becomes

$$B = Q_1^T A = (I - YT^T Y^T)A_2 = A_2 - YT^T Y^T A_2,$$

which now involves only matrix operations.

This partitioned algorithm requires more storage and operations than the point algorithm, namely those needed to produce and store the T matrices. However, for large matrices this is more than offset by the increased rate of execution.

As mentioned in Chapter 7 recursive algorithms can be developed into highly efficient algorithms for high performance computers and are an alternative to the currently more used partitioned algorithms by LAPACK. The reason for this is that recursion leads to automatic variable blocking that dynamically adjusts to an arbitrary number of levels of memory hierarchy.

Consider the partitioned QR factorization

$$A = (\, A_1 \quad A_2 \,) = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

where Let A_1 consist of the first $\lfloor n/2 \rfloor$ columns of A. To develop a recursive algorithm we start with a QR factorization of A_1 and update the remaining part A_2 of the matrix,

$$Q_1^T A_1 = \begin{pmatrix} R_{11} \\ 0 \end{pmatrix}, \qquad Q_1^T A_2 = Q_1^T \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} = \begin{pmatrix} R_{12} \\ \tilde{A}_{22} \end{pmatrix}.$$

Next $\tilde{A}_{22}$ is recursively QR decomposed giving Q_2, R_{22}, and $Q = Q_1 Q_2$.

As an illustration we give below a simple implementation in Matlab, which is convenient to use since it allows for the definition of recursive functions.

```
function [Y,T,R] = recqr(A)
%
% RECQR computes the QR factorization of the m by n matrix A,
% (m >= n). Output is the n by n triangular factor R, and
```

```
% Q = (I - YTY')  represented in aggregated form, where Y is
% m by n and unit lower trapezoidal, and T is n by n upper
% triangular
    [m,n] = size(A);
    if n == 1
    [Y,T,R] = house(A);
    else
      n1 = floor(n/2);
      n2 = n - n1; j = n1+1;
      [Y1,T1,R1]= recqr(A(1:m,1:n1));
      B = A(1:m,j:n) - (Y1*T1')*(Y1'*A(1:m,j:n));
      [Y2,T2,R2] = recqr(B(j:m,1:n2));
      R = [R1, B(1:n1,1:n2); zeros(n-n1,n1), R2];
      Y2 = [zeros(n1,n2); Y2];
      Y = [Y1, Y2];
      T = [T1, -T1*(Y1'*Y2)*T2; zeros(n2,n1), T2];
    end
%
```

The algorithm uses the function house(a) to compute a Householder transformation $P = I - +\tau\, uu^T$, such that $Pa = \sigma\, e1$, $\sigma = -\text{sign}\,(a_1)\|a\|_2$. A serious defect of this algorithm is the overhead in storage and operations caused by the T matrices. In the partitioned algorithm n/nb T-matrices of size we formed and stored, giving a storage overhead of $\frac{1}{2}n \cdot nb$. In the recursive QR algorithm in the end a T-matrix of size $n \times n$ is formed and stored, leading to a much too large storage and operation overhead. Therefore a better solution is to use a hybrid between the partitioned and the recursive algorithm, where the recursive QR algorithm is used to factorize the blocks in the partitioned algorithm.

8.3.6 Least Squares Problems by QR Factorization

We now show how to use the QR factorization to solve the linear least squares problem (8.1.1).

Theorem 8.3.6.

Let the QR factorization of $A \in \mathbf{R}^{m\times n}$ with $\text{rank}\,(A) = n \le m$ be given by (8.3.39). *Then the unique solution x to $\min_x ||Ax - b||_2$ and for the corresponding residual vector r are given by*

$$x = R^{-1}c_1, \quad c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = Q^T b, \quad r = Q \begin{pmatrix} 0 \\ c_2 \end{pmatrix}, \tag{8.3.60}$$

and hence $\|r\|_2 = \|c_2\|_2$.

Proof. Since Q is orthogonal we have

$$\|Ax-b\|_2^2 = \left\|Q^T(Ax-b)\right\|_2^2 = \left\|\begin{pmatrix} Rx \\ 0 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}\right\|_2^2 = \|Rx-c_1\|_2^2 + \|c_2\|_2^2.$$

Obviously the minimum residual norm $\|c_2\|_2$ is obtained by taking $x = R^{-1}c_1$. With c defined by (8.3.60) and using the orthogonality of Q we have

$$b = QQ^Tb = Q_1c_1 + Q_2c_2 = Ax + r$$

which shows the formula for r. □

By Theorem 8.3.6, when R and $H_1, H_2, \ldots, H_n$ have been computed by Algorithm 8.3.5 the least squares solution x and residual r can be computed from

$$\begin{aligned} n\{\ \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} &= H_n \cdots H_2H_1b, \qquad Rx = c_1, \\ r &= H_1 \cdots H_{n-1}H_n \begin{pmatrix} 0 \\ c_2 \end{pmatrix}, \end{aligned} \tag{8.3.61}$$

and $\|r\|_2 = \|c_2\|_2$. Note that the matrix Q should not be explicitly formed.

When $\operatorname{rank}(A) = m \le n$, i.e., the matrix A has full row rank, the QR factorization of A^T (which is equivalent to the LQ factorization of A) can be used to solve the minimum norm problem (8.1.2).

Theorem 8.3.7.
Let $A \in \mathbf{R}^{m\times n}$ with $\operatorname{rank}(A) = m$, have the LQ factorization

$$A = \begin{pmatrix} L & 0 \end{pmatrix} \begin{pmatrix} Q_1^T \\ Q_2^T \end{pmatrix}, \quad Q_1 \in \mathbf{R}^{n\times m},$$

Then the general solution to the underdetermined system $Ax = b$ is

$$x = Q_1y_1 + Q_2y_2, \qquad y_1 = L^{-1}b \tag{8.3.62}$$

where y_2 is arbitrary. The minimum norm solution is obtained by taking $y_2 = 0$,

$$x = Q_1L^{-1}b. \tag{8.3.63}$$

Proof. Since $A = \begin{pmatrix} L & 0 \end{pmatrix} Q^T$ the system $Ax = b$ can be written

$$\begin{pmatrix} L & 0 \end{pmatrix} y = b, \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = Q^Tx.$$

L is nonsingular, and thus y_1 is determined by $Ly_1 = b$. The vector y_2 can be chosen arbitrarily. Further, since $\|x\|_2 = \|Qy\|_2 = \|y\|_2$ the minimum norm solution is obtained by taking $y_2 = 0$. □

The operation count $mn^2 - n^3/3$ for the QR method can be compared with that for the method of normal equations, which requires $\frac{1}{2}(mn^2 + n^3/3)$ multiplications. Hence, for $m = n$ both methods require the same work but for $m \gg n$ the QR method is twice as expensive. To compute c by (8.3.61) requires $(2mn - n^2)$ multiplications, and thus to compute the solution for each new right hand side takes only $(2mn - n^2/2)$ multiplications. The Householder QR algorithm, and the resulting method for solving the least squares problem are backwards stable, both for x and r, and the following result holds.

Theorem 8.3.8.

*Let $\bar{R}$ denote the computed R. Then there exists an exactly orthogonal matrix $\tilde{Q} \in \mathbf{R}^{m\times m}$ (*not *the matrix corresponding to exact computation throughout) such that*

$$A + E = \tilde{Q}\begin{pmatrix} \bar{R} \\ 0 \end{pmatrix}, \qquad \|E\|_F \le cu\|A\|_F,$$

where $\|\cdot\|_F$ denotes the Frobenius norm, $c = 6n(m - n/2 + 7)$, and u is the machine precision. Further, the computed solution $\bar{x}$ is the exact solution of a slightly perturbed least squares problem

$$\min_x \|(A + \delta A)x - (b + \delta b)\|_2,$$

where the perturbation can be bounded in norm by

$$\|\delta A\|_F \le cu\|A\|_F, \qquad \|\delta b\|_2 \le cu\|b\|_2, \tag{8.3.64}$$

Proof. See Higham [33, Theorem 19.5]. □

A method combining LU factorization and orthogonalization can be developed by solving the least squares problem in (8.2.22) by an orthogonal reduction of L to lower triangular form. The solution is then obtained by solving $\tilde{L}y = c_1$ by forward substitution, where

$$L = Q\begin{pmatrix} \tilde{L} \\ 0 \end{pmatrix}, \qquad Q^T\Pi_1 b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}.$$

In **Cline's method**, Householder transformations can be used to perform this reduction of L. The kth Householder transformation P_k is chosen to affect only rows $k, n+1, \ldots, m$ and zero elements in column k below row n. The total number of flops required for computing the least squares solution x by Cline's method is about $n^2(\frac{3}{2}m - \frac{7}{6}n)$ flops. Since the method of normal equations using the Cholesky factorization on A^TA requires $n^2(\frac{1}{2}m + \frac{1}{6}n)$ flops Cline's method uses fewer operations if $m \le \frac{4}{3}n$. Hence for slightly overdetermined least squares problems, the elimination method combined with Householder transformations is very efficient.

A version solving (8.2.22) with the MGS method has been analyzed by Plemmons [50, 1974]. If the lower triangular structure of L is taken advantage of then

this method requires $n^2(\frac{3}{2}m - \frac{5}{6}n)$ flops, which is slightly more than Cline's variant. Similar methods for the underdetermined case ($m < n$) based on the LU decomposition of A have been studied by Cline and Plemmons [17, 1976].

An algorithm similar to Algorithm 8.3.5, but using Givens rotations, can easily be developed. The greater flexibility of Givens rotations can be taken advantage of when the matrix A is structured or sparse; see, e.g., Problem 3, where the QR factorization of a Hessenberg matrix is considered.

Peters and Wilkinson commented in 1970: "*Evidence is accumulating that the modified Gram–Schmidt method gives better results than Householder. . . . The reasons for this phenomenon appear not to have been elucidated yet.*" A key observation for understanding the good numerical properties of the modified Gram–Schmidt algorithm is that it can be interpreted as Householder QR factorization applied to the matrix A *augmented with a square matrix of zero elements on top.* These two algorithms are not only mathematically but also *numerically* equivalent. In the MGS method the columns are transformed by

$$a_j^{(k+1)} = M_k a_j^{(k)}, \qquad M_k = I - q_k q_k^T,$$

where M_k is the orthogonal projection onto the complement of q_k. In the Householder method one computes the factorization

$$P^T \begin{pmatrix} 0 \\ A \end{pmatrix} = \begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad P^T = P_n \cdots P_2 P_1,$$

$$P_k = I - v_k v_k^T, \qquad v_k = \begin{pmatrix} -e_k \\ q_k \end{pmatrix}.$$

Here $\|v_k\|_2^2 = 2$, and hence P_k is a Householder reflection. Because of the special structure of the augmented matrix the vectors v_k have a special form. Since the first n rows are initially zero, the scalar products of the vector v_k with later columns will only involve q_k, and it can be verified that the quantities r_{kj} and q_k are *numerically* equivalent to the quantities computed in the modified Gram–Schmidt method.

8.3.7 Condition and Error Estimation

Using the above pivoting strategy, a lower bound for $\kappa(A) = \kappa(R)$ can be obtained from the diagonal elements of R. We have $|r_{11}| \le \sigma_1 = \|R\|_2$, and since the diagonal elements of R^{-1} equal r_{ii}^{-1}, $i = 1, \ldots, n$, it follows that $r_{nn}^{-1}| \le \sigma_n^{-1} = \|R^{-1}\|_2$, provided $r_{nn} \neq 0$. Combining these estimates we obtain the *lower bound*

$$\kappa(A) = \sigma_1/\sigma_n \ge |r_{11}/r_{nn}| \tag{8.3.65}$$

Although this may considerably underestimate $\kappa(A)$, it has proved to give a fairly reliable estimate in practice. Extensive numerical testing has shown that (8.3.65) usually underestimates $\kappa(A)$ only by a factor of 2–3, and seldom by more than 10.

When column pivoting has not been performed, the above estimate of $\kappa(A)$ is not reliable. Then a condition estimator similar to that described in Sec. 7.6.5 can be used. Let u be a given vector and define v and w from

$$R^T v = u, \qquad Rw = v.$$

We have $w = R^{-1}(R^{-T}u) = (A^TA)^{-1}u$ so this is equivalent to one step of inverse iteration with A^TA, and requires about $0(n^2)$ multiplications. Provided that u is suitably chosen (cf. Sec. 7.6.5)

$$\sigma_n^{-1} \approx \|w\|_2/\|v\|_2$$

will usually be a good estimate of σ_n^{-1}. We can also take u as a random vector and perform and 2–3 steps of inverse iteration. This condition estimator will usually detect near rank deficiency even in the case when this is not revealed by a small diagonal element in R.

More reliable estimates can be based on the componentwise error bound (8.1.33) given in Sec. 8.1.5. This estimate has the form

$$\|\delta x\|_\infty \leq \omega(\| \, |B_1|g_1\|_\infty + \| \, |B_2|g_2\|_\infty), \tag{8.3.66}$$

where

$$B_1 = A^\dagger, \quad g_1 = |b| + |A||x|, \quad B_2 = (A^TA)^{-1}, \quad g_2 = |A^T||r|. \tag{8.3.67}$$

Consider now a general expression of the form $\| \, |B^{-1}|d\|_\infty$, where $d > 0$ is a known nonnegative vector. Writing $D = \operatorname{diag}(d)$ and $e = (1, 1, \ldots, 1)$, we have[3]

$$\| \, |B^{-1}|d\|_\infty = \| \, |B^{-1}|De\|_\infty = \| \, |B^{-1}D|e\|_\infty = \| \, |B^{-1}D|\|_\infty = \|B^{-1}D\|_\infty. \tag{8.3.68}$$

Hence the problem is equivalent to that of estimating $\|C\|_\infty$, where $C = B^{-1}D$. There are algorithms that produce reliable order-of-magnitude estimates of $\|C^T\|_1 = \|C\|_\infty$ using only a few matrix-vector products of the form Cx and C^Ty for some carefully selected vectors x and y. Since these are rather tricky we will not describe them in detail here. An excellent discussion is given in Higham [33, Chapter 15].

If A has full rank and $A = QR$ then $A^\dagger = R^{-1}Q^T$ and $(A^\dagger)^T = QR^{-T}$. Hence the required products can be computed inexpensively.

Review Questions

1 Let $w \in \mathbf{R}^n$, $\|w\|_2 = 1$. Show that $I - 2ww^T$ is orthogonal. What is the geometrical significance of the matrix $I - 2ww^T$? Give the eigenvalues and eigenvectors of these matrices.

2. Define a Givens transformations $G_{ij}(\phi) \in \mathbf{R}^{n\times n}$. Give a geometrical interpretations for the case $n = 2$.

3. Describe the difference between the classical and modified Gram–Schmidt methods for computing the factorization $A = Q_1R$. What can be said about the orthogonality of the computed matrix Q_1 for these two algorithms?

[3] This clever observation is due to Arioli, Demmel, and Duff [2].

4. Define the QR factorization of a matrix $A \in \mathbf{R}^{m\times n}$, in the case that $\operatorname{rank}(A) = n \le m$. What is its relation to the Cholesky factorization of A^TA?

Problems

1. Compute using Householder reflectors H_1, H_2, the factorization

$$Q^TA = H_2H_1A = \begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad A = (a_1, a_2) = \begin{pmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{pmatrix},$$

to four decimal places

2. Solve the least squares problem $\min_x \|Ax - b\|_2$, where

$$\begin{pmatrix} \sqrt{2} & 0 \\ 1 & -1 \\ 1 & 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

using a QR factorization computed with Givens transformation;

3. Suppose the square root free version of modified Gram–Schmidt is used to compute the factorization $A = \tilde{Q}_1\tilde{R}$. Modify Algorithm 8.3.2 for computing the least squares solution and residual from this factorization.

4. Describe in detail how to compute the QR factorization of a Hessenberg matrix $H \in \mathbf{R}^{n\times n}$ using Givens transformations. For $n = 5$ such a matrix has the form

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ & h_{32} & h_{33} & h_{34} & h_{35} \\ & & h_{43} & h_{44} & h_{45} \\ & & & h_{54} & h_{55} \end{pmatrix}.$$

Approximately how many multiplications are needed for general n?

5. (a) If the matrix Q in the QR factorization is explicitly required in the Householder algorithm it can be computed by setting $Q^{(n)} = I_m$, and computing $Q = Q^{(0)}$ by backward recursion

$$Q^{(k-1)} = H_kQ^{(k)}, \quad k = n : -1 : 1.$$

Show that if advantage is taken of the property that $H_k = \operatorname{diag}(I_{k-1}, \tilde{H}_k)$ this accumulation requires $2(m^2n - mn^2 + n^3/3)$ flops. What is the corresponding operation count if forward recursion is used?

(b) Show how we can compute

$$Q_1 = Q\begin{pmatrix} I_n \\ 0 \end{pmatrix}, \qquad Q_2 = Q\begin{pmatrix} 0 \\ I_{m-n} \end{pmatrix}$$

separately in $mn^2 - n^3/3$ and $2m^2n - 3mn^2 + n^3$ multiplications, respectively.

6. Let $Q = Q_1 = (q_1, q_2, \ldots, q_n) \in \mathbf{R}^{n\times n}$ be a real orthogonal matrix.
(a) Determine a reflector $H_1 = I - 2v_1v_1^T$, such that $H_1q_1 = e_1 = (1, 0, \ldots, 0)^T$, and show that $H_1Q_1 = Q_2$ has the form

$$Q_2 = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \tilde{Q}_2 & \\ 0 & & & \end{pmatrix},$$

where $\tilde{Q}_2 = (\tilde{q}_1, \tilde{q}_2, \ldots, \tilde{q}_n) \in \mathbf{R}^{(n-1)\times(n-1)}$ is a real orthogonal matrix.
(b) Show, using the result in (a), that Q can be transformed to diagonal form with a sequence of orthogonal transformations

$$H_{n-1} \cdots H_2H_1Q = \text{diag}\,(1, \ldots, 1, \pm 1).$$

7. An orthogonal matrix Q such that $\det(Q) = 1$ is called a rotation matrix. Show that any rotation matrix $Q \in \mathbf{R}^{3\times3}$ can be written as a product of three Givens rotations

$$Q = G_{23}(\phi)G_{12}(\theta)G_{23}(\psi).$$

The three angles ϕ, θ, and ψ are called the **Euler angles**.
Hint: Consider the QR factorization of Q.

8. Test the recursive QR algorithm `recqr(A)` given in Sec. sec8.3.6 on some matrices. Check that you obtain the same result as from the built-in function qr(A).

8.4 Rank Deficient and Ill-Posed Problems

8.4.1 Regularized Least Squares Problems

In solving linear systems and linear least squares problems failure to detect ill-conditioning and possible rank deficiency in A can lead to a meaningless solution of very large norm, or even to breakdown of the numerical algorithm. In this section we discuss how to assign a **numerical rank** to a matrix and how algorithms should be modified to cope with rank deficiency and ill-conditioning.

Example 8.4.1. Consider an example based on the integral equation of the first kind

$$\int_1^1 k(s,t)f(s)ds = g(t), \quad k(s,t) = e^{-(s-t)^2},$$

on $-1 \le t \le 1$. To compute $g(t)$ given $f(s)$ is well conditioned problem. However, the inverse problem of reconstructing $f(s)$ given $g(t)$ is a very ill-conditioned problem.

The equation can be discretized using a uniform mesh on [-1,1] and the trapezoidal rule, giving a finite-dimensional linear system $Kf = g$, where $K \in \mathbf{R}^{n\times n}$, and $f, g \in \mathbf{R}^n$. For $n = 100$ the singular values σ_k of the matrix K are displayed in

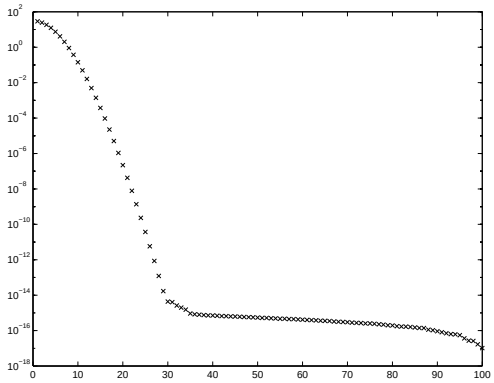


Figure 8.4.1. *Singular values of the matrix K.*

logarithmic scale in Figure 8.4.1. Note that for $k > 30$ all σ_k are close to roundoff level, so the numerical rank of K certainly is smaller than 30. This means that the linear system $Kf = g$ is numerically *under-determined* and has a meaningful solution only for special right hand sides g.

The choice of the parameter δ in Definition 8.1.15 is not always an easy matter. If the errors in a_{ij} satisfy $|e_{ij}| \leq \epsilon$, for all i, j, an appropriate choice is $\delta = (mn)^{1/2}\epsilon$. On the other hand, if the absolute size of the errors e_{ij} differs widely, then Definition 8.1.15 is not appropriate. One could then scale the rows and columns of A so that the magnitude of the errors become nearly equal. (Note that any such diagonal scaling $D_r A D_c$ will induce the same scaling $D_r E D_c$ of the error matrix.)

We now consider solving the linear least squares problem

$$\min_x \|Ax - b\|_2, \tag{8.4.1}$$

where the matrix A is ill-conditioned and possibly rank deficient. If A has numerical rank equal to $k < n$, we can get a more stable approximative solution by discarding terms in the expansion (8.1.10) corresponding to singular values smaller or equal to δ, and take the solution as the **truncated SVD (TSVD) solution**

$$x(\delta) = \sum_{\sigma_i > \delta} \frac{c_i}{\sigma_i} v_i. \tag{8.4.2}$$

If $\sigma_k > \delta \geq \sigma_{k+1}$ then the TSVD solution is $x(\delta) = A_k^\dagger b$ and solves the related least squares problem

$$\min_x \|A_k x - b\|_2, \qquad A_k = \sum_{\sigma_i > \delta} \sigma_i u_i v_i^T,$$

where A_k is the best rank k approximation of A. We have

$$\|A - A_k\|_2 = \|AV_2\|_2 \leq \delta, \quad V_2 = (v_{k+1}, \ldots, v_n).$$

In general the most reliable way to determine an approximate pseudo-inverse solution of a numerically rank deficient least squares problems is by first computing the SVD of A and then using an appropriate truncated SVD solution (8.4.2). However, this is also an expensive method. In practice the QR factorization often works as well, provided that some form of **column pivoting** is carried out.

An alternative to the truncated SVD (**TSVD**) solution is to consider the **regularized** problem

$$\min_x \ \|Ax - b\|_2^2 + \mu^2 \|Dx\|_2^2, \tag{8.4.3}$$

where $D = \operatorname{diag}(d_1, \ldots, d_n) > 0$ is a positive diagonal matrix. The problem (8.4.3), also called a **damped** least squares problem, is equivalent to the least squares problem

$$\min_x \left\| \begin{pmatrix} A \\ \mu D \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2, \tag{8.4.4}$$

where the matrix A has been modified by appending the matrix μD. When $\mu > 0$ this problem is always of full column rank and has a unique solution. (Often d_j is taken to be proportional to the 2-norm of the jth column in A.)

The solution to problem (8.4.3) satisfies the normal equations

$$(A^T A + \mu^2 D^2)x = A^T b.$$

However, from the formulation (8.4.4) it is seen that the solution can also be obtained from the QR factorization

$$\begin{pmatrix} A \\ \mu D \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{8.4.5}$$

which can be computed by some of the algorithms described before. The special structure can be taken advantage of. For example, in the Householder QR factorization the shape of the transformed matrix after $k = 2$ steps is as follows ($m = n = 4$):

$$\begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & + & + \\ & 0 & + & + \\ & & \times & \\ & & & \times \end{pmatrix}$$

Notice that the first two rows of D have filled in, but the remaining rows of D are still not touched. For each step $k = 1, \ldots, n$ there are m elements in the current column to be annihilated. Therefore the operation count for the Householder QR factorization will increase with $n^3/3$ to mn^2 flops. A similar increase in operations occurs in Givens or MGS QR factorizations. If $A = R$ already is in upper triangular form then the flop count for the reduction is reduced to approximately $n^3/3$ (cf. Problem 1b).

If $D = I$ the singular values of the modified matrix in (8.4.4) are equal to $\tilde{\sigma}_i = (\sigma_i^2 + \mu^2)^{1/2}$, $i = 1, \ldots, n$. In this case the solution can be expressed in terms of the SVD as

$$x(\mu) = \sum_{i=1}^{n} f_i \frac{c_i}{\sigma_i} v_i, \quad f_i = \frac{\sigma_i^2}{\sigma_i^2 + \mu^2}. \tag{8.4.6}$$

The quantities f_i are often called **filter factors**. Notice that as long as $\mu \ll \sigma_i$ we have $f_i \approx 1$, and if $\mu \gg \sigma_i$ then $f_i \ll 1$. This establishes a relation to the truncated SVD solution (8.4.2) which corresponds to a filter factor which is a step function $f_i = 1$ if $\sigma_i > \delta$ and $f_i = 0$ otherwise.

Note that the regularized problem (8.4.3) can be used also when $m < n$ (i.e., when A has fewer rows than columns). However, in this case it may be better to consider the regularized problem

$$\min \left\| \begin{pmatrix} x \\ z \end{pmatrix} \right\|_2^2, \quad \text{subject to} \quad (A \ \mu D) \begin{pmatrix} x \\ z \end{pmatrix} = b. \tag{8.4.7}$$

The solution of this problem can be written $x = A^T y$, $z = (\mu D)^{-1}(b - Ax)$, where y satisfies the system of normal equations

$$(AA^T + \mu^2 D^2)y = b.$$

Using Theorem 8.3.7, a method for solving problem (8.4.7) is obtained which uses the QR factorization of the matrix $(\mu D \ A)^T$, which can be computed in $m^2 n$ operations. Surprisingly, when $D = I$ the two problems (8.4.3) and (8.4.7) are equivalent. To see this set note that since $z = (\mu)^{-1}(b - Ax)$, both problems (8.4.4) and (8.4.7) are equivalent to

$$\min_x \left\{ \|r\|_2^2 + \mu^2 \|x\|_2^2 \right\}, \quad r = b - Ax.$$

Even with regularization we may not be able to compute the solution of an ill-conditioned problem with the accuracy that the data allows. In those cases it is possible to improve the solution by the following **iterated regularization** scheme. Take $x^{(0)} = 0$, and compute a sequence of approximate solutions by

$$x^{(q+1)} = x^{(q)} + \delta x^{(q)},$$

where $\delta x^{(q)}$ solves the least squares problem

$$\min_{\delta x} \left\| \begin{pmatrix} A \\ \mu I \end{pmatrix} \delta x - \begin{pmatrix} r^{(q)} \\ 0 \end{pmatrix} \right\|_2, \qquad r^{(q)} = b - Ax^{(q)}. \tag{8.4.8}$$

This iteration may be implemented very effectively since only the QR factorization (8.4.5) (with $D = I$) is needed. The convergence of iterated regularization can be expressed in terms of the SVD of A.

$$x^{(q)}(\mu) = \sum_{i=1}^{n} f_i^{(q)} \frac{c_i}{\sigma_i} v_i, \quad f_i^{(q)} = 1 - \Big(\frac{\mu^2}{\sigma_i^2 + \mu^2} \Big)^q. \tag{8.4.9}$$

Thus for $q = 1$ we have the standard regularized solution and as $q \to \infty$ $x^{(q)} \to A^\dagger b$.

8.4.2 QR Factorization and Rank Deficient Matrices

Although any matrix $A \in R^{m\times n}$ has a QR factorization, the following example shows that this may not always be useful when $\operatorname{rank}(A) < n$:

Example 8.4.2.
For any c and s such that $c^2 + s^2 = 1$ we have

$$A = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} 0 & s \\ 0 & c \end{pmatrix} = QR.$$

Here $\operatorname{rank}(A) = 1 < 2 = n$. Note that the columns of Q no longer provide any information about an orthogonal basis for $R(A)$ and its complement.

We now indicate how the QR factorization should be modified in the rank deficient case.

Theorem 8.4.1.
Given $A \in \mathbf{R}^{m\times n}$ with $\operatorname{rank}(A) = r \le \min(m, n)$ there is a permutation matrix Π and an orthogonal matrix $Q = \in \mathbf{R}^{m\times m}$ such that

$$A\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \tag{8.4.10}$$

where $R_{11} \in \mathbf{R}^{r\times r}$ is upper triangular with positive diagonal elements.

Proof. Since $\operatorname{rank}(A) = r$, we can always choose a permutation matrix Π such that $A\Pi = (A_1, A_2)$, where $A_1 \in \mathbf{R}^{m\times r}$ has linearly independent columns. Then A_1 has a QR factorization and we can write

$$Q^T A\Pi = (\, Q^T A_1 \quad Q^T A_2 \,) = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$

where R_{11} has positive diagonal elements. From $\operatorname{rank}(Q^T A\Pi) = \operatorname{rank}(A) = r$ it follows that $R_{22} = 0$, since otherwise $Q^T A\Pi$ would have more than r linearly independent rows. □

Note that it is not required that $m \ge n$ in Theorem 8.4.1. The factorization is not unique, since there may be several ways to choose the permutation Π. Pivoting strategies for determining a suitable Π will be discussed later in this section.

The factorization (8.4.10) can be used to solve rank deficient linear least squares problems. To simplify notations we assume in the following that $\Pi = I$. (This is no restriction since the column permutation of A can always be assumed to have been carried out in advance.) Using the invariance of the l_2-norm problem (8.1.1) becomes

$$\min_x \left\| \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \right\|_2,$$

where x and c have been partitioned conformally. Since R_{11} is nonsingular the first r equations can be satisfied for any x_2 by taking x_1 to be the solution to $R_{11}x_1 = c_1 - R_{12}x_2$. Hence the general least squares solutions can be written

$$x_1 = R_{11}^{-1}(c_1 - R_{12}x_2) = x_b - C_1x_2, \tag{8.4.11}$$

where x_2 is arbitrary and

$$d = R_{11}^{-1}c_1, \qquad C = R_{11}^{-1}R_{12}. \tag{8.4.12}$$

Here C can be computed by solving $n - r$ triangular systems $R_{11}C = R_{12}$, which requires $r^2(n-r)/2$ multiplications.

Taking $x_2 = 0$ we we obtain a particular solution $x_1 = d$ with at most $r = \text{rank}(A)$ nonzero components Any solution x such that Ax only involves at most r columns of A, is called a **basic least squares solution**. Such a solution is appropriate when we want to fit a vector b of observations using *as few columns of* A as possible. It is not unique and depends on the initial column permutation.

We now show how the pseudo-inverse solution can be computed using the factorization (8.4.10). Then we want to choose x_2 so that $\|x\|_2$ is minimized. From (8.4.11) it follows that this is achieved by solving the linear least squares problem for x_2

$$\min \left\| \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\|_2 = \min_{x_2} \left\| \begin{pmatrix} d \\ 0 \end{pmatrix} - \begin{pmatrix} C \\ -I_{n-r} \end{pmatrix} x_2 \right\|_2. \tag{8.4.13}$$

Note that this problem always has a unique solution x_2 and that the pseudo-inverse solution $x = A^{\dagger}b$ equals *the residual* of the problem.

To compute x_2 we can form and solve the normal equations

$$(I + CC^T)x_2 = C^T d. \tag{8.4.14}$$

Alternatively we can use Householder QR factorization

$$Q_C^T \begin{pmatrix} C \\ I_{n-r} \end{pmatrix} = \begin{pmatrix} R_C \\ 0 \end{pmatrix}, \qquad Q_C^T \begin{pmatrix} d \\ 0 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix},$$

taking the special structure into account, to obtain x_2 from $R_C x_2 = d_1$.

We have

$$A \begin{pmatrix} C \\ -I_{n-r} \end{pmatrix} = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} R_{11}^{-1}R_{12} \\ -I_{n-r} \end{pmatrix} = 0,$$

from which it follows that the nullspace of A is given by

$$\mathcal{N}(A) = \mathcal{R}(W), \quad W = \begin{pmatrix} C \\ -I_{n-r} \end{pmatrix}. \tag{8.4.15}$$

By Theorem 8.1.7 the pseudo-inverse solution is the unique least squares solution which satisfies $x \perp \mathcal{N}(A)$. Hence it can be obtained by Gram–Schmidt orthogonalization applied to

$$\begin{pmatrix} C & d \\ I_{n-r} & 0 \end{pmatrix}. \tag{8.4.16}$$

It is possible to carry the factorization one step further to give the related **complete QR factorization** of A.

Theorem 8.4.2.

Given $A \in \mathbf{R}^{m\times n}$ with $\operatorname{rank}(A) = r \le \min(m,n)$. Then there are orthogonal matrices $Q = (Q_1, Q_2) \in \mathbf{R}^{m\times m}$, and $V = (V_1, V_2) \in \mathbf{R}^{n\times n}$ such that

$$A = Q \begin{pmatrix} R & 0 \\ 0 & 0 \end{pmatrix} V^T \tag{8.4.17}$$

where $R \in \mathbf{R}^{r\times r}$ is upper triangular with positive diagonal elements. The pseudo-inverse of A is then given by

$$A^\dagger = V \begin{pmatrix} R^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q^T = V_1 R^{-1} Q_1^T. \tag{8.4.18}$$

Proof. Starting from the factorization in (8.4.10) we can determine a sequence of Householder matrices such that

$$(\, R_{11} \quad R_{12}\,) P_r \cdots P_1 = (\, R \quad 0\,).$$

Here P_k, $k = r, r-1, \ldots, 1$, is constructed to zero elements in row k and only affect columns $k, r+1, \ldots, n$. These transformations require $r^2(n-r)$ multiplications. Then (8.4.17) holds with $V = \Pi P_1 \cdots P_r$. Using the orthogonal invariance of the l_2-norm it follows that $x = V_1 R^{-1} Q_1^T b$ is the minimum norm solution of the least squares problem (8.1.1). Since the pseudo-inverse is uniquely defined by this property, cf. Theorem 8.1.5, the last assertion follows. □

8.4.3 Rank Revealing QR Factorization

In Sec. 8.3.5 we studied the pivoted QR factorization. It was shown that if the pivot column in each step of the reduction was chosen as a column of largest norm in the remaining part, then we have the inequalities

$$r_{kk}^2 \ge \sum_{i=k}^{j} r_{ij}^2, \quad j = k+1, \ldots, n. \tag{8.4.19}$$

in particular it holds that $|r_{kk}| \ge |r_{kj}|$, $j > k$ and the diagonal elements form a non-increasing sequence, $|r_{11}| \ge |r_{22}| \ge \cdots \ge |r_{nn}|$.

Taking $x = e_1$ in $\sigma_1 = \max_{\|x\|=1} \|Ax\|_2$ we find that the lower bound $|r_{11}| \le \sigma_1(R)$ for the largest singular value σ_1. The matrix R^{-1} has diagonal elements $1/r_{kk}$ and singular values $1/\sigma_k(A)$. Hence we also have the inequality $\sigma_n \le |r_{nn}|$.

For a triangular matrix satisfying (8.4.19) we also have the upper bound

$$\sigma_1(R) = \|R\|_2 \le \Big(\sum_{i\le j} r_{ij}^2 \Big)^{1/2} \le \sqrt{n} r_{11},$$

and hence $\sigma_1(R) \le n^{1/2} r_{11}$. Using the interlacing property of singular values (Theorem 8.1.13), a similar argument gives the upper bounds

$$\sigma_k(R) \le (n-k+1)^{1/2} |r_{k,k}|, \quad 1 \le k \le n. \tag{8.4.20}$$

If after k steps in the pivoted QR factorization it holds that

$$|r_{k,k}| \leq (n-k+1)^{-1/2}\delta,$$

then $\sigma_k(A) = \sigma_k(R) \leq \delta$, and A has numerical rank at most equal to $k-1$, and we should terminate the algorithm. Unfortunately, the converse is not true, i.e., the rank is not always revealed by a small element $|r_{kk}|$, $k \leq n$. Let R be an upper triangular matrix whose elements satisfy (8.3.52). The best known *lower* bound for the smallest singular value is

$$\sigma_n \geq 3|r_{nn}|/\sqrt{4^n + 6n - 1} \geq 2^{1-n}|r_{nn}|. \tag{8.4.21}$$

(For a proof see Lawson and Hanson [38, Ch. 6].)

The lower bound in (8.4.21) can almost be attained as shown in the example below due to Kahan. Then the pivoted QR factorization may not reveal the rank of A.

Example 8.4.3. Consider the upper triangular matrix

$$R_n = \operatorname{diag}(1, s, s^2, \ldots, s^{n-1}) \begin{pmatrix} 1 & -c & -c & \ldots & -c \\ & 1 & -c & \ldots & -c \\ & & 1 & & \vdots \\ & & & \ddots & -c \\ & & & & 1 \end{pmatrix}, \qquad s^2 + c^2 = 1.$$

It can be verified that the elements in R_n satisfies the inequalities in (8.4.21), and that R_n is invariant under QR factorization with column pivoting. For $n = 100$, $c = 0.2$ the last diagonal element of R is $r_{nn} = s^{n-1} = 0.820$. This can be compared with the smallest singular value which is $\sigma_n = 0.368 \cdot 10^{-8}$. If the columns are reordered as $(n, 1, 2, \ldots, n-1)$ and the rank is revealed from the pivoted QR factorization!

The above example did inspire research into alternative column permutation strategies. The following theorem, which we state without proof, shows that a column permutation Π can always be found so that the numerical rank of A is revealed by the QR factorization of $A\Pi$.

Theorem 8.4.3. (H. P. Hong and C. T. Pan [1992].)

Let $A \in \mathbf{R}^{m\times n}$, $(m \geq n)$, and r be a given integer $0 < r < n$. Then there exists a permutation matrix Π_r, such that the QR factorization has the form

$$Q^T A\Pi_r = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}, \tag{8.4.22}$$

with $R_{11} \in \mathbf{R}^{r\times r}$ upper triangular, $c = \sqrt{r(n-r) + \min(r, n-r)})$, and

$$\sigma_{\min}(R_{11}) \geq \frac{1}{c}\sigma_r(A), \qquad \sigma_{\max}(R_{22}) \leq c\sigma_{r+1}(A). \tag{8.4.23}$$

Note that the bounds in this theorem are much better than those in (8.4.21).

From the interlacing properties of singular values (Theorem 8.1.13) it follows by induction that for any factorization of the form (8.4.22) we have the inequalities

$$\sigma_{\min}(R_{11}) \le \sigma_r(A), \qquad \sigma_{\max}(R_{22}) \ge \sigma_{r+1}(A). \tag{8.4.24}$$

Hence to achieve (8.4.23) we want to choose the permutation Π to maximize $\sigma_{\min}(R_{11})$ and simultaneously minimize $\sigma_{\max}(R_{22})$. These two problems are in a certain sense dual; cf. Problem 2.

Assume now that A has a well defined numerical rank $r < n$, i.e.,

$$\sigma_1 \ge \ldots \ge \sigma_r \gg \delta \ge \sigma_{r+1} \ge \ldots \ge \sigma_n.$$

Then the above theorem says that if the ratio σ_k/σ_{k+1} is sufficiently large then there is a permutation of the columns of A such that the rank of A is revealed by the QR factorization. Unfortunately, to find such a permutation may be a hard problem. The naive solution, to try all possible permutations, is not feasible since the cost prohibitive—it is exponential in the dimension n.

Many other pivoting strategies for computing rank revealing QR factorizations have been proposed. A strategy by T. F. Chan [14] makes use of approximate right singular vectors of A, which can be determined by inverse iteration (see Sec. 9.4.3). In case $r = n-1$, the column permutation Π is constructed from an approximation to the right singular vector corresponding to the smallest singular value σ_n.

8.4.4 The URV and ULV decompositions

In signal processing problems it is often the case that one wants to determine the rank of A as well as the range (signal subspace) and null space of A. Since the data analyzed arrives in real time it is necessary to update an appropriate matrix decompositions at each time step. For such applications the SVD has the disadvantage that it cannot in general be updated in less than $\mathcal{O}(n^3)$ operations, when rows and columns are added or deleted to A. Although the RRQR decomposition can be updated, it is less suitable in applications where a basis for the approximate null space of A is needed, since the matrix W in (8.4.15) cannot easily be updated.

For this reason we introduce the **URV decomposition**

$$A = URV^T = (\,U_1 \quad U_2\,) \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}, \tag{8.4.25}$$

where U and V are orthogonal matrices, $R_{11} \in \mathbb{R}^{k\times k}$, and

$$\sigma_k(R_{11}) \ge \frac{1}{c}\sigma_k, \qquad \left(\|R_{12}\|_F^2 + \|R_{22}\|_F^2\right)^{1/2} \le c\sigma_{k+1}. \tag{8.4.26}$$

Note that here both submatrices R_{12} and R_{22} have small elements.

From (8.4.25) we have

$$\|AV_2\|_2 = \left\| \begin{pmatrix} R_{12} \\ R_{22} \end{pmatrix} \right\|_F \le c\sigma_{k+1},$$

and hence the orthogonal matrix V_2 can be taken as an approximation to the numerical null space $\mathcal{N}_k$.

Algorithms for computing an URV decomposition start with an initial QR decomposition, followed by a rank revealing stage in which singular vectors corresponding to the smallest singular values of R are estimated. Assume that w is a unit vector such that $\|Rw\| = \sigma_n$. Let P and Q be a orthogonal matrices such that $Q^T w = e_n$ and $P^T RQ = \hat{R}$ where $\hat{R}$ is upper triangular. Then

$$\|\hat{R}e_n\| = \|P^T RQQ^T w\| = \|P^T Rw\| = \sigma_n,$$

which shows that the entire last column in $\hat{R}$ is small. Given w the matrices P and Q can be constructed as a sequence of Givens rotations, Algorithms can also be given for updating an URV decomposition when a new row is appended.

Like the RRQR decompositions the URV decomposition yield approximations to the singular values. In [41] the following bounds are derived

$$f\sigma_i \le \sigma_i(R_{11}) \le \sigma_i, \quad i = 1 : r,$$

and

$$\sigma_i \le \sigma_{i-k}(R_{22}) \le \sigma_i/f, \quad i = r+1 : n,$$

where

$$f = \left(1 - \frac{\|R_{12}\|_2^2}{\sigma_{\min}(R_{11})^2 - \|R_{22}\|_2^2}\right)^{1/2}.$$

Hence the smaller the norm of the off-diagonal block R_{12}, the better the bounds will be. Similar bounds can be given for the angle between the range of V_2 and the right singular subspace corresponding to the smallest $n - r$ singular values of A.

An alternative decomposition that is more satisfactory for applications where an accurate approximate null space is needed, is the rank-revealing **ULV decomposition**

$$A = U \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} V^T. \tag{8.4.27}$$

where the middle matrix has lower triangular form. For this decomposition

$$\|AV_2\|_2 = \|L_{22}\|_F, \qquad V = (V_1,\ V_2),$$

and hence the size of $\|L_{21}\|$ does not adversely affect the null space approximation. On the other hand the URV decomposition usually gives a superior approximation for the numerical range space and the updating algorithm for URV is much simpler.

We finally mention that rank-revealing QR decompositions can be effectively computed only if the numerical rank r is either high, $r \approx n$ or low, $r \ll n$. The low rank case is discussed in [15]. Matlab templates for rank-revealing UTV decompositions are described in [22].

An advantage of the complete QR factorization of A is that V_2 gives an orthogonal basis for the nullspace $\mathcal{N}(A)$. This is often useful, e.g., in signal processing applications, where one wants to determine the part of the signal that corresponds to noise. The factorization (8.4.18) can be generalized to the case when A is only

numerically rank deficient in a similar way as done above for the QR factorization. The resulting factorizations have one of the forms

$$A = Q \begin{pmatrix} R & F \\ 0 & G \end{pmatrix} V^T \qquad A = Q \begin{pmatrix} R^T & 0 \\ F^T & G^T \end{pmatrix} V^T \tag{8.4.28}$$

where R is upper triangular and

$$\sigma_k(R) > \frac{1}{c}, \qquad (\|F\|_F^2 + \|G\|_F^2)^{1/2} \le c\sigma_{k+1}.$$

An advantage is that unlike the SVD it is possible to efficiently update the factorizations (8.4.28) when rows/columns are added/deleted.

8.4.5 Bidiagonal Decomposition and Least Squares

So far we have considered methods based on the QR factorization of A for solving least squares problems. It is possible to carry this reduction further using a two-sided orthogonal factorization.

Theorem 8.4.4.
Any matrix $A \in \mathbb{R}^{m\times n}$ can be decomposed as

$$A = UBV^T, \tag{8.4.29}$$

where B is a lower bidiagonal matrix and U and V are orthogonal matrices. In the nondegenerate case the decomposition is uniquely determined by $u_1 := Ue_1$, which can be chosen arbitrarily.

Note that, since $A^T = VB^TU^T$, it follows that an arbitrary matrix A can alternatively be transformed to *upper* bidiagonal form.

This decomposition is usually the first step in computing the SVD of A; see Sec. 9.7. It is also powerful tool for solving various least squares problems. We will give a constructive proof of this theorem below.

In the Golub–Kahan algorithm the reduction is achieved by applying a sequence of Householder reflections alternately from left and right. We set $A = A^{(1)}$ and in the first step compute

$$A^{(2)} = Q_1(AP_1) = \begin{pmatrix} \alpha_1 & 0 & 0 & \cdots & 0 \\ \beta_2 & \tilde{a}_{22} & \tilde{a}_{23} & \cdots & \tilde{a}_{2n} \\ 0 & \tilde{a}_{32} & \tilde{a}_{33} & \cdots & \tilde{a}_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \tilde{a}_{m2} & \tilde{a}_{m3} & \cdots & \tilde{a}_{mn} \end{pmatrix}.$$

First P_1 is chosen to zero the $n-1$ elements in the first column of A above the main diagonal. Next Q_1 is chosen to zero the last $m-2$ elements in the the first row of AP_1. This transformation does not affect the zeros introduced in the first row.

All later steps are similar and in the kth step, $k = 1 : \min(m, n)$, we compute

$$A^{(k+1)} = Q_k(A^{(k)}P_k),$$

where Q_k and P_k are Householder reflections. Here P_k is chosen to zero the last $n-k$ elements in the kth row of $A^{(k)}$. Then Q_k is chosen to zero the last $m-(k+1)$ elements in the kth column of $A^{(k)}P_k$.

When $m > n$ the process ends with the factorization

$$U^TAV = \begin{pmatrix} B \\ 0 \end{pmatrix}, \quad B = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_n \\ & & & \beta_{n+1} \end{pmatrix} \in \mathbf{R}^{(n+1)\times n}, \tag{8.4.30}$$

$$U = Q_1Q_2\cdots Q_n, \qquad V = P_1P_2\cdots P_{n-1}. \tag{8.4.31}$$

Note that since Q_k, only works on rows $k+1 : m$, and P_k, only works on columns $k : m$. It follows that

$$u_1 = e_1. \quad u_k = Ue_k = Q_1\cdots Q_ke_k, \quad k = 2 : n, \tag{8.4.32}$$

$$v_k = Ve_k = P_1\cdots P_ke_k, \quad k = 1 : n-1, \quad v_n = e_n. \tag{8.4.33}$$

If $m \le n$ then we obtain

$$U^TAV = (\, B \quad 0\,), \quad B = \begin{pmatrix} \alpha_1 & & & & \\ \beta_2 & \alpha_2 & & & \\ & \beta_3 & \ddots & & \\ & & \ddots & \alpha_{m-1} & \\ & & & \beta_m & \alpha_m \end{pmatrix} \in \mathbf{R}^{m\times m}.$$

$$U = Q_1Q_2\cdots Q_{m-2}, \qquad V = P_1P_2\cdots P_{m-1}.$$

The above process can always be carried through although some elements in B may vanish. Note that the singular values of B equal those of A; in particular $\operatorname{rank}(A) = \operatorname{rank}(B)$. Using complex Householder transformations (see Sec. 9.6.2) a complex matrix A can be reduced to *real* bidiagonal form. by the algorithm above.

The reduction to bidiagonal form is backward stable in the following sense. The computed $\bar{B}$ can be shown to be the exact result of an orthogonal transformation from left and right of a matrix $A + E$, where

$$\|E\|_F \le cn^2u\|A\|_F, \tag{8.4.34}$$

and c is a constant of order unity. Moreover, if we use the information stored to generate the products $U = Q_1\cdots Q_n$ and $V = P_1\cdots P_{n-2}$ then the computed matrices are close to the exact matrices U and V which reduce $A + E$. This will guarantee that the singular values and transformed singular vectors of $\bar{B}$ are accurate approximations to those of a matrix close to A.

The bidiagonal reduction algorithm as described above requires approximately

$$4(mn^2 - n^3/3) \text{ flops}$$

when $m \geq n$, which is twice the work for a Householder QR factorization. The Householder vectors associated with U can be stored in the lower triangular part of A and those associated with V in the upper triangular part of A. Normally U and V are not explicitly required. They can be accumulated at a cost of $4(m^2n - mn^2 + \frac{1}{3}n^3)$ and $\frac{4}{3}n^3$ flops respectively.

When $m \gg n$ it is more efficient to use a two-step procedure as originally suggested by Lawson and Hanson [38] and later analyzed by T. Chan. In the first step the QR factorization of A is computed (possibly using column pivoting)

$$AP = Q \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad R \in \mathbf{R}^{n \times n},$$

which requires $4mn^2 - \frac{2}{3}n^3$ flops. In the second step the upper triangular matrix R is transformed to bidiagonal form using the algorithm described above. Note that no advantage can be taken of the triangular structure of R in the Householder algorithm. Already the first postmultiplication of R with P_1 will cause the lower triangular part of R to fill in. Hence the Householder reduction of R to bidiagonal form will require $\frac{4}{3}n^3$ flops. The complete reduction to bidiagonal form then takes a total of

$$2(mn^2 + n^3) \text{ flops.}$$

This is less than the original Golub–Kahan algorithm when $m/n > 5/3$. Trefethen and Bau [62, pp. 237–238] have suggested a blend of the two above approaches that reduces the operation count slightly for $1 < m/n < 2$. They note that after k steps of the Golub–Kahan reduction the aspect ratio of the reduced matrix is $(m-k)/(n-k)$. and thus increases with k. To minimize the total operation count one should switch to the Chan algorithm when $(m-k)/(n-k) = 2$. This gives the operation count

$$4(mn^2 - n^3/3 - (m-n)^3/6) \text{ flops,}$$

a modest approval over the two other methods when $n > m > 2n$.

If Givens transformation are used to reduce R to upper bidiagonal form it is possible to take advantage of the triangular form, provided that the elements are annihilated in a suitable order. In the first major step we can zero the elements in the first row from right to left, i.e. in the order $r_{1n}, \ldots, r_{13}$. To zero r_{1j} the columns $(j-1, j)$ are rotated using a Givens rotation $G_{j-1,j}$ from the right. This introduces one new non-zero element $r_{j,j-1}$ in the *lower triangular part*, which can be annihilated by a rotation of the rows $(j-1, j)$, applying a Givens rotation $\tilde{G}_{j-1,j}$ from the left. This is illustrated below, where the element r_{14} is zeroed by first rotating columns 3,4 followed by a rotation of rows 3,4.

$$\begin{pmatrix} \times & \times & \times & \otimes & 0 & 0 \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & \otimes & \times & \times & \times \\ & & & & \times & \times \\ & & & & & \times \end{pmatrix}$$

After zeroing the last $n-2$ elements in the first row we continue the reduction on the triangular submatrix in rows and columns $2:n$ in the same fashion.

Since *two* Givens rotations are needed to zero each of the $(n-1)(n-2)/2$ elements, the operation count turns out to be about the same as for the Householder reduction if standard Givens rotations are used. If the transformations are to be accumulated the Givens reduction will require more work, unless fast Givens transformations are used.

When A is a banded matrix of bandwidth $w > 2$ then R will be an upper triangular banded matrix ($w = 2$ corresponds to a bidiagonal matrix). In this case the reduction of R to bidiagonal form can be accomplished by successively reducing the bandwidth by one. (This algorithm is similar to an algorithm by Schwarz [55] for reducing a symmetric banded matrix to tridiagonal form.) Each zero element introduced generates a new nonzero element that has to be chased across the border of the matrix. Because of this the reduction is expensive unless the bandwidth is small.

Example 8.4.4.

Let $w = 3$ and $n = 7$. The figure below illustrates the steps in zeroing out the element r_{13} using Givens rotations applied alternately from the right and left

$$\begin{pmatrix} \times & \times & \times & & & & \\ & \times & \times & \times & & & \\ & & \times & \times & \times & & \\ & & & \times & \times & \times & \\ & & & & \times & \times & \times \\ & & & & & \times & \times \\ & & & & & & \times \end{pmatrix} \Rightarrow \begin{pmatrix} \times & \times & \otimes & & & & \\ & \times & \times & \times & \oplus & & \\ & \oplus & \times & \times & \times & & \\ & & & \times & \times & \times & \oplus \\ & & & \oplus & \times & \times & \times \\ & & & & & \times & \times \\ & & & & & \oplus & \times \end{pmatrix}.$$

corresponding to the transformations

$$G_{67}((G_{45}((G_{23}(RG_{23}))G_{45}))G_{67}).$$

Then the elements $r_{13}, \ldots, r_{n-2,n}$ are eliminated in this order. Such "chasing" algorithms are also commonly used in eigenvalue algorithms. Reduction of an upper triangular matrix of bandwidth w to bidiagonal form requires $\approx 4n^2(w-2)$ multiplications.

We now derive an algorithm for solving the linear least squares problem $\min \|Ax - b\|_2$, where $A \in \mathbf{R}^{m\times n}$, $m \geq n$. Let Q_0 be a Householder reflection such that

$$Q_1 b = \beta_1 e_1. \tag{8.4.35}$$

Using the Golub–Kahan algorithm $Q_1 A$ to lower triangular form. we obtain

$$U^T \begin{pmatrix} b & AV \end{pmatrix} = \begin{pmatrix} \beta_1 e_1 & B_n \\ 0 & 0 \end{pmatrix}, \tag{8.4.36}$$

where e_1 is the first unit vector, and B_n is lower bidiagonal,

$$B_n = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_n & \alpha_n \\ & & & \beta_{n+1} \end{pmatrix} \in \mathbf{R}^{(n+1)\times n}, \tag{8.4.37}$$

and

$$U = Q_1 Q_2 \cdots Q_{n+1}. \qquad V = P_1 P_2 \cdots P_{n-1}. \tag{8.4.38}$$

(Note the minor difference in notation in that Q_{k+1} now zeros elements in the kth column of A.)

Setting $x = Vy$ and using the invariance of the l_2-norm it follows that

$$\begin{aligned} \|b - Ax\|_2 &= \left\| \begin{pmatrix} b & A \end{pmatrix} \begin{pmatrix} -1 \\ x \end{pmatrix} \right\|_2 = \left\| U^T \begin{pmatrix} b & AV \end{pmatrix} \begin{pmatrix} -1 \\ y \end{pmatrix} \right\|_2 \\ &= \|\beta_1 e_1 - B_n y\|_2. \end{aligned}$$

Hence if y solves the bidiagonal least squares problem

$$\min_y \|B_n y - \beta_1 e_1\|_2, \tag{8.4.39}$$

then $x = Vy$ minimizes $\|Ax - b\|_2$.

The least squares solution to (8.4.39) is obtained by a QR factorization of B_n, which takes the form

$$G_n(B_n \mid \beta_1 e_1) = \left(\begin{array}{c|c} R_n & \begin{matrix} f_k \\ \bar{\phi}_{n+1} \end{matrix} \end{array} \right) = \left(\begin{array}{ccccc|c} \rho_1 & \theta_2 & & & & \phi_1 \\ & \rho_2 & \theta_3 & & & \phi_2 \\ & & \rho_3 & \ddots & & \phi_3 \\ & & & \ddots & \theta_n & \vdots \\ & & & & \rho_n & \phi_n \\ \hline & & & & & \phi_{n+1} \end{array} \right) \tag{8.4.40}$$

where G_n is a product of n Givens rotations. The solution is obtained by back-substitution from $R_n y = d_n$. The norm of the corresponding residual vector equals $|\bar{\phi}_{n+1}|$. To zero out the element β_2 we premultiply rows (1,2) with a rotation G_{12}, giving

$$\begin{pmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{pmatrix} \left(\begin{array}{cc|c} \alpha_1 & 0 & \beta_1 \\ \beta_2 & \alpha_2 & 0 \end{array} \right) = \left(\begin{array}{cc|c} \rho_1 & \theta_2 & \phi_1 \\ 0 & \bar{\rho}_2 & \bar{\phi}_2 \end{array} \right).$$

(Here and in the following only elements affected by the rotation are shown.) Here the elements ρ_1, θ_2 and ϕ_1 in the first row are final, but $\bar{\rho}_2$ and $\bar{\phi}_2$ will be transformed into ρ_2 and ϕ_2 in the next step.

Continuing in this way in step j the rotation $G_{j,j+1}$ is used to zero the element β_{j+1}. In steps, $j = 2 : n-1$, the rows $(j, j+1)$ are transformed

$$\left(\begin{array}{cc} c_j & s_j \\ -s_j & c_j \end{array}\right)\left(\begin{array}{cc|c} \bar{\rho}_j & 0 & \bar{\phi}_j \\ \beta_{j+1} & \alpha_{j+1} & 0 \end{array}\right) = \left(\begin{array}{cc|c} \rho_j & \theta_{j+1} & \phi_j \\ 0 & \bar{\rho}_{j+1} & \bar{\phi}_{j+1} \end{array}\right).$$

where

$$\begin{aligned} &\phi_j = c_j\bar{\phi}_j, \qquad \bar{\phi}_{j+1} = -s_j\bar{\phi}_j, \qquad \rho_j = \sqrt{\bar{\rho}_j^2 + \beta_{j+1}^2}, \\ &\theta_{j+1} = s_j\alpha_{j+1}, \qquad \bar{\rho}_{n+1} = c_j\alpha_{j+1}. \end{aligned}$$

Note that by construction $|\bar{\phi}_{j+1}| \le \bar{\phi}_j$. Finally, in step n we obtain

$$\left(\begin{array}{cc} c_n & s_n \\ -s_n & c_n \end{array}\right)\left(\begin{array}{cc|c} \bar{\rho}_n & & \bar{\phi}_n \\ \beta_{n+1} & & 0 \end{array}\right) = \left(\begin{array}{c|c} \rho_n & \phi_n \\ 0 & \bar{\phi}_{n+1} \end{array}\right).$$

After n steps, we have obtained the factorization (8.4.40) with

$$G_n = G_{n,n+1}\cdots G_{23}G_{12}.$$

Now consider the result after $k < n$ steps of the above bidiagonalization process have been carried out. At this point we have computed $Q_1, Q_2, \ldots, Q_{k+1}$, $P_1, P_2, \ldots, P_k$ such that the first k columns of A are in lower bidiagonal form, i.e.

$$Q_{k+1}\cdots Q_2Q_1\, A\, P_1P_2\cdots P_k\begin{pmatrix} I_k \\ 0 \end{pmatrix} = \begin{pmatrix} B_k \\ 0 \end{pmatrix} = \begin{pmatrix} I_{k+1} \\ 0 \end{pmatrix} B_k,$$

where $B_k \in \mathbf{R}^{(k+1)\times k}$ is a leading submatrix of B_n. Multiplying both sides with $Q_1Q_2\cdots Q_{k+1}$ and using orthogonality we obtain the relation

$$AV_k = U_{k+1}B_k = \hat{B}_k + \beta_{k+1}v_{k+1}e_k^T, \quad k = 1 : n, \tag{8.4.41}$$

where

$$\begin{aligned} &P_1P_2\cdots P_k\begin{pmatrix} I_k \\ 0 \end{pmatrix} = V_k = (v_1, \ldots, v_k), \\ &Q_1Q_2\cdots Q_{k+1}\begin{pmatrix} I_{k+1} \\ 0 \end{pmatrix} = U_{k+1} = (u_1, \ldots, u_{k+1}). \end{aligned}$$

If we consider the intermediate result after applying also P_{k+1} the first $k+1$ rows have been transformed into bidiagonal form, i.e.

$$(\, I_{k+1} \quad 0\,)\, Q_{k+1}\cdots Q_2Q_1\, A\, P_1P_2\cdots P_{k+1} = (\, B_k \quad \alpha_{k+1}e_{k+1}\,)\,(\, I_{k+1} \quad 0\,).$$

Transposing this gives a second relation

$$U_{k+1}^TA = B_kV_k^T + \alpha_{k+1}e_{k+1}v_{k+1}^T, \tag{8.4.42}$$

We now show that the bidiagonalization can be stopped prematurely if a zero element occurs in B. Assume first that the first zero element to occur is $\alpha_{k+1} = 0$. Then we have obtained the decomposition

$$\tilde{U}_{k+1}^T A \tilde{V}_k = \begin{pmatrix} B_k & 0 \\ 0 & A_k \end{pmatrix},$$

where $A_k \in \mathbf{R}^{(m-k-1)\times(n-k)}$, and

$$\tilde{U}_{k+1} = Q_{k+1} \cdots Q_2 Q_1, \qquad \tilde{V}_k = P_1 P_2 \cdots P_k,$$

are square orthogonal matrices. Then, setting $x = \tilde{V}_k y$, the transformed least squares problem takes the form

$$\min_y \left\| \begin{pmatrix} B_k & 0 \\ 0 & A_2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} \beta_1 e_1 \\ 0 \end{pmatrix} \right\|_2, \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \tag{8.4.43}$$

$y_1 \in \mathbf{R}^k$, $y_2 \in \mathbf{R}^{n-k}$. This problem is separable and decomposes into two independent subproblems

$$\min_{y_1} \|B_k y_1 - \beta_1 e_1\|_2, \qquad \min_{y_2} \|A_2 y_2\|_2. \tag{8.4.44}$$

By construction B_k has nonzero elements in its two diagonals. Thus it has full column rank and the solution y_1 to the first subproblem is unique. Further, the minimum norm solution of the initial problem is obtained simply by taking $y_2 = 0$. We call the first subproblem (8.4.44) a **core subproblem**. It can be solved by QR factorization exactly as outlined for the full system when $k = n$.

When $\beta_{k+1} = 0$ is the first zero element to occur then the reduced problem has a similar separable form similar to (8.4.44). The core subproblem is now

$$\hat{B}_k y_1 = \beta_1 e_1, \quad \hat{B}_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \end{pmatrix} \in \mathbf{R}^{k\times k}. \tag{8.4.45}$$

Here $\hat{B}_k$ is square and lower triangular, and the solution y_1 is obtained by forward substitution. Taking $y_2 = 0$ the corresponding residual $b - AVy$ is zero and hence the original system $Ax = b$ is consistent.

We give two simple examples of when premature termination occurs. First assume that $b \perp \mathcal{R}(A)$. Then the reduction will terminate with $\alpha_1 = 0$. The core system is empty and $x = Vy_2 = 0$ is the minimal norm least squares solution.

If the bidiagonalization instead terminates with $\beta_2 = 0$, then the system $Ax = b$ is consistent and the minimum norm solution equals

$$x = (\beta_1/\alpha_1)v_1, \quad v_1 = V_1 e_1 = P_1 e_1 4.$$

Paige and Strakoš [46] have shown the following important properties of the core subproblem obtained by the bidiagonalization algorithm:

Theorem 8.4.5.

Assume that the bidiagonalization of $(\,b \quad A\,)$ *terminates prematurely with* $\alpha_k = 0$ *or* $\beta_{k+1} = 0$. *Then the core corresponding subproblem* (8.4.44) *or* (8.4.45) *is minimally dimensioned. Further, the singular values of the core matrix* B_k *or* $\hat{B}_k$, *are simple and the right hand side* βe_1 *has nonzero components in along each left singular vector.*

Proof. Sketch: The minimal dimension is a consequence of the uniqueness of the decomposition (8.4.36), as long as no zero element in B appears. That the matrix $\hat{B}_k$ has simple singular values follows from the fact that all subdiagonal elements are nonzero. The same is true for the square bidiagonal matrix $(B_k\ 0)$ and therefore also for B_k. Finally, if βe_1 did not have nonzero components along a left singular vector, then the reduction must have terminated earlier. For a complete proof we refer to [46].) □

In many applications the numerical rank of the matrix A is much smaller than $\min\{m, n\}$. For example, in multiple linear regression often some columns are nearly linearly dependent. Then one wants to express the solution by restricting it to lie in a lower dimensional subspace. This can be achieved by neglecting small singular values of A and using a truncated SVD solution; see Sec. 8.4.1. In **partial least squares** (PLS) method this is achieved by a partial bidiagonalization of the matrix $(\,b \quad A\,)$. It is known that PLS often gives a faster reduction of the residual than TSVD.

We remark that the solution steps can be interleaved with the reduction to bidiagonal form. This makes it possible to compute a sequence of *approximate solutions* $x_k = P_1 P_2 \cdots P_k y_k$, where $y_k \in \mathbf{R}^k$ solves

$$\min_y \|\beta_1 e_1 - B_k y\|_2, \quad k = 1, 2, 3, \ldots. \tag{8.4.46}$$

After each (double) step in the bidiagonalization we advance the QR decomposition of B_k. The norm of the least squares residual corresponding to x_k is then given by

$$\|b - Ax_k\|_2 = |\bar{\phi}_{k+1}|.$$

The sequence of residual norms is nonincreasing. We stop and accept $x = V_k y_k$ as an approximate solution of the original least squares problem. if this residual is sufficiently small. This method is called the **Partial Least Squares** (PLS) method in statistics.

The sequential method outlined here is mathematically equivalent to a method called LSQR, which is a method of choice for solving *sparse* linear least squares. LSQR uses a Lanczos-type process for the bidiagonal reduction, which works only with the original sparse matrix. A number of important properties of the successive approximations x_k in PLS are best discussed in connection with LSQR; see Sec. 10.6.4.

Review Questions

1. When and why should column pivoting be used in computing the QR factorization of a matrix? What inequalities will be satisfied by the elements of R if the standard column pivoting strategy is used?
2. Show that the singular values and condition number of R equal those of A. Give a simple lower bound for the condition number of A in terms of its diagonal elements. Is it advisable to use this bound when no column pivoting has been performed?
3. Give a simple lower bound for the condition number of A in terms of the diagonal elements of R. Is it advisable to use this bound when no column pivoting has been performed?
4. What is meant by a Rank-revealing QR factorization? Does such a factorization always exist?
5. How is the *numerical* rank of a matrix A defined? Give an example where the numerical rank is not well determined.

Problems

1. (a) Describe how the QR factorizations of a matrix of the form

$$\begin{pmatrix} A \\ \mu D \end{pmatrix}, \quad A \in \mathbf{R}^{m\times n},$$

where $D \in \mathbf{R}^{n\times n}$ is diagonal, can be computed using Householder transformations in mn^2 flops.

(b) Estimate the number of flops that are needed for the reduction using Householder transformations in the special case that $A = R$ is upper triangular? Devise a method using Givens rotations for this special case!

Hint: In the Givens method zero one diagonal at a time in R working from the main diagonal inwards.

2. Let the vector v, $\|v\|_2 = 1$, satisfy $\|Av\|_2 = \epsilon$, and let Π be a permutation such that

$$|w_n| = \|w\|_\infty, \qquad \Pi^T v = w.$$

(a) Show that if R is the R factor of $A\Pi$, then $|r_{nn}| \le n^{1/2}\epsilon$.

Hint: Show that $\epsilon = \|Rw\|_2 \ge |r_{nn}w_n|$ and then use the inequality $|w_n| = \|w\|_\infty \ge n^{-1/2}\|w\|_2$.

(b) Show using (a) that if $v = v_n$, the right singular vector corresponding to the smallest singular value $\sigma_n(A)$, then

$$\sigma_n(A) \ge n^{-1/2}|r_{nn}|.$$

4. Consider a nonsingular 2×2 upper triangular matrix and its inverse

$$R = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix}, \qquad R^{-1} = \begin{pmatrix} a^{-1} & a^{-1}bc^{-1} \\ 0 & c^{-1} \end{pmatrix}.$$

(a) Suppose we want to choose Π to *maximize* the $(1,1)$ element in the QR factorization of $R\Pi$. Show that this is achieved by taking $\Pi = I$ if $|a| \geq \sqrt{b^2 + c^2}$, else $\Pi = \Pi_{12}$, where Π_{12} interchanges columns 1 and 2.

(b) Unless $b = 0$ the permutation chosen in (a) may not *minimize* the (2,2) element in the QR factorization of $R\Pi$. Show that this is achieved by taking $\Pi = I$ if $|c^{-1}| \geq \sqrt{a^{-2} + b^2(ac)^{-2}}$ else $\Pi = \Pi_{12}$. Hence, the test compares *row* norms in R^{-1} instead of *column* norms in R.

6. To minimize $\|x\|_2$ is not always a good way to resolve rank deficiency, and therefore the following generalization of problem (8.4.13) is often useful: For a given matrix $B \in \mathbf{R}^{p\times n}$ consider the problem

$$\min_{x\in S} \|Bx\|_2, \qquad S = \{x \in \mathbf{R}^n \mid \|Ax - b\|_2 = \min\}.$$

(a) Show that this problem is equivalent to

$$\min_{x_2} \|(BC)x_2 - (Bd)\|_2,$$

where C and d are defined by (8.4.12).

(b) Often one wants to choose B so that $\|Bx\|_2$ is a measure of the smoothness of the solution x. For example one can take B to be a discrete approximation to the second derivative operator,

$$B = \begin{pmatrix} 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & \ddots & \ddots & \ddots \\ & & & 1 & -2 & 1 \end{pmatrix} \in \mathbf{R}^{(n-2)\times n}.$$

Show that provided that $\mathcal{N}(A) \cap \mathcal{N}(B) = \emptyset$ this problem has a unique solution, and give a basis for $\mathcal{N}(B)$.

5. Let $A \in \mathbf{R}^{m\times n}$ with $\operatorname{rank}(A) = r$. A rank revealing LU factorizations of the form

$$\Pi_1 A \Pi_2 = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \end{pmatrix},$$

where Π_1 and Π_2 are permutation matrices and $L_{11}, U_{11} \in \mathbf{R}^{r\times r}$ are triangular and nonsingular can also be used to compute pseudo-inverse solutions $x = A^{\dagger}b$. Show, using Theorem 8.1.7 that

$$A^{\dagger} = \Pi_2 \begin{pmatrix} I_r & S \end{pmatrix}^{\dagger} U_{11}^{-1} L_{11}^{-1} \begin{pmatrix} I_r \\ T \end{pmatrix}^{\dagger} \Pi_1,$$

where $T = L_{21}L_{11}^{-1}$, $S = U_{11}^{-1}U_{12}$. (Note that S is empty if $r = n$, and T empty if $r = m$.)

6. Consider the block upper-bidiagonal matrix

$$A = \begin{pmatrix} B_1 & C_1 & \\ & B_2 & C_2 \\ & & B_3 \end{pmatrix}$$

Outline an algorithm for computing the QR factorization of A, which treats one block row at a time. (It can be assumed that A has full column rank.) Generalize the algorithm to an arbitrary number of block rows!

7. (a) Suppose that we have computed the pivoted QR factorization of A,

$$Q^T A\Pi = \begin{pmatrix} R \\ 0 \end{pmatrix} \in \mathbf{R}^{m\times n},$$

of a matrix $A \in \mathbf{R}^{m\times n}$. Show that by postmultiplying the *upper* triangular matrix R by a sequence of Householder transformations we can transform R into a *lower* triangular matrix $L = RP \in \mathbf{R}^{n\times n}$ and that by combining these two factorizations we obtain

$$Q^T A\Pi P = \begin{pmatrix} L \\ 0 \end{pmatrix}. \tag{8.4.47}$$

This factorization, introduced by G. W. Stewart, who calls it the **QLP decomposition** of A.

(b) Show that the total cost for computing the QLP decomposition is roughly $2mn^2 + 2n^3/3$ flops. How does that compare with the cost for computing the bidiagonal decomposition of A?

(c) Show that the two factorizations can be interleaved. What is the cost for performing the first k steps?

8. Work out the details of an algorithm for transforming a matrix $A \in \mathbf{R}^{m\times n}$ to *lower* bidiagonal form. Consider both cases when $m > n$ and $m \le n$.

Hint: It can be derived by applying the algorithm for transformation to upper bidiagonal form to A^T.

8.5 Some Structured Least Squares Problems

8.5.1 Banded Least Squares Problems

We now consider orthogonalization methods for the special case when A is a banded matrix of row bandwidth w, see Definition 8.2.3. From Theorem 8.2.4 we know that the matrix A^TA will also be a banded matrix with only the first $r = w - 1$ superdiagonals nonzero. Since the factor R in the QR factorization equals the unique Cholesky factor of A^TA it will have only w nonzeros in each row.

Even though the *final* factor R is independent of the row ordering in A, the intermediate fill-in will vary. For banded rectangular matrices the QR factorization can be obtained efficiently by sorting the rows of A and suitably subdividing the Householder transformations. The rows of A should be sorted by leading entry

order (i.e., increasing minimum column subscript order) That is, if $f_i, i = 1, 2, \ldots, m$ denotes the column indices of the first nonzero element in row i we should have,

$$i \le k \Rightarrow f_i \le f_k.$$

Such a band matrix can then be written as

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_q \end{pmatrix}, \quad q \le n,$$

is said to be in **standard form**. where in block A_i the first nonzero element of each row is in column i. The Householder QR process is then applied to the matrix in q major steps. In the first step a QR decomposition of the first block A_1 is computed, yielding R_1. Next at step k, $k = 2 : q$, R_{k-1} will be merged with A_k yielding

$$Q_k^T \begin{pmatrix} R_{k-1} \\ A_k \end{pmatrix} = R_k.$$

Since the rows of block A_k has their first nonzero elements in column k, the first $k-1$ rows of R_{k-1} will not be affected. The matrix Q can be implicitly represented in terms of the Householder vectors of the factorization of the subblocks. This sequential Householder algorithm, which is also described in [38, Ch. 27], requires $(m + 3n/2)w(w + 1)$ multiplications or about twice the work of the less stable Cholesky approach. For a detailed description of this algorithm, see Lawson and Hanson [38, Ch. 11].

In Sec. 4.6.4 we considered the interpolation of a function f where with a linear combination of $m + k$ B-splines of degree k, see (4.6.18), on $\Delta = \{x_0 < x_1 < \cdots < x_m\}$. Assume that we are given function values $f_j = f(\tau_j)$, where $\tau_1 < \tau_2 < \ldots < \tau_n$ are distinct points and $n \ge m + k$. Then we consider the least squares approximation problem

$$\min \sum_{j=1}^{n} e_j^2, \qquad e_j = w_i \Big(f_j - \sum_{i=-k}^{m-1} c_i B_{i,k+1}(\tau_j) \Big). \tag{8.5.1}$$

where w_j are positive weights. This is an overdetermined linear system for c_i, $i = -k, \ldots, m-1$. The elements of its coefficient matrix $B_{i,k+1}(\tau_j)$ can be evaluated by the recurrence (4.6.19). The coefficient matrix has a band structure since in the jth row the ith element will be zero if $\tau_j \notin [x_i, x_{i+k+1}]$. It can be shown, see de Boor [1978, p. 200], that the coefficient matrix will have full rank equal to $m + k$ if and only if there is a subset of points τ_j satisfying

$$x_{j-k-1} < \tau_j < x_j, \qquad \forall j = 1, 2, \ldots, m + k. \tag{8.5.2}$$

Example 8.5.1.

The least squares approximation of a discrete set of data by a linear combination of cubic B-splines gives rise to a banded linear least squares problem. Let

$$s(t) = \sum_{j=1}^{n} x_j B_j(t),$$

where $B_j(t)$, $j = 1 : n$ are the normalized cubic B-splines, and let (y_i, t_i), $i = 1 : m$ be given data points. If we determine x to minimize

$$\sum_{i=1}^{m} (s(t_i) - y_i)^2 = \|Ax - y\|_2^2,$$

then A will be a banded matrix with $w = 4$. In particular if $m = 13$, $n = 8$ the matrix may have the form shown in Fig. 8.4.2. Here A consists of blocks A_k^T, $k = 1 : 7$. In the Fig. 8.4.2 we also show the matrix after the first three blocks have been reduced by Householder transformations $H_1, \ldots, H_9$. Elements which have been zeroed by H_j are denoted by j and fill-in elements by $+$. In step $k = 4$ only the indicated part of the matrix is involved.

$$\begin{pmatrix}
\times & \times & \times & \times & & & & \\
1 & \times & \times & \times & + & & & \\
1 & 2 & \times & \times & + & + & & \\
 & 3 & 4 & \times & \times & + & & \\
 & 3 & 4 & 5 & \times & + & & \\
 & & 6 & 7 & 8 & \times & & \\
 & & 6 & 7 & 8 & 9 & & \\
 & & 6 & 7 & 8 & 9 & & \\
 & & & \times & \times & \times & \times & \\
 & & & \times & \times & \times & \times & \\
 & & & & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times \\
 & & & & \times & \times & \times & \times
\end{pmatrix}$$

Figure 8.5.1. *A banded rectangular matrix A after $k = 3$ steps in the QR reduction.*

In the algorithm the Householder transformations can also be applied to one or several right hand sides b to produce

$$c = Q^T b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, \quad c_1 \in \mathbf{R}^n.$$

The least squares solution is then obtained from $Rx = c_1$ by back-substitution. The vector c_2 is not stored but used to accumulate the residual sum of squares $\|r\|_2^2 = \|c_2\|_2^2$.

It is also possible to perform the QR factorization by treating one row at a time using Givens' rotations. Each step then is equivalent to updating a full triangular matrix formed by columns $f_i(A)$ to $l_i(A)$. Further, if the matrix A is in standard form the first $f_i(A)$ rows of R are already finished at this stage. The reader is encouraged to work through Example 8.5.1 below in order to understand how the algorithm proceeds!

8.5.2 Two-Block Least Squares Problems

In many least squares problem $\min_x \|Ax - b\|_2^2$, $A \in \mathbf{R}^{m\times n}$ the unknowns can be naturally partitioned into two groups,

$$\min_{x_1,x_2} \left\| \begin{pmatrix} A_1 & A_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - b \right\|_2, \tag{8.5.3}$$

with n_1 and n_2 components, respectively, $n = n_1 + n_2$. Assume that the matrix $A = \begin{pmatrix} A_1 & A_2 \end{pmatrix}$ has full column rank.

Let $P_{\mathcal{R}(A_1)}$ be the orthogonal projection onto $\mathcal{R}(A_1)$. For any x_2 we can split the vector $b - A_2x_2 = r_1 + r_2$ into two orthogonal components

$$r_1 = P_{\mathcal{R}(A_1)}(b - A_2x_2), \quad r_2 = (I - P_{\mathcal{R}(A_1)})(b - A_2x_2).$$

Then the problem (8.5.3) takes the form

$$\min_{x_1,x_2} \left\| (A_1x_1 - r_1) - P_{\mathcal{N}(A_1^T)}(b - A_2x_2) \right\|_2. \tag{8.5.4}$$

Here, since $r_1 \in \mathcal{R}(A_1)$ the variables x_1 can always be chosen so that $A_1x_1 - r_1$. It follows that x_2 is the solution to the **reduced least squares problem**

$$\min_{x_2} \|P_{\mathcal{N}(A_1^T)}(A_2x_2 - b)\|_2. \tag{8.5.5}$$

When this reduced problem has been solved for x_2 the unknowns x_1 can be computed from the least squares problem

$$\min_{x_1} \|A_1x_1 - (b - A_2x_2)\|_2. \tag{8.5.6}$$

Sometimes it may be advantageous to carry out a **partial** QR factorization, where only the first $k < n$ columns are orthogonalized. Suppose that after k steps of MGS, we have computed the partial factorization

$$(A, b) = (Q_k, A^{(k+1)}, b^{(k+1)}) \begin{pmatrix} R_{11} & R_{12} & z_k \\ 0 & I & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

where R_{11} is nonsingular. Then we can decompose the residual as $r = b - Ax = r_1 + r_2$, $r_1 \perp r_2$, where

$$r_1 = Q_k(z_k - R_{12}x_2 - R_{11}x_1), \qquad r_2 = b^{(k+1)} - A^{(k+1)}x_2.$$

Then x_2 is the solution to the reduced least squares problem

$$\min_{x_2} \|b^{(k+1)} - A^{(k+1)}x_2\|_2.$$

With x_2 known x_1 can been computed by back-substitution from

$$R_{11}x_1 = z_k - R_{12}x_2.$$

In some applications, e.g., when A has block angular structure (see Sec. 8.5.4), it may be preferable instead not to save R_{11} and R_{12} and instead to refactorize A_1 and solve (8.5.6) for x_1.

8.5.3 Block Triangular Form of a Rectangular Matrix

An arbitrary rectangular matrix $A \in \mathbf{R}^{m\times n}$, $m \geq n$, can by row and column permutations be brought into the block triangular form

$$PAQ = \begin{pmatrix} A_h & U_{hs} & U_{hv} \\ & A_s & U_{sv} \\ & & A_v \end{pmatrix}, \tag{8.5.7}$$

where the diagonal block A_h is underdetermined (i.e., has more columns than rows), A_s is square and A_v is overdetermined (has more rows than columns), and all three blocks have a nonzero diagonal; see the example in Figure 7.6.4. The submatrices A_v and A_h^T both have the strong Hall property. The off-diagonal blocks denoted by U are possibly nonzero matrices of appropriate dimensions. This block triangular form (8.5.7) of a sparse matrix is based on a canonical decomposition of bipartite graphs.

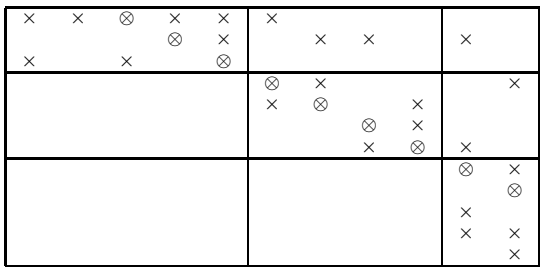


Figure 8.5.2. *The coarse block triangular decomposition of A.*

We call the decomposition of A into the submatrices A_h, A_s, and A_v the **coarse decomposition**. One or two of the diagonal blocks may be absent in the coarse decomposition. It may be possible to further decompose the diagonal blocks in (8.5.7) to obtain the **fine decompositions** of these submatrices. Each of the blocks A_h and A_v may be further decomposable into block diagonal form,

$$A_h = \begin{pmatrix} A_{h1} & & \\ & \ddots & \\ & & A_{hp} \end{pmatrix}, \quad A_v = \begin{pmatrix} A_{v1} & & \\ & \ddots & \\ & & A_{vq} \end{pmatrix},$$

where each $A_{h1}, \ldots, A_{hp}$ is underdetermined and each $A_{v1}, \ldots, A_{vq}$ is overdetermined. The submatrix A_s may be decomposable in block upper triangular form

$$A_s = \begin{pmatrix} A_{s1} & U_{12} & \ldots & U_{1,t} \\ & A_{s2} & \ldots & U_{2,t} \\ & & \ddots & \vdots \\ & & & A_{st} \end{pmatrix} \tag{8.5.8}$$

with square diagonal blocks $A_{s1}, \ldots, A_{st}$ which have nonzero diagonal elements. The resulting decomposition can be shown to be essentially unique. Any one block triangular form can be obtained from any other by applying row permutations that involve the rows of a single block row, column permutations that involve the columns of a single block column, and symmetric permutations that reorder the blocks.

An algorithm for the more general block triangular form described above due to Pothen and Fan depends on the concept of matchings in bipartite graphs. The algorithm consists of the following steps:

1. Find a maximum matching in the bipartite graph $G(A)$ with row set R and column set C.
2. According to the matching, partition R into the sets VR, SR, HR and C into the sets VC, SC, HC corresponding to the horizontal, square, and vertical blocks.
3. Find the diagonal blocks of the submatrix A_v and A_h from the connected components in the subgraphs $G(A_v)$ and $G(A_h)$. Find the block upper triangular form of the submatrix A_s from the strongly connected components in the associated directed subgraph $G(A_s)$, with edges directed from columns to rows.

The reordering to block triangular form in a preprocessing phase can save work and intermediate storage in solving least squares problems. If A has structural rank equal to n, then the first block row in (8.5.7) must be empty, and the original least squares problem can after reordering be solved by a form of block back-substitution. First compute the solution of

$$\min_{\tilde{x}_v} \|A_v \tilde{x}_v - \tilde{b}_v\|_2, \tag{8.5.9}$$

where $\tilde{x} = Q^T x$ and $\tilde{b} = Pb$ have been partitioned conformally with PAQ in (8.5.7). The remaining part of the solution $\tilde{x}_k, \ldots, \tilde{x}_1$ is then determined by

$$A_{si}\tilde{x}_i = \tilde{b}_i - \sum_{j=i+1}^{k} U_{ij}\tilde{x}_j, \quad i = k, \ldots, 2, 1. \tag{8.5.10}$$

Finally, we have $x = Q\tilde{x}$. We can solve the subproblems in (8.5.9) and (8.5.10) by computing the QR decompositions of A_v and $A_{s,i}$, $i = 1, \ldots, k$. Since $A_{s1}, \ldots, A_{sk}$ and A_v have the strong Hall property the structures of the matrices R_i are correctly predicted by the structures of the corresponding normal matrices.

If the matrix A has structural rank less than n, then we have an underdetermined block A_h. In this case we can still obtain the form (8.5.8) with a square block A_{11} by permuting the extra columns in the first block to the end. The least squares solution is then not unique, but a unique solution of minimum length can be found as outlined in Section 2.7.

8.5.4 Block Angular Least Squares Problems

There is often a substantial similarity in the structure of many large scale sparse least squares problems. In particular, the problem can often be put in the following bordered block diagonal or **block angular form**:

$$A = \left(\begin{array}{cccc|c} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_M & B_M \end{array}\right), \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \\ x_{M+1} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{pmatrix}, \tag{8.5.11}$$

where

$$A_i \in \mathbf{R}^{m_i \times n_i}, \quad B_i \in \mathbf{R}^{m_i \times n_{M+1}}, \quad i = 1, 2, \ldots, M,$$

and

$$m = m_1 + m_2 + \cdots + m_M, \qquad n = n_1 + n_2 + \cdots + n_{M+1}.$$

Note that the variables $x_1, \ldots, x_M$ are coupled only to the variables x_{M+1}, which reflects a "local connection" structure in the underlying physical problem. Applications where the form (8.5.11) arises naturally include photogrammetry, Doppler radar and GPS positioning, and geodetic survey problems.

The normal matrix of A in (8.5.11) is of doubly bordered block diagonal form,

$$A^T A = \left(\begin{array}{cccc|c} A_1^T A_1 & & & & A_1^T B_1 \\ & A_2^T A_2 & & & A_2^T B_2 \\ & & \ddots & & \vdots \\ & & & A_M^T A_M & A_M^T B_M \\ \hline B_1^T A_1 & B_2^T A_2 & \cdots & B_M^T A_M & C \end{array}\right),$$

where

$$C = \sum_{k=1}^{M} B_k^T B_k = R_{M+1}^T R_{M+1},$$

and R_{M+1} is the Cholesky factor of C. We assume in the following that $\operatorname{rank}(A) = n$, which implies that the matrices $A_i^T A_i$, $i = 1, 2, \ldots, M$, and C are positive definite. It is easily seen that then the Cholesky factor R of $A^T A$ will have a block

structure similar to that of A,

$$R = \left(\begin{array}{cccc|c} R_1 & & & & S_1 \\ & R_2 & & & S_2 \\ & & \ddots & & \vdots \\ & & & R_M & S_M \\ \hline & & & & R_{M+1} \end{array}\right) \tag{8.5.12}$$

where $R_i \in \mathbf{R}^{n_i \times n_i}$, the Cholesky factor of $A_i^T A_i$, is nonsingular and

$$S_i = (A_i R_i^{-1})^T B_i, \quad i = 1, \ldots, M+1.$$

An algorithm for least squares problems of block angular form based on QR factorization of A proceeds in the following three steps:

1. For $i = 1, 2, \ldots, M$ reduce the diagonal block A_i to upper triangular form by a sequence of orthogonal transformations applied to (A_i, B_i) and the right-hand side b_i, yielding

$$Q_i^T(A_i, B_i) = \begin{pmatrix} R_i & S_i \\ 0 & T_i \end{pmatrix}, \qquad Q_i^T b_i = \begin{pmatrix} c_i \\ d_i \end{pmatrix}.$$

   It is usually advantageous to continue the reduction in step 1 so that the matrices T_i, $i = 1, \ldots, M$, are brought into upper trapezoidal form.

2. Set

$$T = \begin{pmatrix} T_1 \\ \vdots \\ T_M \end{pmatrix}, \qquad d = \begin{pmatrix} d_1 \\ \vdots \\ d_M \end{pmatrix}$$

   and compute the QR decomposition

$$\tilde{Q}_{M+1}^T \begin{pmatrix} T & d \end{pmatrix} = \begin{pmatrix} R_{M+1} & c_{M+1} \\ 0 & d_{M+1} \end{pmatrix}.$$

   The solution to $\min_{x_{M+1}} \|Tx_{M+1} - d\|_2$ is then obtained from the triangular system

$$R_{M+1} x_{M+1} = c_{M+1},$$

   and the residual norm is given by $\rho = \|d_{M+1}\|_2$.

3. For $i = M, \ldots, 1$ compute $x_M, \ldots, x_1$ by back-substitution in the triangular systems

$$R_i x_i = c_i - S_i x_{M+1}.$$

In steps 1 and 3 the computations can be performed in parallel on the M subsystems. There are alternative ways to organize this algorithm. Note that when

x_{M+1} has been computed in step 2, then the vectors x_i, $i = 1, \ldots, M$, solves the least squares problem

$$\min_{x_i} \|A_i x_i - g_i\|_2, \quad g_i = b_i - B_i x_{M+1}.$$

Hence it is possible to discard the R_i, S_i and c_i in step 1 if the QR factorizations of A_i are recomputed in step 3. In some practical problems this modification can reduce the storage requirement by an order of magnitude, while the recomputation of R_i may only increase the operation count by a few percent.

Using the structure of the R-factor in (8.5.12), the diagonal blocks of the variance-covariance matrix $C = (R^T R)^{-1} = R^{-1} R^{-T}$ can be written

$$\begin{aligned} &C_{M+1,M+1} = R_{M+1}^{-1} R_{M+1}^{-T}, \\ &C_{i,i} = R_i^{-1}(I + W_i^T W_i) R_i^{-T}, \quad W_i^T = S_i R_{M+1}^{-1}, \quad i = 1, \ldots, M. \end{aligned} \tag{8.5.13}$$

If we compute the QR decompositions

$$Q_i \begin{pmatrix} W_i \\ I \end{pmatrix} = \begin{pmatrix} U_i \\ 0 \end{pmatrix}, \quad i = 1, \ldots, M,$$

we have $I + W_i^T W_i = U_i^T U_i$ and then

$$C_{i,i} = (U_i R_i^{-T})^T (U_i R_i^{-T}), \quad i = 1, \ldots, M.$$

This assumes that all the matrices R_i and S_i have been retained.

8.5.5 Kronecker Product Problems

Sometimes least squares problems occur which have a highly regular block structure. Here we consider least squares problems of the form

$$\min_x \|(A \otimes B)x - d\|_2, \tag{8.5.14}$$

where the $A \otimes B$ is the **Kronecker product** of $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{p \times q}$. This product is the $mp \times nq$ block matrix,

$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}.$$

Problems of Kronecker structure arise in several application areas including signal and image processing, photogrammetry, and multidimensional approximation. It applies to least squares fitting of multivariate data on a rectangular grid. Such problems can be solved with great savings in storage and operations. Since often the size of the matrices A and B is large, resulting in models involving several hundred thousand equations and unknowns, such savings may be essential.

We recall from Sec. 7.7.3 the important rule (7.7.14) for the inverse of a Kronecker product

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}.$$

It follows that if P and Q are orthogonal $n \times n$ matrices then $P \otimes Q$ is an orthogonal $n^2 \times n^2$ matrix. This rule for the inverse holds also for pseudo-inverses.

Lemma 8.5.1.
Let $A^\dagger$ and $B^\dagger$ be the pseudo-inverses of A and B. Then

$$(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger.$$

Proof. The theorem follows by verifying that $X = A^\dagger \otimes B^\dagger$ satisfies the four Penrose conditions in (8.1.11)–(8.1.12). □

Using Lemmas 7.7.6 and 8.5.1 the solution to the Kronecker least squares problem (8.5.14) can be written

$$x = (A \otimes B)^\dagger \mathrm{vec}\, C = (A^\dagger \otimes B^\dagger)\mathrm{vec}\, C = \mathrm{vec}\,(B^\dagger C (A^\dagger)^T). \tag{8.5.15}$$

This formula leads to a great reduction in the cost of solving Kronecker least squares problems. For example, if A and B are both $m \times n$ matrices, the cost of computing is reduced from $O(m^2 n^4)$ to $O(mn^2)$.

In some areas the most common approach to computing the least squares solution to (8.5.14) is to use the normal equations. If we assume that both A and B have full column rank, then we can use the expressions

$$A^\dagger = (A^T A)^{-1} A^T, \qquad B^\dagger = (B^T B)^{-1} B^T.$$

However, because of the instability associated with the explicit formation of $A^T A$ and $B^T B$, an approach based on orthogonal decompositions should generally be preferred. If we have computed the complete QR decompositions of A and B,

$$A\Pi_1 = Q_1 \begin{pmatrix} R_1 & 0 \\ 0 & 0 \end{pmatrix} V_1^T, \qquad B\Pi_2 = Q_2 \begin{pmatrix} R_2 & 0 \\ 0 & 0 \end{pmatrix} V_2^T,$$

with R_1, R_2 upper triangular and nonsingular, then from Section 2.7.3 we have

$$A^\dagger = \Pi_1 V_1 \begin{pmatrix} R_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q_1^T, \qquad B^\dagger = \Pi_2 V_2 \begin{pmatrix} R_2^{-1} & 0 \\ 0 & 0 \end{pmatrix} Q_2^T.$$

These expressions can be used in (8.5.15) to compute the pseudo-inverse solution of problem (8.5.14) even in the rank deficient case.

We finally note that the singular values and singular vectors of the Kronecker product $A \otimes B$ can be simply expressed in terms of the singular values and singular vectors of A and B.

Lemma 8.5.2. *Let A and B have the singular value decompositions*

$$A = U_1\Sigma_1 V_1^T, \qquad B = U_2\Sigma_2 V_2^T.$$

Then we have

$$A \otimes B = (U_1 \otimes U_2)(\Sigma_1 \otimes \Sigma_2)(V_1 \otimes V_2)^T.$$

Proof. The proof follows from Lemma 8.5.1. □

Review Questions

1. What is meant by the standard form of a banded rectangular matrix A? Why is it important that a banded matrix is permuted into standard form before its orthogonal factorization is computed?
2. In least squares linear regression the first column of A often equals the vector $a_1 = e = (1, 1, \ldots, 1)^T$ (cf. Example 8.2.1). Setting $A = (\, e \quad A_2\,)$, show that performing one step in MGS is equivalent to "subtracting out the means".

Problems

1. Consider the two-block least squares problem (8.5.3). Work out an algorithm to solve the reduced least squares problem $\min_{x_2} \|P_{\mathcal{N}(A_1^T)}(A_2x_2 - b)\|_2$ using the method of normal equations.

   Hint: First show that $P_{\mathcal{N}(A_1^T)}(A) = I - A_1(R_1^TR_1)^{-1}A_1^T$, where R_1 is the Cholesky factor of $A_1^TA_1$.
2. (a) Suppose we want to fit two set of points $(x_i, y_i) \in \mathbf{R}^2$, $i = 1, \ldots, p$, and $i = p+1, \ldots, m$, to two *parallel* lines

   $$cx + sy = h_1, \qquad cx + sy = h_2, \quad c^2 + s^2 = 1,$$

   so that the sum of orthogonal distances are minimized. Generalize the approach of Example 8.6.3 to sketch an algorithm for solving this problem.

   (b) Modify the algorithm in (a) to fit two *orthogonal* lines.
3. Use the Penrose conditions to prove the formula

   $$(A \otimes B)^\dagger = A^\dagger \otimes B^\dagger,$$

   where $\otimes$ denotes the Kronecker product

8.6 Generalized Least Squares

8.6.1 Generalized Least Squares

Let $A \in \mathbf{R}^{m\times n}$, $m \geq n$, and let $B \in \mathbf{R}^{m\times m}$ be symmetric positive semidefinite. Augmented linear systems of the form

$$\begin{pmatrix} B & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} s \\ x \end{pmatrix} = \begin{pmatrix} b \\ c \end{pmatrix} \tag{8.6.1}$$

in (8.6.1) occur in many application areas since they represent the condition for equilibrium of a physical system. The system (8.6.1) is often called a **saddle-point system** or, in optimization, a KKT (Karush–Kuhn–Tucker) system. The system matrix in (8.6.1) is symmetric but in general indefinite; it is nonsingular if and only if

1. A has full column rank;
2. the matrix $(B\ A)$ has full row rank.

A unified formulation of generalized least squares and minimum norm problems can be obtained as follows.

Theorem 8.6.1. *If B is positive definite then the linear system (8.6.1) is nonsingular and gives the condition for the solution of the two problems:*

$$\min_x \tfrac{1}{2}\|Ax - b\|^2_{B^{-1}} + c^T x, \tag{8.6.2}$$

$$\min_s \tfrac{1}{2}\|s - b\|_B, \quad \textit{subject to} \quad A^T s = c, \tag{8.6.3}$$

where $\|x\|^2_G = x^T G x$ for any symmetric positive definite matrix G.

Proof. If B is symmetric positive definite so is B^{-1}. The system (8.6.1) can be obtained by differentiating (8.6.2) to give

$$A^T B^{-1}(Ax - b) + c = 0,$$

and setting $s = B^{-1}(b - Ax)$. The system can also be obtained by differentiating the Lagrangian

$$L(x, s) = \frac{1}{2}s^T B s - s^T b + x^T(A^T s - c)$$

of (8.6.3), and equating to zero. Here x is the vector of Lagrange multipliers. □

Remark: Theorem 8.6.1 can be generalized to to the semidefinite case, see Gulliksson and Wedin [31, Theorem 3.2]. A case when B is indefinite and nonsingular is considered in Sec. 8.6.4

If we take $c = 0$ in Theorem 8.6.1, then the solution x gives the best linear unbiased estimate for the linear model

$$Ax + \epsilon = b, \quad \mathcal{V}(\epsilon) = \sigma^2 B^{-1}.$$

The standard linear least squares problem (8.1.1) is obtained by taking $B = I$.

Taking $B = I$ in problem (8.6.3), we have $s = r = b - Ax$ and this problem becomes

$$\min_s \frac{1}{2}\|r - b\|_2, \quad \text{subject to} \quad A^T r = c, \tag{8.6.4}$$

i.e. to find the point s closest to b in the set of solutions to the underdetermined linear system $A^T r = c$. This problem frequently occurs as a subproblem in linearly constrained optimization. Another application, for which $c = 0$, is in structural optimization, where A^T is called the equilibrium matrix, B the element flexibility matrix, y is the force, and x a Lagrange multiplier vector.

There are two different approaches to the solution of systems of the form (8.6.1). In the **range space method** the y variables are eliminated to obtain the **generalized normal equations**

$$A^T B^{-1} A x = A^T B^{-1} b - c. \tag{8.6.5}$$

From the assumptions in Theorem 8.6.1, it follows that the matrix $A^T B^{-1} A$ is symmetric, positive definite. The normal equations can be solved for x, and then y obtained by solving $Bs = b - Ax$. Setting $B = W$ and $c = 0$ in (8.6.2) gives a weighted linear least squares problem; see Sec. 8.6.2.

$$A^T B^{-1} A x = A^T B^{-1} b - c, \qquad y = B^{-1}(b - Ax). \tag{8.6.6}$$

For $B = I$ the first equation in (8.6.6) is the normal equations for the least squares problem. If B is positive definite then one way to solve these equations is to compute the Cholesky factorization $B = R^T R$ and then solve

$$\min_x \|R^{-1}(Ax - b)\|_2 \tag{8.6.7}$$

using the QR factorization of $R^{-1}A$. However, a more stable approach is to use a generalized QR (GQR) factorization of the matrix pair A, B; to be described in Sec. 8.6.3.

Using (8.6.5) the solution to problem (8.6.4) can be written

$$r = b - Ax = P_{\mathcal{N}(A^T)} b + A(A^T A)^{-1} c. \tag{8.6.8}$$

In particular, taking $b = 0$, this is the **minimum norm solution** of the system $A^T y = c$.

In the **null space method** the solution y to (8.6.6) is split as

$$y = y_1 + y_2, \quad y_1 \in \mathcal{R}(A), \quad y_2 \in \mathcal{N}(A^T). \tag{8.6.9}$$

Let y_1 be the minimum norm solution of $A^T y = c$. This can be computed using the QR factorization of A. If we set $Q = (\, Q_1 \quad Q_2 \,)$ then

$$y_1 = Q_1 z_1, \quad z_1 = R^{-T} c.$$

Next y_2 is obtained by solving the reduced system

$$Q_2^T B Q_2 z_2 = Q_2^T (b - B y_1), \qquad y_2 = Q_2 z_2. \tag{8.6.10}$$

Finally, form

$$y = Q_1 z_1 + Q_2 z_2 \quad \text{and} \quad x = R^{-1} Q_1^T (b - Vy).$$

In the special case that $B = I$ the generalized least squares problems associated with (8.6.1) simplify to

$$\min_y \|y - b\|_2^2 \quad \text{subject to} \quad A^T y = c, \tag{8.6.11}$$

$$\min_x \{\|b - Ax\|_2^2 + 2c^T x\}. \tag{8.6.12}$$

When a Householder QR factorization is available the algorithm is as follows:

$$z = R^{-T} c, \qquad \begin{pmatrix} d \\ f \end{pmatrix} = Q^T b, \qquad r = Q \begin{pmatrix} z \\ f \end{pmatrix}, \qquad x = R^{-1}(d - z).$$

Assuming that the matrix $M = A^T B^{-1} A$ has rank n, a first order perturbation analysis for the generalized least squares problem can be obtained. We assume that B is not perturbed and for simplicity take $c = 0$. Proceeding as in Sec. 8.1.5, we denote the perturbed data $A + \delta A$ and $b + \delta b$ and the perturbed solution $x + \delta x$ and $s + \delta s$.

The perturbed solution satisfies the system

$$\begin{pmatrix} B & A + \delta A \\ (A + \delta A)^T & 0 \end{pmatrix} \begin{pmatrix} \tilde{s} \\ \tilde{x} \end{pmatrix} = \begin{pmatrix} b + \delta b \\ 0 \end{pmatrix}. \tag{8.6.13}$$

Subtracting the equations (8.6.13) and neglecting second order quantities the perturbations $\delta s = B^{-1}\delta r$ and δx satisfy the linear system

$$\begin{pmatrix} B & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} \delta s \\ \delta x \end{pmatrix} = \begin{pmatrix} \delta b - \delta A x \\ -\delta A^T s \end{pmatrix}. \tag{8.6.14}$$

From the Schur–Banachiewicz formula (see Sec. 7.1.5) it follows that the inverse of the matrix in this system equals

$$\begin{pmatrix} B & A \\ A^T & 0 \end{pmatrix}^{-1} = \begin{pmatrix} (I - B^{-1}AM^{-1}A^T)B^{-1} & B^{-1}AM^{-1} \\ M^{-1}A^T B^{-1} & -M^{-1} \end{pmatrix}. \tag{8.6.15}$$

where $M = A^T B^{-1} A$ is the negative Schur complement. Hence we obtain

$$\delta x = M^{-1} A^T B^{-1} (\delta b - \delta A\, \tilde{x}) + M^{-1} \delta A^T \tilde{s}, \tag{8.6.16}$$

$$\delta r = (B - AM^{-1}A^T) B^{-1} (\delta b - \delta A\, \tilde{x}) - AM^{-1} \delta A^T \tilde{s}, \tag{8.6.17}$$

Taking norms in (8.6.17) and (8.6.17) we obtain

$$\|\delta x\|_2 \lessapprox \|M^{-1} A^T B^{-1}\| \left(\|\delta b\| + \|\delta A\|\, \|x\|\right) + \|M^{-1}\|\, \|\delta A\|\, \|s\|, \tag{8.6.18}$$

$$\|\delta r\| \lessapprox \|(B - AM^{-1}A^T)\|\, \|B^{-1}\| (\|\delta b\| + \|\delta A\|\, \|x\|) \tag{8.6.19}$$

$$+ \|AM^{-1}\|\, \|\delta A\| \|s\|, \tag{8.6.20}$$

8.6.2 Weighted Problems

We now consider a simple special case of the generalized least squares problem. In the general univariate linear model (8.1.5) the covariance matrix W is a positive diagonal matrix

$$W = \sigma^2 \mathrm{diag}\,(w_1, w_2, \ldots, w_m) > 0.$$

The corresponding problem then is the **weighted** linear least squares problem (8.1.5)

$$\min_x \|D(Ax - b)\|_2, \quad D = W^{-1/2} = \mathrm{diag}\,(d_1, d_2, \ldots, d_m). \tag{8.6.21}$$

When the ith component of the error vector in the linear model has small variance then $d_i = 1/\sqrt{w_{ii}}$ will be large. In the limit when some d_i tend to infinity, the corresponding ith equation becomes a linear constraint.

We assume in the following that the matrix A is row equilibrated, that is,

$$\max_{1 \le j \le n} |a_{ij}| = 1, \quad i = 1 : m.$$

and that the rows of A are ordered so that the weights satisfy

$$\infty > d_1 \ge d_2 \ge \cdots \ge d_m > 0. \tag{8.6.22}$$

We call a weighted least squares problems **stiff** if $d_1 \gg d_m$; see Example 8.2.2. For stiff problems the condition number $\kappa(DA)$ will be large. An upper bound is given by

$$\kappa(DA) \le \kappa(D)\kappa(A) = \gamma\kappa(A).$$

It is important to note that this does *not* mean that the problem of computing x from given data $\{D, A, b\}$ necessarily is ill-conditioned. Problems with extremely ill-conditioned weight matrices arise, e.g., in electrical networks, certain classes of finite element problems, and interior point methods for constrained optimization.

In many cases it is possible to compute $\tilde{A} = DA$, $\tilde{b} = Db$ and solve this as a standard least squares problem

$$\min_x \|\tilde{A}x - \tilde{b}\|_2.$$

However, if the weights $d_1, \ldots, d_m$ vary widely in magnitude this is not in general a numerically stable approach. Special care may be needed in solving stiff weighted linear least squares problems. In general the method of normal equations is not well suited for solving stiff problems. To illustrate this, we consider the important special case where only the first p equations are weighted:

$$\min_x \left\| \begin{pmatrix} \gamma A_1 \\ A_2 \end{pmatrix} x - \begin{pmatrix} \gamma b_1 \\ b_2 \end{pmatrix} \right\|_2^2, \tag{8.6.23}$$

$A_1 \in \mathbf{R}^{p \times n}$ and $A_2 \in \mathbf{R}^{(m-p) \times n}$. Such problems occur, for example, when the method of weighting is used to solve least squares problems with the linear equality

constraints $A_1x = b_1$; see Section 5.1.4. For this problem the matrix of normal equations becomes

$$B = (\,\gamma A_1^T \quad A_2^T\,)\begin{pmatrix}\gamma A_1\\ A_2\end{pmatrix} = \gamma^2 A_1^T A_1 + A_2^T A_2.$$

If $\gamma > u^{-1/2}$ (u is the unit roundoff) and $A_1^T A_1$ is dense, then $B = A^T A$ will be completely dominated by the first term and the data contained in A_2 may be lost. However, if the number p of very accurate observations is less than n, then the solution depends critically on the less precise data in A_2. (The matrix in Example 2.2.1 is of this type.) We conclude that for weighted least squares problems with $\gamma \gg 1$ the method of normal equations generally is not well behaved.

We now consider the use of methods based on the QR decomposition of A for solving weighted problems. We first examine the Householder QR method, and show by an example that this method can give poor accuracy for stiff problems unless the algorithm is extended to include *row interchanges*.

Example 8.6.1.

Consider the least squares problem ([51]) with

$$A = \begin{pmatrix}0 & 2 & 1\\ \gamma & \gamma & 0\\ \gamma & 0 & \gamma\\ 0 & 1 & 1\end{pmatrix}, \qquad b = \begin{pmatrix}3\\ 2\gamma\\ 2\gamma\\ 2\end{pmatrix}.$$

The exact solution is equal to $x = (1, 1, 1)$. Using exact arithmetic we obtain after the first step of QR decomposition of A by Householder transformations the reduced matrix

$$\tilde{A}^{(2)} = \begin{pmatrix}\frac{1}{2}\gamma - 2^{1/2} & -\frac{1}{2}\gamma - 2^{-1/2}\\ -\frac{1}{2}\gamma - 2^{1/2} & \frac{1}{2}\gamma - 2^{-1/2}\\ 1 & 1\end{pmatrix}.$$

If $\gamma > u^{-1}$ the terms $-2^{1/2}$ and $-2^{-1/2}$ in the first and second rows are lost. However, this is equivalent to the loss of all information present in the first row of A. This loss is disastrous because the number of rows containing large elements is less than the number of components in x, so there is a substantial dependence of the solution x on the first row of A. (However, compared to the method of normal equations, which fails already when $\gamma > u^{-1/2}$, this is an improvement!)

The Householder algorithm can be extended to include *row interchanges*. In each step a pivot column is first selected in the reduced matrix, and then the element of largest absolute value in the pivot column is permuted to the top. The resulting algorithm has good stability properties for stiff problems as well.

There is no need to perform row pivoting in Householder QR, provided that an initial **row sorting** is performed, where the rows are sorted after decreasing so that

$$\max_j |a_{1j}| \geq \max_j |a_{1j}| \geq \cdots \geq \max_j |a_{1j}|. \tag{8.6.24}$$

For example, in Example 8.6.1 the two large rows will be permuted to the top of the matrix A. The Householder algorithm then works well without any further row interchanges.

The stability of row sorting has been shown by Cox and Higham [18]. Note that row sorting has the advantage over row pivoting in that after sorting the rows any library routine can be used for the QR factorization. In particular this allows for the use of BLAS 3 subroutines, which is not the case for row pivoting.

It is also essential that *column pivoting* is performed when QR decomposition is used for weighted problems. To illustrate the need for column pivoting, consider an example of the form (8.6.23), where

$$A_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \end{pmatrix},$$

Then stability is lost without column pivoting because the first two columns of the matrix A_1 are linearly dependent.

When column pivoting is introduced this difficulty disappears. With QR factorization with **complete pivoting** we will mean that both row sorting (or row pivoting) and column pivoting is used.

Another suitable transformation for weighted problems is to make a preliminary LU factorization of the matrix A. If the problem has the form (8.6.23) with $\operatorname{rank}(A_1) = p$, then p steps of Gaussian elimination are performed on the weighted system using row and column pivoting. The resulting factorization can be written

$$\Pi_r \begin{pmatrix} \gamma A_1 \\ A_2 \end{pmatrix} \Pi_c = LDU, \tag{8.6.25}$$

where Π_r and Π_c are permutation matrices, and

$$L = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \in \mathbf{R}^{m\times n}, \quad U = \begin{pmatrix} U_{11} & U_{12} \\ & I \end{pmatrix} \in \mathbf{R}^{n\times n}.$$

Here $L_{11} \in \mathbf{R}^{p\times p}$ is unit lower triangular, and $U_{11} \in \mathbf{R}^{p\times p}$ is unit upper triangular. Assuming that A has full rank, D is nonsingular. Then (4.4.1) is equivalent to

$$\min_y \|Ly - \Pi_r b\|_2, \qquad DU\Pi_c^T x = y.$$

The least squares problem in y is usually well-conditioned, since any ill-conditioning from the weights is usually reflected in D. We illustrate the method in a simple example. For a fuller treatment of weighted and the general linear model, see Björck [11, Chap.3].

Example 8.6.2. In Example 8.2.2 it was shown that the method of normal equations can fail. After multiplication with $\gamma = \epsilon^{-1}$ this becomes

$$A = \begin{pmatrix} 1 & 1 & 1 \\ \epsilon & & \\ & \epsilon & \\ & & \epsilon \end{pmatrix}, \qquad b = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

which is of the form (8.6.23) with $p = 1$. After one step of Gaussian elimination we have the factorization $A = LDU$, where

$$L = \begin{pmatrix} 1 & & \\ \epsilon & -1 & -1 \\ & 1 & \\ & & 1 \end{pmatrix}, \qquad D = \begin{pmatrix} 1 & & \\ & \epsilon & \\ & & \epsilon \end{pmatrix}, \qquad U = \begin{pmatrix} 1 & 1 & 1 \\ & 1 & \\ & & 1 \end{pmatrix}.$$

As is easily verified L and U are well-conditioned. Setting $DUx = y$, the solution can be accurately computed by first solving the normal equations $L^T Ly = L^T b$ for y and then finding x by back-substitution and scaling. □

8.6.3 Generalized Orthogonal Decompositions

The motivation for introducing different generalizations of orthogonal decompositions is basically to avoid the explicit computation of matrix products and quotients of matrices. For example, let A and B be square and nonsingular matrices and assume we need the SVD of AB^{-1} (or AB). Then the explicit calculation of AB^{-1} (or AB) may result in a loss of precision and should be avoided.

Consider a pair of matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times p}$. The generalized QR (GQR) decomposition of A and B is written

$$A = QR, \qquad B = QTZ, \tag{8.6.26}$$

where $Q \in \mathbb{R}^{m\times m}$ and $Z \in \mathbb{R}^{p\times p}$ are orthogonal matrices and R and T have one of the forms

$$R = \begin{pmatrix} R_{11} \\ 0 \end{pmatrix} \quad (m \geq n), \qquad R = \begin{pmatrix} R_{11} & R_{12} \end{pmatrix} \quad (m < n), \tag{8.6.27}$$

and

$$T = \begin{pmatrix} 0 & T_{12} \end{pmatrix} \quad (m \leq p), \qquad T = \begin{pmatrix} T_{11} \\ T_{21} \end{pmatrix} \quad (m > p). \tag{8.6.28}$$

If B is square and nonsingular GQR implicitly gives the QR factorization of $B^{-1}A$. There is also a similar generalized RQ factorization related to the QR factorization of AB^{-1}. Routines for computing a GQR decomposition of are included in LAPACK. These decompositions allow the solution of very general formulations of several least squares problems.

8.6.4 Indefinite Least Squares

The indefinite least squares problem (ILS) has the form

$$\min_x (b - Ax)^T J(b - Ax), \tag{8.6.29}$$

where $A \in \mathbf{R}^{m\times n}$, $m \geq n$, and $b \in \mathbf{R}^m$ are given and J is a **signature matrix**, i.e. a diagonal matrix with elements equal to ± 1. In the following we assume for

simplicity that

$$J = \begin{pmatrix} I_p & 0 \\ 0 & -I_q \end{pmatrix}, \quad p+q = m, \tag{8.6.30}$$

While the standard least squares is obtained if $p = 0$ or $q = 0$, for $pq \neq 0$ the problem is to minimize an indefinite quadratic form.

The normal equations

$$A^T J(b - Ax) = 0 \tag{8.6.31}$$

give first order conditions for optimality. In the following we assume that the Hessian matrix $A^T JA$ is positive definite. Then the ILS problem has a unique solution.

Chandrasekaran, Gu and Sayed [16] proposed a QR-Cholesky method for solving the ILS problem. It uses a QR factorization

$$A = QR = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} R, \quad Q_1 \in \mathbf{R}^{p\times n}, \quad Q_2 \in \mathbf{R}^{q\times n}. \tag{8.6.32}$$

Then

$$A^T JA = R^T (Q_1^T Q_1 - Q_2^T Q_2) R,$$

and it follows that R is nonsingular and $Q_1^T Q_1 - Q_2^T Q_2$ is positive definite. Using (8.6.32) the normal equations (8.6.31) can be written

$$(Q_1^T Q_1 - Q_2^T Q_2) Rx = Q^T Jb. \tag{8.6.33}$$

Using the Cholesky factorization $Q_1^T Q_1 - Q_2^T Q_2 = U^T U$, this becomes

$$U^T URx = Q^T Jb,$$

which can be solved by one forward and two backward substitutions.

The operation count for the QR–Cholesky algorithm is approximately $n^2(5m - n)$, which can be compared to the normal equations $n^2(m + n/3)$.

Sometimes it is useful to consider **hyperbolic rotations** $\breve{G}$ of the form

$$\breve{G} = \begin{pmatrix} c & -s \\ -s & c \end{pmatrix}, \quad c = \cosh\theta, \quad s = \sinh\theta, \tag{8.6.34}$$

and $c^2 - s^2 = 1$. The matrix $\breve{G}$ is S-orthogonal, $\breve{G}^T S\breve{G} = I$, for the signature matrix $S = \operatorname{diag}(1, -1)$.

A hyperbolic rotation can be used to zero a selected component in a vector. Provided that $|\alpha| > |\beta|$ and

$$s = \beta/\alpha, \quad c = \sqrt{(1+s)(1-s)}, \quad \sigma = \alpha c,$$

we have $s^2 + c^2 = 1$ and

$$\breve{G}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \frac{1}{c}\begin{pmatrix} 1 & -s \\ -s & 1 \end{pmatrix}\begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \end{pmatrix}.$$

A matrix representing a rotation a hyperbolic rotation in $\mathbf{R}^m$ in the plane spanned by the unit vectors e_i and e_j, $i < j$, is obtained as for Givens' rotations; see (8.3.34).

Thee condition number of $\breve{G}$ in (8.6.34) is not bounded, and to form a product $\breve{G}x$ the straightforward way is not numerically stable. Instead we note the equivalence of

$$\breve{G}\begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} \hat{x}_i \\ \hat{x}_j \end{pmatrix}, \qquad G\begin{pmatrix} \hat{x}_i \\ x_j \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}\begin{pmatrix} \hat{x}_i \\ x_j \end{pmatrix} = \begin{pmatrix} x_i \\ \hat{x}_j \end{pmatrix}, \tag{8.6.35}$$

where G is an orthogonal Givens rotation. The mixed method where $\hat{x}_i$ is determined from the hyperbolic rotation and then $\hat{x}_j$ from the equivalent Givens rotation

$$\hat{x}_i = (x_i - sx_j)/c, \qquad \hat{x}_j = -s\hat{x}_i + cx_j, \tag{8.6.36}$$

has been shown to be numerically more stable.

A hyperbolic QR factorization method for solving the indefinite least squares problem has been devised by Bojanczyk, Higham and Patel [13]. We first use Householder transformations to compute the factorization

$$\begin{pmatrix} Q_1^T & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} R_1 \\ 0 \\ A_2 \end{pmatrix}.$$

where A has been partitioned conformally with J.

8.6.5 Orthogonal Regression

We consider here the following **orthogonal regression** problem. Let $y_i \in \mathbf{R}^n$, $i = 1 : m$, be $m > n$ given points. We want to determine a hyperplane M in $\mathbf{R}^n$ such that the sum of squares of the orthogonal distances from the given points to M is minimized. The equation for the hyperplane can be written

$$c^T z = h, \quad z, c \in \mathbf{R}^n, \quad \|c\|_2 = 1,$$

where $c \in \mathbf{R}^n$ is the normal vector of M, and $|h|$ is the orthogonal distance form the origin to the plane. Then the orthogonal projections of the points y_i onto M are given by

$$z_i = y_i - (c^T y_i - h)c. \tag{8.6.37}$$

It is readily verified that the point z_i lies on M and the residual $(z_i - y_i)$ is parallel to c and hence orthogonal to M. It follows that the problem is equivalent to minimizing

$$\sum_{i=1}^{m}(c^T y_i - h)^2, \quad \text{subject to} \quad \|c\|_2 = 1.$$

If we put $Y = (y_1, \ldots, y_m) \in \mathbf{R}^{n\times m}$ and $e = (1, \ldots, 1)^T$, this problem can be written in matrix form

$$\min_{c,h} \left\| \begin{pmatrix} Y^T & -e \end{pmatrix}\begin{pmatrix} c \\ h \end{pmatrix} \right\|_2, \quad \text{subject to} \quad \|c\|_2 = 1. \tag{8.6.38}$$

For a fixed c, this expression is minimized when the residual vector $(Y^Tc - he)$ is orthogonal to e, that is $e^T(Y^Tc - he) = e^TY^Tc - he^Te = 0$. Since $e^Te = m$ it follows that

$$h = \frac{1}{m}c^TYe = c^T\bar{y}, \qquad \bar{y} = \frac{1}{m}Ye, \tag{8.6.39}$$

where $\bar{y}$ is the mean value of the given points y_i. Hence h is determined by the condition that the mean value $\bar{y}$ lies on the optimal plane M.

We now subtract the mean value $\bar{y}$ from the each given point, and form the matrix

$$\bar{Y} = (\bar{y}_1, \ldots, \bar{y}_m), \qquad \bar{y}_i = y_i - \bar{y}, \quad i = 1, \ldots, m.$$

Since by (8.6.39)

$$(Y^T, -e)\begin{pmatrix} c \\ h \end{pmatrix} = Y^Tc - e\bar{y}^Tc = (Y^T - e\bar{y}^T)c = \bar{Y}^Tc,$$

problem (8.6.38) is equivalent to

$$\min_c \|\bar{Y}^Tc\|_2, \qquad \|c\|_2 = 1 \tag{8.6.40}$$

By the min-max characterization of the singular values (Theorem 8.1.11) a solution to (8.6.40) is $c = v_n$, where v_n is a right singular vector of $\bar{Y}^T$ corresponding to the singular value σ_n. Hence, a solution to problem (8.6.38) is given by

$$c = v_n, \qquad h = v_n^T\bar{y}, \qquad \sum_{i=1}^{m}(v_n^Ty_i - h)^2 = \sigma_n,$$

The fitted points $z_i \in M$ are obtained from

$$z_i = \bar{y}_i - (v_n^T\bar{y}_i)v_n + \bar{y},$$

i.e., by first orthogonalizing the shifted points $\bar{y}_i$ against v_n, and then adding the mean value back.

Note that in contrast to the TLS problem the orthogonal regression problem always has a solution. The solution is unique when $\sigma_{n-1} > \sigma_n$, and the minimum sum of squares equals σ_n^2. We have $\sigma_n = 0$, if and only if the given points y_i, $i = 1, \ldots, m$ all lie on the hyperplane M. In the extreme case, all points coincide and then $\bar{Y} = 0$, and any plane going through $\bar{y}$ is a solution.

The above method solves the problem of fitting a $(n-1)$ dimensional linear manifold to a given set of points in R. It is readily generalized to the fitting of an $(n-p)$ dimensional manifold by orthogonalizing the shifted points y against the p left singular vectors of Y corresponding to p smallest singular values. A least squares problem that often arises is to fit to given data points a geometrical element, which may be defined in implicit form. For example, the problem of fitting circles, ellipses, spheres, and cylinders arises in applications such as computer graphics, coordinate meteorology, and statistics. Such problems are nonlinear and will be discussed in Sec. 11.4.7.

Example 8.6.3.

Suppose we want to fit by orthogonal regression m pair of points $(x_i, y_i) \in \mathbf{R}^2$, $i = 1, \ldots, m$, to a straight line

$$cx + sy = h, \quad c^2 + s^2 = 1.$$

First compute the mean values of x_i and y_i and the QR factorization of the matrix of shifted points

$$\bar{Y}^T = \begin{pmatrix} \bar{x}_1 & \bar{y}_1 \\ \bar{x}_2 & \bar{y}_2 \\ \vdots & \vdots \\ \bar{x}_m & \bar{y}_m \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

where R is an upper triangular 2×2 matrix. Since the singular values and right singular vectors of $\bar{Y}^T$ and R are the same, it suffices to compute the SVD

$$R = \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix} = (u_1 \ u_2) \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \end{pmatrix},$$

where $\sigma_1 \geq \sigma_2 \geq 0$. (A stable algorithm for computing the SVD of an upper triangular matrix is given in Algorithm 9.4.2; see also Problem 9.4.5.) Then the coefficients in the equation of the straight line are given by

$$(c \ s) = v_2^T, \qquad h = v_2^T \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix}.$$

If $\sigma_2 = 0$ but $\sigma_1 > 0$ the matrix $\bar{Y}$ has rank one. In this case the given points lie on a straight line. If $\sigma_1 = \sigma_2 = 0$, then $\bar{Y} = 0$, and $x_i = \bar{x}$, $y_i = \bar{y}$ for all $i = 1, \ldots, m$. Note that u_2 is uniquely determined if and only if $\sigma_1 \neq \sigma_2$. It is left to the reader to discuss the case $\sigma_1 = \sigma_2 \neq 0$!

8.6.6 Linear Equality Constraints

In some least squares problems in which the unknowns are required to satisfy a system of linear equations exactly. One source of such problems is in curve and surface fitting, where the curve is required to interpolate certain data points.

Given matrices $A \in \mathbf{R}^{m \times n}$ and $B \in \mathbf{R}^{p \times n}$ we consider the problem **LSE** to find a vector $x \in \mathbf{R}^n$ which solves

$$\min_x \|Ax - b\|_2 \quad \text{subject to} \quad Bx = d. \tag{8.6.41}$$

A solution to problem (8.6.41) exists if and only if the linear system $Bx = d$ is consistent. If $\operatorname{rank}(B) = p$ then B has linearly independent rows, and $Bx = d$ is consistent for any right hand side d. A solution to problem (8.6.41) is unique if and only if the null spaces of A and B intersect only trivially, i.e., if $\mathcal{N}(A) \cap \mathcal{N}(B) = \{0\}$, or equivalently

$$\operatorname{rank} \begin{pmatrix} A \\ B \end{pmatrix} = n. \tag{8.6.42}$$

If (8.6.42) is not satisfied then there is a vector $z \neq 0$ such that $Az = Bz = 0$. Hence if x solves (8.6.41) then $x + z$ is a different solution. In the following we therefore assume that $\operatorname{rank}(B) = p$ and that (8.6.42) is satisfied.

A robust algorithm for problem LSE should check for possible inconsistency of the constraints $Bx = d$. If it is not known a priori that the constraints are consistent, then problem LSE may be reformulated as a sequential least squares problem

$$\min_{x \in S} \|Ax - b\|_2, \qquad S = \{x \mid \|Bx - d\|_2 = \min\ \}. \tag{8.6.43}$$

The most natural way to solve problem LSE is to derive an equivalent unconstrained least squares problem of lower dimension. There are basically two different ways to perform this reduction: **direct elimination** and **the null space method**. We describe both these methods below.

In the method of direct elimination we start by reducing the matrix B to upper trapezoidal form. It is essential that column pivoting is used in this step. In order to be able to solve also the more general problem (8.6.43) we will compute a QR factorization of B. By Theorem 8.4.1 (see next section) there is an orthogonal matrix $U \in \mathbf{R}^{p \times p}$ and a permutation matrix Π_B such that

$$Q_B^T B \Pi_B = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}, \qquad R_{11} \in \mathbf{R}^{r \times r}, \tag{8.6.44}$$

where $r = \operatorname{rank}(B) \leq p$ and R_{11} is upper triangular and nonsingular. Using this factorization, and setting $\bar{x} = \Pi_B^T x$, the constraints become

$$(R_{11}, R_{12})\bar{x} = R_{11}\bar{x}_1 + R_{12}\bar{x}_2 = \bar{d}_1, \qquad \bar{d} = Q_B^T d = \begin{pmatrix} \bar{d}_1 \\ \bar{d}_2 \end{pmatrix}, \tag{8.6.45}$$

where $\bar{d}_2 = 0$ if and only if the constraints are consistent. If we apply the permutation Π_B also to the columns of A and partition the resulting matrix conformally with (8.6.44), $\bar{A}\Pi_B = (A_1, A_2)$. then $Ax - b = A_1\bar{x}_1 + A_2\bar{x}_2 - b$. Solving (8.6.45) for $\bar{x}_1 = R_{11}^{-1}(\bar{d}_1 - R_{12}\bar{x}_2)$, and substituting, we find that the unconstrained least squares problem

$$\begin{aligned} &\min_{\bar{x}_2} \|\hat{A}_2\bar{x}_2 - \hat{b}\|_2, \qquad \hat{A}_2 \in \mathbf{R}^{m \times (n-r)} \\ &\hat{A}_2 = \bar{A}_2 - \bar{A}_1 R_{11}^{-1} R_{12}, \qquad \hat{b} = b - \bar{A}_1 R_{11}^{-1}\bar{d}_1. \end{aligned} \tag{8.6.46}$$

is equivalent to the original problem LSE. Here $\hat{A}_2$ is the Schur complement of R_{11} in

$$\begin{pmatrix} R_{11} & R_{12} \\ \bar{A}_1 & \bar{A}_2 \end{pmatrix}.$$

It can be shown that if the condition in (8.6.42) is satisfied, then $\operatorname{rank}(A_2) = r$. Hence the unconstrained problem has a unique solution, which can be computed from the QR factorization of $\hat{A}_2$.

In the null-space method, assuming that $\operatorname{rank}(B) = p$, we compute the QR factorization

$$B^T = U \begin{pmatrix} R_B \\ 0 \end{pmatrix}, \qquad R_B \in \mathbf{R}^{p \times p}, \tag{8.6.47}$$

where R_B is upper triangular and nonsingular. Using Theorem 8.3.7 we find that the general solution of the system $Bx = d$ can be written as

$$x = x_1 + Q_2 y_2, \qquad x_1 = B^{\dagger} d = Q_1 R_B^{-T} d. \tag{8.6.48}$$

where $U = (Q_1, Q_2)$, $Q_1 \in \mathbf{R}^{n\times p}$, and $Q_2 \in \mathbf{R}^{n\times(n-p)}$. (Note that Q_2 gives an orthogonal basis for the null space of B.) Hence, $Ax - b = Ax_1 + AQ_2y_2 - b$, $y_2 \in \mathbf{R}^{n-p}$, and it remains to solve the unconstrained least squares problem

$$\min_{y_2} \|(AQ_2)y_2 - (b - Ax_1)\|_2. \tag{8.6.49}$$

Let $y_2 = (AQ_2)^I(b - Ax_1)$ be the minimum length solution to (8.6.49), and let x be defined by (8.6.48). Then since $x_1 \perp Q_2y_2$ it follows that

$$\|x\|_2^2 = \|x_1\|_2^2 + \|Q_2y_2\|_2^2 = \|x_1\|_2^2 + \|y_2\|_2^2$$

and x is the minimum norm solution to problem LSE.

If (8.6.42) is satisfied it follows that rank $(AQ_2) = n-p$. Then we can compute the QR factorization

$$Q_A^T(AQ_2) = \begin{pmatrix} R_A \\ 0 \end{pmatrix},$$

where R_A is upper triangular and nonsingular. The unique solution to (8.6.49) can then be computed from

$$R_A y_2 = c_1, \qquad c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = Q_A^T(b - Ax_1), \tag{8.6.50}$$

and we finally obtain $x = x_1 + Q_2y_2$, the unique solution to problem LSE.

The method of direct elimination and the null space method both have good numerical stability. The operation count for the method of direct elimination is slightly lower because Gaussian elimination is used to derive the reduced unconstrained problem.

Review Questions

1. What is meant by a saddle-point system? Which two optimization problems give rise to saddle-point systems?
2. Show the equivalence of the hyperbolic and the Givens rotations in (8.6.35).

Problems

1. Consider the overdetermined linear system $Ax = b$ in Example 8.2.2. Assume that $\epsilon^2 \le u$, where u is the unit roundoff, so that $fl(1 + \epsilon^2) = 1$.

(a) Show that the condition number of A is $\kappa = \epsilon^{-1}\sqrt{3+\epsilon^2} \approx \epsilon^{-1}\sqrt{3}$.

(b) Show that if no other rounding errors are made then the maximum deviation from orthogonality of the columns computed by CGS and MGS, respectively, are

$$\text{CGS}: \quad |q_3^T q_2| = 1/2, \qquad \text{MGS}: \quad |q_3^T q_1| = \frac{\epsilon}{\sqrt{6}} \leq \frac{\kappa}{3\sqrt{3}}u.$$

Note that for CGS orthogonality has been completely lost!

2. Assume that $A \in \mathbf{R}^{m\times m}$ is symmetric and positive definite and $B \in \mathbf{R}^{m\times n}$ a matrix with full column rank. Show that

$$M = \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix},$$

where $S = B^T A^{-1} B$ is the Schur complement (cf. (6.2.12)). Conclude that M is indefinite! (M is called a saddle point matrix.)

8.7 Total Least Squares

8.7.1 The Total Least Squares Problem

In the standard linear model (8.1.3) it is assumed that the vector $b \in \mathcal{R}^m$ is related to the unknown parameter vector $x \in \mathcal{R}^n$ by a linear relation $Ax = b + e$, where $A \in \mathcal{R}^{m\times n}$ is an exactly known matrix and e a vector of random errors. If the components of e are uncorrelated, have zero means and the same variance, then by the Gauss–Markoff theorem (Theorem 8.1.4) the best unbiased estimate of x is obtained by solving the least squares problem

$$\min_x \|r\|_2, \qquad Ax = b + r. \tag{8.7.1}$$

The assumption in the least squares problem that all errors are confined to the right hand side b is frequently unrealistic, and sampling or modeling errors often will affect also the matrix A. In the **errors-in-variables model** it is assumed that a linear relation

$$(A + E)x = b + r,$$

where the rows of the errors $(E,\ r)$ are *independently and identically distributed with zero mean and the same variance*. If this assumption is not satisfied it might be possible to find scaling matrices $D = \text{diag}\,(d_1, \ldots, d_m)$, $T = \text{diag}\,(d_1, \ldots, d_{n+1})$, such that $D(A,\ b)T$ satisfies this assumptions.

Estimates of the unknown parameters x in this model can be obtained from the solution of the **total least squares** (TLS) problem[4]

$$\min_{E,\,r} \|(r,\ E)\|_F, \qquad (A + E)x = b + r, \tag{8.7.2}$$

[4]The term "total least squares problem" was coined by Golub and Van Loan in [28]. The concept has been independently developed in other areas. For example, in statistics this is also known as "latent root regression".

where $\|\cdot\|_F$ denotes the Frobenius matrix norm defined by

$$\|A\|_F^2 = \sum_{i,j} a_{ij}^2 = \text{trace}\,(A^T A).$$

The constraint in (8.7.2) implies that $b + r \in \mathcal{R}(A + E)$. Thus the total least squares is equivalent to the problem of finding the "nearest" compatible linear system, where the distance is measured by the Frobenius norm. If a minimizing perturbation $(E,\ r)$ has been found for the problem (8.7.2) then any x satisfying $(A + E)x = b + r$ is said to solve the TLS problem.

The TLS solution will depend on the scaling of the data $(A,\ b)$. In the following we assume that this scaling has been carried out in advance, so that any statistical knowledge of the perturbations has been taken into account. In particular, the TLS solution depends on the relative scaling of A and b. If we scale x and b by a factor γ we obtain the **scaled TLS problem**

$$\min_{E,\,r} \|(E,\ \gamma r)\|_F \qquad (A + E)x = b + r.$$

Clearly, when γ is small perturbations in b will be favored. In the limit when $\gamma \to 0$ we get the ordinary least squares problem. Similarly, when γ is large perturbations in A will be favored. In the limit when $1/\gamma \to 0$, this leads to the **data least squares** (DLS) problem

$$\min_{E} \|E\|_F, \qquad (A + E)x = b, \tag{8.7.3}$$

where it is assumed that the errors in the data is confined to the matrix A.

8.7.2 Total Least Squares Problem and the SVD

In the following we assume that $b \notin \mathcal{R}(A)$, for otherwise the system is consistent. The constraint in (8.7.2) can be written

$$(\,b + r \quad A + E\,)\begin{pmatrix} -1 \\ x \end{pmatrix} = 0.$$

This constraint is satisfied if the matrix $(b+r\ A+E)$ is rank deficient and $(\,-1 \quad x\,)^T$ lies in its nullspace. Hence the TLS problem involves finding a perturbation matrix having minimal Frobenius norm, which lowers the rank of the matrix $(\,b \quad A\,)$.

The total least squares problem can be analyzed in terms of the SVD

$$(\,b \quad A\,) = U\Sigma V^T = \sum_{i=1}^{k+1} \sigma_i u_i v_i^T, \tag{8.7.4}$$

where $\sigma_1 \geq \ldots \geq \sigma_n \geq \sigma_{n+1} \geq 0$ are the singular values of $(\,b \quad A\,)$. By Theorem 8.1.13 the singular values of $\hat{\sigma}_i$ of A interlace those of $(\,b \quad A\,)$, i.e.,

$$\sigma_1 \geq \hat{\sigma}_1 \geq \sigma_2 > \cdots \geq \sigma_n \geq \hat{\sigma}_n \geq \sigma_{n+1}. \tag{8.7.5}$$

Assume first that rank $(A) = n$ and that $\hat{\sigma}_n > \sigma_{n+1}$, which implies that $\sigma_n > \sigma_{n+1}$. Then by Theorem 8.1.14 the unique perturbation of minimum norm $\|\,(\,r \quad E\,)\,\|_F$ that makes $(A+E)x = b+r$ consistent is the rank one perturbation

$$(\,r \quad E\,) = -\sigma_{n+1} u_{n+1} v_{n+1}^T \tag{8.7.6}$$

for which $\min_{E,\,r} \|\,(\,r \quad E\,)\,\|_F = \sigma_{n+1}$. From (8.7.5), using the orthogonality of the right singular vectors we find that $\sigma_{n+1} u_{n+1} = (\,b \quad A\,)\,v_{n+1}$. Multiplying (8.7.6) from the right with v_{n+1} gives

$$(\,b \quad A\,)\,v_{n+1} = -\,(\,r \quad E\,)\,v_{n+1}. \tag{8.7.7}$$

Writing the relation $(A+E)x = b+r$ in the form

$$(\,b \quad A\,)\begin{pmatrix} 1 \\ -x \end{pmatrix} = -\,(\,r \quad E\,)\begin{pmatrix} 1 \\ -x \end{pmatrix}$$

and comparing with (8.7.7) it is easily seen that the TLS solution can be written in terms of the right singular vector v_{n+1} as

$$x = -\omega^{-1} y, \quad v_{n+1} = \begin{pmatrix} \omega \\ y \end{pmatrix}, \tag{8.7.8}$$

If $\omega = 0$ then the TLS problem has no solution. Note that this is the case if and only if b has no component along u_{n+1}. (This case can only occur when $\hat{\sigma}_n = \sigma_{n+1}$, since otherwise it can be shown that the TLS problem has a unique solution.) In this "nongeneric" case the theory and solution methods become more complicated.

Suppose now that σ_{n+1} is a repeated singular value,

$$\sigma_1 \geq > \cdots \geq \sigma_k \;\; > \hat{\sigma}_{k+1} = \cdots = \sigma_{n+1}.$$

and let $v = V_2 z$ be any unit vector in the subspace $\mathcal{R}(V_2)$, where $V_2 = (v_{k+1}, \ldots, v_{n+1})$ is the matrix consisting of the right singular vectors corresponding to the minimal singular values. Let Q be a Householder transformation such that.

$$V_2 Q = \begin{pmatrix} \omega & 0 \\ y & V_2' \end{pmatrix}$$

Then is $\omega \neq 0$ a TLS solution of minimum norm is given by (8.7.8). Otherwise we have a nongeneric problem.

One way to avoid the complications of nongeneric problems is to compute a regular core TLS problem by bidiagonalizing of the matrix $(\,b \quad A\,)$. This will be discussed in Sec. 8.7.4.

8.7.3 Conditioning of the TLS Problem

We now consider the conditioning of the total least squares problem and its relation to the least squares problem. We denote those solutions by x_{TLS} and x_{LS} respectively.

The TLS solution can also be characterized by

$$\begin{pmatrix} b^T b & b^T A \\ A^T b & A^T A \end{pmatrix} \begin{pmatrix} -1 \\ x_{TLS} \end{pmatrix} = \sigma_{n+1}^2 \begin{pmatrix} -1 \\ x_{TLS} \end{pmatrix}, \tag{8.7.9}$$

i.e., $(\,-1 \quad x_{TLS}\,)^T$ is an eigenvector corresponding to the smallest eigenvalue $\lambda_{n+1} = \sigma_{n+1}^2$ of the "square" of $(\,b \quad A\,)$. This eigenvector is characterized by the property that it minimizes the Rayleigh quotient, that is $\min_x \rho(x) = \sigma_{n+1}^2$, where

$$\rho(x) = \frac{(b - Ax)^T (b - Ax)}{x^T x + 1} = \frac{\|r\|_2^2}{\|x\|_2^2 + 1}, \tag{8.7.10}$$

This also shows that whereas the LS solution minimizes $\|r\|_2$ the TLS solution minimizes $\|r\|_2/(\|x\|_2^2 + 1)^{1/2}$.

From the last block row of (8.7.9) it follows that

$$(A^T A - \sigma_{n+1}^2 I) x_{TLS} = A^T b. \tag{8.7.11}$$

Hence, if we assume that $\hat{\sigma}_n > \sigma_{n+1}$ it follows that the matrix $(A^T A - \sigma_{n+1}^2 I)$ is symmetric positive definite, which ensures that the TLS problem has a unique solution.

This can be compared with the corresponding normal equations for the least squares solution x_{LS},

$$A^T A x_{LS} = A^T b. \tag{8.7.12}$$

In (8.7.11) a positive multiple of the unit matrix is *subtracted* from the matrix $A^T A$ of normal equations. Thus TLS can be considered as a *deregularizing* procedure. (Compare Sec. 8.4.1, where a multiple of the unit matrix was added to improve the conditioning.) Hence the TLS solution is always *worse conditioned* than the LS problem, From a statistical point of view this can be interpreted as removing the bias by subtracting the error covariance matrix (estimated by $\sigma_{n+1}^2 I$ from the data covariance matrix $A^T A$. Subtracting (8.7.12) from (8.7.12) we get

$$x_{TLS} - x_{LS} = \sigma_{n+1}^2 (A^T A - \sigma_{n+1}^2 I)^{-1} x_{LS}.$$

Taking norms we obtain

$$\frac{\|x_{TLS} - x_{LS}\|_2}{\|x_{LS}\|_2} \leq \frac{\sigma_{n+1}^2}{\hat{\sigma}_n^2 - \sigma_{n+1}^2},$$

which shows that when the difference $\hat{\sigma}_n - \sigma_{n+1} \ll \hat{\sigma}_n$ is small then the TLS solution can differ much from the LS solution. It can be shown that an approximate condition number for the TLS solution is

$$\kappa_{TLS} \approx \frac{\hat{\sigma}_1}{\hat{\sigma}_n - \sigma_{n+1}} = \kappa(A) \frac{\hat{\sigma}_n}{\hat{\sigma}_n - \sigma_{n+1}}. \tag{8.7.13}$$

When $\hat{\sigma}_n - \sigma_{n+1} \ll \hat{\sigma}_n$ the TLS condition number can be much worse than for the LS problem.

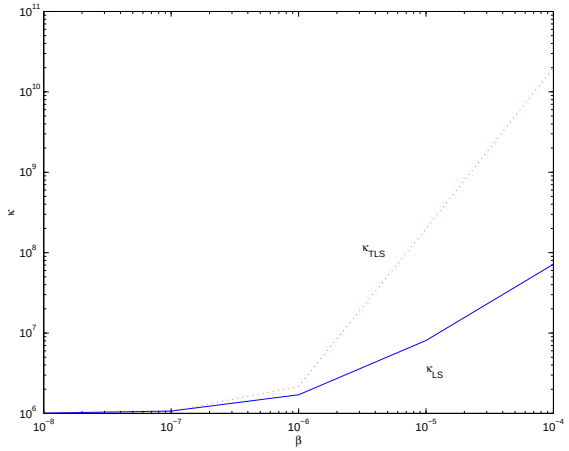


Figure 8.7.1. *Condition numbers κ_{LS} and κ_{TLS} as function of $\beta = \|r_{LS}\|_2$.*

Example 8.7.1.

Consider the overdetermined system

$$\begin{pmatrix} \hat{\sigma}_1 & 0 \\ 0 & \hat{\sigma}_2 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ \beta \end{pmatrix}. \tag{8.7.14}$$

Trivially, the LS solution is $x_{LS} = (c_1/\hat{\sigma}_1, c_2/\hat{\sigma}_2)^T$, $\|r_{LS}\|_2 = |\beta|$. If we take $\hat{\sigma}_1 = c_1 = 1$, $\hat{\sigma}_2 = c_2 = 10^{-6}$, then $x_{LS} = (1,\ 1)^T$ independent of β, and hence does not reflect the ill-conditioning of A. However,

$$\kappa_{LS}(A, b) = \kappa(A) \left(1 + \frac{\|r_{LS}\|_2}{\|\hat{\sigma}_1 x_{LS}\|_2}\right)$$

will increase proportionally to β. The TLS solution is of similar size as the LS solution as long as $|\beta| \leq \hat{\sigma}_2$. However, when $|\beta| \gg \hat{\sigma}_2$ then $\|x_{TLS}\|_2$ becomes large.

In Figure 8.7.1 the two condition numbers are plotted as a function of $\beta \in [10^{-8}, 10^{-4}]$. For $\beta > \hat{\sigma}_2$ the condition number κ_{TLS} grows proportionally to β^2. It can be verified that $\|x_{TLS}\|_2$ also grows proportionally to β^2.

Setting $c_1 = c_2 = 0$ gives $x_{LS} = 0$. If $|\beta| \geq \sigma_2(A)$, then $\sigma_2(A) = \sigma_3(A, b)$ and the TLS problem is nongeneric.

8.7.4 Bidiagonalization and TLS Problems.

Consider the total least squares (TLS) problem

$$\min_{E,r} \|(E,\ r)\|_F, \qquad (A + E)x = b + r.$$

It was shown in Sec. 8.4.5 that we can always find square orthogonal matrices $\tilde{U}_{k+1}$ and $\tilde{V}_k = P_1 P_2 \cdots P_k$, such that

$$\tilde{U}_{k+1}^T \begin{pmatrix} b & A\tilde{V}_k \end{pmatrix} = \begin{pmatrix} \beta_1 e_1 & B_k & 0 \\ 0 & 0 & A_k \end{pmatrix}, \tag{8.7.15}$$

where

$$B_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix} \in \mathbf{R}^{(k+1)\times k},$$

and

$$\beta_j \alpha_j \neq 0, \quad j = 1:k. \tag{8.7.16}$$

Setting $x = \tilde{V}_k \begin{pmatrix} y \\ z \end{pmatrix}$, the approximation problem $Ax \approx b$ then decomposes into the two subproblems

$$B_k y \approx \beta_1 e_1, \qquad A_k z \approx 0.$$

It seems reasonable to simply take $z = 0$, and separately solve the first subproblem, which is the minimally dimensioned **core subproblem**. Setting

$$V_k = \tilde{V}_k \begin{pmatrix} I_k \\ 0 \end{pmatrix}, \qquad U_{k+1} = \tilde{U}_{k+1} \begin{pmatrix} I_{k+1} \\ 0 \end{pmatrix},$$

it follows that

$$\begin{pmatrix} b & AV_k \end{pmatrix} = U_{k+1} \begin{pmatrix} \beta_1 e_1 & B_k \end{pmatrix}.$$

If $x = V_k y \in \mathcal{R}(V_k)$ then

$$(A + E)x = (A + E)V_k y = (U_{k+1} B_k + EV_k)y = \beta_1 U_{k+1} e_1 + r,$$

Hence the consistency relation $(A + E_k)x = b + r$ becomes

$$(B_k + F)y = \beta_1 e_1 + s, \quad F = U_{k+1}^T E V_k, \quad s = U_{k+1}^T r. \tag{8.7.17}$$

Using the orthogonality of U_{k+1} and V_k it follows that

$$\|(E, r)\|_F = \|(F, s)\|_F. \tag{8.7.18}$$

Hence to minimize $\|(E, r)\|_F$ we should take y_k to be the solution to the TLS core subproblem

$$\min_{F,s} \|(F,\ s)\|_F, \qquad (B_k + F)y = \beta_1 e_1 + s. \tag{8.7.19}$$

From (8.7.16) and Theorem 8.4.5 it follows that the singular values of the matrix B_k are simple and that the right hand side βe_1 has nonzero components along each

left singular vector. This TLS problem therefore must have a unique solution. Note that we can assume that $\beta_{k+1} \neq 0$, since otherwise the system is compatible.

To solve this subproblem we need to compute the SVD of the bidiagonal matrix

$$(\beta_1 e_1,\ B_k) = \begin{pmatrix} \beta_1 & \alpha_1 & & & \\ & \beta_2 & \alpha_2 & & \\ & & \beta_3 & \ddots & \\ & & & \ddots & \alpha_k \\ & & & & \beta_{k+1} \end{pmatrix} \in \mathbf{R}^{(k+1)\times(k+1)}. \tag{8.7.20}$$

The SVD of this matrix

$$(\beta_1 e_1,\ B_k) = P\mathrm{diag}\,(\sigma_1, \ldots, \sigma_{k+1})Q^T, \quad P, Q \in \mathbf{R}^{(k+1)\times(k+1)}$$

can be computed, e.g., by the implicit QR-SVD algorithm; see Sec. 9.7.6. (Note that the first stage in this is a transformation to bidiagonal form, so the work in performing the reduction (8.7.15) has not been wasted!) Then with

$$q_{k+1} = Qe_{k+1} = \begin{pmatrix} \omega \\ z \end{pmatrix}.$$

Here it is always the case that $\omega \neq 0$ and the solution to the original TLS problem (8.7.19) equals

$$x_{TLS} = V_k y = -\omega^{-1} V_k z.$$

Further the norm of the perturbation equals

$$\min_{E,r} \|(E, r)\|_F = \sigma_{k+1}.$$

8.7.5 Some Generalized TLS Problems

We now consider the more general TLS problem with $d > 1$ right-hand sides

$$\min_{E,\ F} \| \begin{pmatrix} E & F \end{pmatrix} \|_F, \qquad (A + E)X = B + F, \tag{8.7.21}$$

where $B \in \mathbf{R}^{m\times d}$. The consistency relations can be written

$$\begin{pmatrix} B + F & A + E \end{pmatrix} \begin{pmatrix} -I_d \\ X \end{pmatrix} = 0,$$

Thus we now seek perturbations $(E,\ F)$ that reduces the rank of the matrix $(A,\ B)$ by d. We call this a multidimensional TLS problem. As remarked before, for this problem to be meaningful the rows of the error matrix $(E,\ F)$ should be independently and identically distributed with zero mean and the same variance.

Note that the multidimensional problem is different from solving d one-dimensional TLS problems with right-hand sides $b_1, \ldots, b_d$. This is because in the multidimensional problem we require that the matrix A be similarly perturbed for all right-hand sides. This is in contrast to the usual least squares solution and may lead to improved predicted power of the TLS solution.

The solution to the TLS problem with multiple right-hand sides can be expressed in terms of the SVD

$$(\, B \quad A\,) = (U_1\ U_2) \begin{pmatrix} \Sigma_1 & \\ & \Sigma_2 \end{pmatrix} \begin{pmatrix} V_1^T \\ V_2^T \end{pmatrix}, \tag{8.7.22}$$

where $\Sigma_1 = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$, $\Sigma_2 = \operatorname{diag}(\sigma_{n+1}, \ldots, \sigma_{n+d})$, and U and V are partitioned conformally. The minimizing perturbation is given by

$$(\, F \quad E\,) = -U_2 \Sigma_2 V_2^T = -\,(\, B \quad A\,)\, V_2 V_2^T,$$

for which $\|\,(\, F \quad E\,)\,\|_F = \sum_{j=1}^{d} \sigma_{n+j}^2$ and

$$(\, B + F \quad A + E\,)\, V_2 = 0, \qquad V_2 = \begin{pmatrix} V_{12} \\ V_{22} \end{pmatrix}.$$

where $V_{12} \in \mathbf{R}^{d\times d}$. If $\hat{\sigma}_n > \sigma_{n+1}$, where $\hat{\sigma}_i$, $i = 1, \ldots, n$, are the singular values of A, it can be shown that V_{12} is nonsingular. Then the solution to the TLS problem is unique and given by

$$X = -V_{22} V_{12}^{-1} \in \mathbf{R}^{n\times d}.$$

Otherwise assume that $\sigma_k > \sigma_{k+1} = \cdots = \sigma_{n+1}$, $k \le n$, and set $V_2 = (v_{k+1}, \ldots, v_{n+d})$. Let Q be a product of Householder transformations such that

$$V_2 Q = \begin{pmatrix} \Gamma & 0 \\ Z & Y \end{pmatrix},$$

where $\Gamma \in \mathbf{R}^{d\times d}$ is lower triangular. If Γ is nonsingular, then the TLS solution of minimum norm is given by

$$X = -Z\Gamma^{-1}.$$

In many parameter estimation problems, some of the columns are known exactly. It is no restriction to assume that the error-free columns are in leading positions in A. In the multivariate version of this **mixed LS-TLS problem** one has a linear relation

$$(A_1, A_2 + E_2)X = B + F, \qquad A_1 \in \mathbf{R}^{m\times n_1},$$

where $A = (A_1,\ A_2) \in \mathbf{R}^{m\times n}$, $n = n_1 + n_2$. It is assumed that the rows of the errors $(E_2,\ F)$ are independently and identically distributed with zero mean and the same variance. The mixed LS-TLS problem can then be expressed

$$\min_{E_2, F} \|(E_2,\ F)\|_F, \qquad (A_1,\ A_2 + E_2)X = B + F. \tag{8.7.23}$$

When A_2 is empty, this reduces to solving an ordinary least squares problem. When A_1 is empty this is the standard TLS problem. Hence this mixed problem includes both extreme cases.

The solution of the mixed LS-TLS problem can be obtained by first computing a QR factorization of A and then solving a TLS problem of reduced dimension.

Algorithm 8.7.1 Mixed LS-TLS problem
Let $A = (A_1,\ A_2) \in \mathbf{R}^{m\times n}$, $n = n_1 + n_2$, $m \geq n$, and $B \in \mathbf{R}^{m\times d}$. Assume that the columns of A_1 are linearly independent. Then the following algorithm solves the mixed LS-TLS problem (8.7.23).

Step 1. Compute the QR factorization

$$(A_1,\ A_2,\ B) = Q\begin{pmatrix} R \\ 0 \end{pmatrix}, \quad R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix},$$

where Q is orthogonal, and $R_{11} \in \mathbf{R}^{n_1\times n_1}$, $R_{22} \in \mathbf{R}^{(n_2+d)\times(n_2+d)}$ are upper triangular. If $n_1 = n$, then the solution X is obtained by solving $R_{11}X = R_{12}$ (usual least squares); otherwise continue (solve a reduced TLS problem).

Step 2. Compute the SVD of R_{22}

$$R_{22} = U\Sigma V^T, \qquad \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_{n_2+d}),$$

where the singular values are ordered in decreasing order of magnitude.

Step 3a. Determine $k \leq n_2$ such that

$$\sigma_k > \sigma_{k+1} = \cdots = \sigma_{n_2+d} = 0,$$

and set $V_{22} = (v_{k+1}, \ldots, v_{n_2+d})$. If $n_1 > 0$ then compute V_2 by back-substitution from

$$R_{11}V_{12} = -R_{12}V_{22}, \qquad V_2 = \begin{pmatrix} V_{12} \\ V_{22} \end{pmatrix},$$

else set $V_2 = V_{22}$.

Step 3b. Perform Householder transformations such that

$$V_2Q = \begin{pmatrix} \Gamma & 0 \\ Z & Y \end{pmatrix},$$

where $\Gamma \in \mathbf{R}^{d\times d}$ is upper triangular. If Γ is nonsingular then the solution is

$$X = -Z\Gamma^{-1}.$$

Otherwise the TLS problem is nongeneric and has no solution.

Note that the QR factorization in the first step would be the first step in computing the SVD of A.

8.7.6 Iteratively Reweighted Least Squares.

In some applications it might be more adequate to solve the problem

$$\min \|Ax - b\|_p \tag{8.7.24}$$

for some l_p-norm with $p \neq 2$. For $p = 1$ the solution may not be unique, while for $1 < p < \infty$ the problem (8.7.24) is strictly convex and hence has exactly one solution. Minimization in the l_1-norm or l_∞-norm is more complicated since the function $f(x) = \|Ax - b\|_p$ is not differentiable for $p = 1, \infty$.

Example 8.7.2. *To illustrate the effect of using a different norm we consider the problem of estimating the scalar x from m observations $b \in \mathbf{R}^m$. This is equivalent to minimizing $\|Ax - b\|_p$, with $A = e = (1, 1, \ldots, 1)^T$. It is easily verified that if $b_1 \geq b_2 \geq \ldots \geq b_m$, then the solution x_p for some different values p are*

$$\begin{aligned} x_1 &= b_{\frac{m+1}{2}}, \quad (m \text{ odd}) \\ x_2 &= \frac{1}{m}(b_1 + b_2 + \ldots + b_m), \\ x_\infty &= \frac{1}{2}(b_1 + b_m). \end{aligned}$$

These estimates correspond to the median, mean, and midrange respectively. Note that the estimate x_1 is insensitive to the extreme values of b_i, while x_∞ only depends on the extreme values. The l_∞ solution has the property that the absolute error in at least n equations equals the maximum error.

The simple example above shows that the l_1 norm of the residual vector has the advantage of giving a solution that is **robust**, i.e., a small number of isolated large errors will usually not change the solution much. A similar effect is also achieved with p greater than but close to 1.

For solving the l_p norm problem when $1 < p < 3$, the **iteratively reweighted least squares** (IRLS) method (see Osborne [44, 1985]) can be used to reduce the problem to a sequence of weighted least squares problems.

We start by noting that, provided that $|r_i(x)| = |b - Ax|_i > 0$, $i = 1, \ldots, m$, the problem (8.7.24) can be restated in the form $\min_x \psi(x)$, where

$$\psi(x) = \sum_{i=1}^{m} |r_i(x)|^p = \sum_{i=1}^{m} |r_i(x)|^{p-2} r_i(x)^2. \tag{8.7.25}$$

This can be interpreted as a weighted least squares problem

$$\min_x \|D(r)^{(p-2)/2}(b - Ax)\|_2, \qquad D(r) = \operatorname{diag}(|r|), \tag{8.7.26}$$

where $\operatorname{diag}(|r|)$ denotes the diagonal matrix with ith component $|r_i|$.

The diagonal weight matrix $D(r)^{(p-2)/2}$ in (8.7.26) depends on the unknown solution x, but we can attempt to use the following iterative method.

Algorithm 8.7.2
IRLS for l_p Approximation $1 < p < 2$
Let $x^{(0)}$ be an initial approximation such that $r_i^{(0)} = (b - Ax^{(0)})_i \neq 0$, $i = 1, \ldots, n$.

$$
\begin{aligned}
&\text{for } k = 0, 1, 2, \ldots \\
&\quad r_i^{(k)} = (b - Ax^{(k)})_i; \\
&\quad D_k = \operatorname{diag}\left(\left(|r_i^{(k)}|\right)^{(p-2)/2}\right); \\
&\quad \text{solve } \delta x^{(k)} \text{ from} \\
&\qquad \min_{\delta x} \left\| D_k (r^{(k)} - A\delta x) \right\|_2; \\
&\quad x^{(k+1)} = x^{(k)} + \delta x^{(k)}; \\
&\text{end}
\end{aligned}
$$

Since $D_k b = D_k(r^{(k)} - Ax^{(k)})$, it follows that $x^{(k+1)}$ in IRLS solves $\min_x \|D_k(b - Ax)\|_2$, but the implementation above is to be preferred. It has been assumed that in the IRLS algorithm, at each iteration $r_i^{(k)} \neq 0$, $i = 1, \ldots, n$. In practice this cannot be guaranteed, and it is customary to modify the algorithm so that

$$D_k = \operatorname{diag}\left(\left(100ue + |r^{(k)}|\right)^{(p-2)/2}\right),$$

where u is the machine precision and $e^T = (1, \ldots, 1)$ is the vector of all ones. Because the weight matrix D_k is not constant, the simplest implementations of IRLS recompute, e.g., the QR factorization of $D_k A$ in each step. It should be pointed out that the iterations can be carried out entirely in the r space without the x variables. Upon convergence to a residual vector r_{opt} the corresponding solution can be found by solving the consistent linear system $Ax = b - r_{\text{opt}}$.

It can be shown that in the l_p case any fixed point of the IRLS iteration satisfies the necessary conditions for a minimum of $\psi(x)$. The IRLS method is convergent for $1 < p < 3$, and also for $p = 1$ provided that the l_1 approximation problem has a unique nondegenerate solution. However, the IRLS method can be extremely slow when p is close to unity.

Review Questions

1. Formulate the total least squares (TLS) problem. The solution of the TLS problem is related to a theorem on matrix approximation. Which?

Problems and Computer Exercises

1. Consider a TLS problem where $n = 1$ and

$$C = (A, b) = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}.$$

Show that the unique minimizing ΔC gives

$$C + \Delta C = (A + E, b + r) = \begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix}$$

so the perturbed system is not compatible, but that an arbitrary small perturbation ϵ in the (2,1) element will give a compatible system with solution $x = 2/\epsilon$.

2. Write a MATLAB program for fitting a straight line $c_1 x + c_2 y = h$ to given points $(x_i, y_i) \in \mathbf{R}^2$, $i = 1, 2, \ldots, m$. Follow the outline in Example 8.6.3. Use the Algorithm 10.4.2 to compute the SVD of R. The program should handle all exceptional cases, e.g., $c_1 = 0$ or and/or $c_2 = 0$.

3. (a) Let $A \in \mathbf{R}^{m\times n}$, $m \geq n$, $b \in \mathbf{R}^m$, and consider the **total least squares** (TLS) problem. $\min_{E,r} \|(E, r)\|_F$, where $(A+E)x = b+r$. If we have the QR factorization

$$Q^T(A,\ b) = \begin{pmatrix} S \\ 0 \end{pmatrix}, \qquad S = \begin{pmatrix} R & z \\ 0 & \rho \end{pmatrix}.$$

then the ordinary least squares solution is $x_{LS} = R^{-1}z$, $\|r\|_2 = \rho$.
Show that if a TLS solution x_{TLS} exists, then it holds

$$\begin{pmatrix} R^T & 0 \\ z^T & \rho \end{pmatrix} \begin{pmatrix} R & z \\ 0 & \rho \end{pmatrix} \begin{pmatrix} x_{TLS} \\ -1 \end{pmatrix} = \sigma_{n+1}^2 \begin{pmatrix} x_{TLS} \\ -1 \end{pmatrix},$$

where σ_{n+1} is the smallest singular value of $(A,\ b)$.

(b) Write a program using inverse iteration to compute x_{TLS}, i.e., for $k = 0, 1, 2, \ldots$, compute a sequence of vectors $x^{(k+1)}$ by

$$\begin{pmatrix} R^T & 0 \\ z^T & \rho \end{pmatrix} \begin{pmatrix} R & z \\ 0 & \rho \end{pmatrix} \begin{pmatrix} y^{(k+1)} \\ -\alpha \end{pmatrix} = \begin{pmatrix} x^{(k)} \\ -1 \end{pmatrix}, \qquad x^{(k+1)} = y^{(k+1)}/\alpha.$$

As starting vector use $x^{(0)} = x_{LS}$ on the assumption that x_{TLS} is a good approximation to x_{LS}. Will the above iteration always converge? Try to make it fail!

(c) Study the effect of scaling the right hand side in the TLS problem by making the substitution $z := \theta z$, $\rho := \theta\rho$. Plot $\|x_{TLS}(\theta) - x_{LS}\|_2$ as a function of θ and verify that when $\theta \to 0$, then $x_{TLS} \to x_{LS}$.

Hint For generating test problems it is suggested that you use the function qmult(A) from the MATLAB collection of test matrices by N. Higham to generate a matrix $C = (A,\ b) = Q_1 * D * Q_2^T$, where Q_1 and Q_2 are random real orthogonal matrices and D a given diagonal matrix. This allows you to generate problems where C has known singular values and vectors.

Notes

Several of the great mathematicians at the turn of the 19th century worked on methods for solving overdetermined linear systems. Laplace in 1799 used the principle of

minimizing the sum of absolute errors $|r_i|$. This leads to a solution x that satisfies at least n equations exactly. The method of least squares was first published as an algebraic procedure by Legendre 1805 in [39]. Gauss justified the least squares principle as a statistical procedure in [24], where he claimed to have used the method since 1795. This led to one of the most famous priority dispute in the history of mathematics. Gauss further developed the statistical aspects in 1821–1823. For an interesting accounts of the history of the invention of least squares, see Stiegler [60, 1981].

Because of its success in analyzing astronomical data the method of least squares rapidly became the method of choice when analyzing observation. Geodetic calculations was another early area of application of the least squares principle. In the last decade applications in control and signal processing has been a source of inspiration for developments in least squares calculations.

The singular value decomposition was independently developed by E. Beltrami 1873 and C. Jordan 1874; see G. W. Stewart [57, 1993] for an interesting account of the early history of the SVD. The first stable algorithm for computing the SVD the singular value was developed by Golub, Kahan and Wilkinson in the late 1960's. Several other applications of the SVD to matrix approximation can be found in Golub and Van Loan [29, Sec. 12.4].

A good introduction to generalized inverses Ben-Israel and Greville [5]. These should be used with caution since they tend to hide the computational difficulties involved with rank deficient matrices. A more complete and thorough treatment is given in the monograph by the same authors [6]. The use of generalized inverses in geodetic calculations is treated in Bjerhammar [8].

Peters and Wilkinson [49, 1970] developed methods based on Gaussian elimination from a uniform standpoint and the excellent survey by Noble [43, 1976]. Sautter [53, 1978] gives a detailed analysis of stability and rounding errors of the LU algorithm for computing pseudo-inverse solutions.

The different computational variants of Gram–Schmidt have an interesting history. The "modified" Gram–Schmidt (MGS) algorithm was in fact already derived by Laplace in 1816 as an elimination method using weighted row sums. Laplace did not interpret his algorithm in terms of orthogonalization, nor did he use it for computing least squares solutions! Bienaymé in 1853 gave a similar derivation of a slightly more general algorithm; see Björck [10, 1994]. What is now called the "classical" Gram–Schmidt (CGS) algorithm first appeared explicitly in papers by Gram 1883 and Schmidt 1908. Schmidt treats the solution of linear systems with infinitely many unknowns and uses the orthogonalization as a theoretical tool rather than a computational procedure.

In the 1950's algorithms based on Gram–Schmidt orthogonalization were frequently used, although their numerical properties were not well understood at the time. Björck [9] analyzed the modified Gram–Schmidt algorithm and showed its stability for solving linear least squares problems.

The systematic use of orthogonal transformations to reduce matrices to simpler form was initiated by Givens [25, 1958] and Householder [36, 1958]. The application of these transformations to linear least squares is due to Golub [26, 1965], where it was shown how to compute a QR factorization of A using Householder

transformations.

How to find the optimal backward error for the linear least squares problem was an open problem for many years, until it was elegantly answered by Karlsson et al. [64]; see also [37]. Gu [30] gives several approximations to that are optimal up to a factor less than 2. Optimal backward perturbation bounds for underdetermined systems are derived in [61]. The extension of backward error bounds to the case of constrained least squares problems is discussed by Cox and Higham [19].

The QR algorithm for banded rectangular matrices was first given by Reid [52]. Rank-revealing QR (RRQR) decompositions have been studied by a number of authors. A good survey can be found in Hansen [32]. The URV and ULV decompositions were introduced by G. W. Stewart [56, 58].

The systematic use of GQR as a basic conceptual and computational tool are explored by [45]. These generalized decompositions and their applications are discussed in [1]. Algorithms for computing the bidiagonal decomposition are due to Golub and Kahan [27, 1965]. The partial least squares (PLS) method, which has become a standard tool in chemometrics, goes back to Wold et al. [66].

The term "total least squares problem", which was coined by Golub and Van Loan [28], renewed the interest in the "errors in variable model". A thorough and rigorous treatment of the TLS problem is found in Van Huffel and Vandewalle [63]. The important role of the core problem for weighted TLS problems was discovered by Paige and Strakoš [47].

Modern numerical methods for solving least squares problems are surveyed in the two comprehensive monographs [38] and [11]. The latter contains a bibliography of 860 references, indicating the considerable research interest in these problems. Hansen [32] gives an excellent survey of numerical methods for the treatment of numerically rank deficient linear systems arising, for example, from discrete ill-posed problems.

Bibliography

[1] Edward Anderssen, Zhaojun Bai, and J. J. Dongarra. Generalized QR factorization and its applications. *Linear Algebra Appl.*, 162–164:243–271, 1992.

[2] Mario Arioli, James W. Demmel, and Iain S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl.*, 10:165–190, 1989.

[3] Jesse L. Barlow. More accurate bidiagonal reduction algorithm for computing the singular value decomposition. *SIAM J. Matrix Anal. Appl.*, 23:3:761–798, 2002.

[4] Jesse L. Barlow, Nela Bosner, and Zlatko Dramăk. A new stable bidiagonal reduction algorithm. *Linear Algebra Appl.*, 397:35–84, 2005.

[5] Adi Ben-Israel and T. N. E. Greville. Some topics in generalized inverses of matrices. In M. Z. Nashed, editor, *Generalized Inverses and Applications*, pages 125–147. Academic Press, Inc., New York, 1976.

[6] Adi Ben-Israel and T. N. E. Greville. *Generalized Inverses: Theory and Applications*. Springer, Berlin–Heidelberg–New York, 2003.

[7] Commandant Benoit. Sur la méthode de résolution des équations normales, etc. (procédés du commandant Cholesky). *Bull. Géodesique*, 2:67–77, 1924.

[8] Arne Bjerhammar. *Theory of Errors and Generalized Inverse Matrices*. Elsevier Scientific Publishing Co., Amsterdam, 1973.

[9] Å. Björck. Solving linear least squares problems by Gram–Schmidt orthogonalization. *BIT*, 7:1–21, 1967.

[10] Åke Björck. Numerics of Gram–Schmidt orthogonalization. *Linear Algebra Appl.*, 197–198:297–316, 1994.

[11] Åke Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, PA, 1996.

[12] Åke Björck and Christopher C. Paige. Loss and recapture of orthogonality in the modified Gram–Schmidt algorithm. *SIAM J. Matrix Anal. Appl.*, 13:176–190, 1992.

[13] Adam Bojanczyk, Nicholas J. Higham, and H. Patel. Solving the indefinite least squares problem by hyperbolic qr factorization. *SIAM J. Matrix Anal. Appl.*, 24:4:914–931, 2003.

[14] Tony F. Chan. Rank revealing QR factorizations. *Linear Algebra Appl.*, 88/89:67–82, 1987.

[15] Tony F. Chan and Per Christian Hansen. Low-rank revealing QR factorizations. *Numer. Linear Algebra Appl.*, 1:33–44, 1994.

[16] S. Chandrasekaran, Ming Gu, and A. H. Sayed. A stable and efficient algorithm for the indefinite linear least-squares problem. *SIAM J. Matrix Anal. Appl.*, 20:2:354–362, 1998.

[17] R. E. Cline and Robert J. Plemmons. ℓ_2-solutions to underdetermined linear systems. *SIAM Review*, 18:92–106, 1976.

[18] Anthony J. Cox and Nicholas J. Higham. Stability of Householder qr factorization for weighted least squares problems. In D. F. Griffiths, D. J. Higham, and G. A. Watson, editors, *Numerical Analysis 1997: Proceedings of the 17th Dundee Biennial Conference*, Pitman Research Notes in mathematics, vol. 380, pages 57–73. Longman Scientific and Technical, Harlow, Essex, UK, 1998.

[19] Anthony J. Cox and Nicholas J. Higham. Backward error bounds for constrained least squares problems. *BIT*, 39:2:210–227, 1999.

[20] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrica*, 1:211–218, 1936.

[21] Ky Fan and Alan J. Hoffman. Some metric inequalities in the space of matrices. *Proc. Amer. Math. Soc.*, 6:111–116, 1955.

[22] R. D. Fierro, P. C. Hansen, and P. S. K. Hansen. UTV tools: Matlab templates for rank-revealing UTV decompositions. *Numerical Algorithms*, to appear, 2003.

[23] C. F. Gauss. *The Theory of the Combination of Observations Least Subject to Errors. Pars Prior.* SIAM, Philadelphia, PA, G. W. Stewart, Translation 1995, 1821.

[24] Carl Friedrich Gauss. *Theory of the Motion of of the Heavenly Bodies Moving about the Sun in Conic Sections.* Dover, New York, (1963), C. H. Davis, Translation, 1809.

[25] Wallace G. Givens. Computation of plane unitary rotations transforming a general matrix to triangular form. *SIAM J. Appl. Math.*, 6:26–50, 1958.

[26] Gene H. Golub. Numerical methods for solving least squares problems. *Numer. Math.*, 7:206–216, 1965.

[27] Gene H. Golub and W. Kahan. Calculating the singular values and pseudoinverse of a matrix. *SIAM J. Numer. Anal. Ser. B*, 2:205–224, 1965.

[28] Gene H. Golub and Charles F. Van Loan. An analysis of the total least squares problem. *SIAM J. Numer. Anal.*, 17:883–893, 1980.

[29] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[30] Ming Gu. Backward perturbation bounds for linear least squares problems. *SIAM J. Matrix. Anal. Appl.*, 20:2:363–372, 1998.

[31] M. Gulliksson and P.-Å. Wedin. Perturbation theory for generalized and constrained linear least squares. *Numer. Linear Algebra Appl.*, 7:181–196, 2000.

[32] Per Christian Hansen. *Rank-Deficient and Discrete Ill-Posed Problems. Numerical Aspects of Linear Inversion.* SIAM, Philadelphia, 1998.

[33] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, second edition, 2002.

[34] Nicholas J. Higham. J-Orthogonal matrices: Properties and generation. *SIAM Review*, 45:3:504–519, 2003.

[35] Y. T. Hong and C. T. Pan. Rank-revealing QR decompositions and the singular value decomposition. *Math. Comp.*, 58:213–232, 1992.

[36] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. Assoc. Comput. Mach.*, 5:339–342, 1958.

[37] Rune Karlsson and Bertil Waldén. Estimation of optimal backward perturbation bounds for the linear least squares problem. *BIT*, 37:4:862–869, 1997.

[38] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, NJ, 1974. Reprinted by SIAM, Philadelphia, PA, 1995.

[39] Adrien-Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. Courcier, Paris, 1805.

[40] A. A. Markov. *Wahrscheinlichheitsrechnung*. Liebmann, Leipzig, second edition, 1912.

[41] Roy Mathias and G. W. Stewart. A block QR algorithm and the singular value decompositions. *Linear Algebra Appl.*, 182:91–100, 1993.

[42] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford*, 11:50–59, 1960.

[43] Ben Noble. Methods for computing the moore–penrose generalized inverse and related matters. In M. Z. Nashed, editor, *Generalized Inverses and Applications*, pages 245–302. Academic Press, Inc., New York, 1976.

[44] M. R. Osborne. *Finite Algorithms in Optimization and Data Analysis*. John Wiley, New York, 1985.

[45] Christopher C. Paige. Some aspects of generalized QR factorizations. In M. G. Cox and S. J. Hammarling, editors, *Reliable Numerical Computation*, pages 71–91. Clarendon Press, Oxford, UK, 1990.

[46] Christopher C. Paige and Z. Strakoš. Unifying least squares, total least squares and data least squares. In S. Van Huffel and P. Lemmerling, editors, *Total Least Squares and Errors-in-Variables Modeling*, pages 25–34. Kluwer Academic Publishers, Dordrecht, 2002.

[47] Christopher C. Paige and Zdeněk Strakoš. Scaled total least squares fundamentals. *Numer. Math.*, 91:1:117–146, 2002a.

[48] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics 20. SIAM, Philadelphia, PA, 1998.

[49] G. Peters and James H. Wilkinson. The least squares problem and pseudo-inverses. *Comput. J.*, 13:309–316, 1970.

[50] Robert J. Plemmons. Linear least squares by elimination and MGS. *J. Assoc. Comput. Mach.*, 21:581–585, 1974.

[51] M. J. D. Powell and J. K. Reid. On applying Householder's method to linear least squares problems. In A. J. M. Morell, editor, *Proceedings of the IFIP Congress 68*, pages 122–126. North-Holland, Amsterdam, 1969.

[52] J. K. Reid. A note on the least squares solution of a band system of linear equations by Householder reductions. *Comput J.*, 10:188–189, 1967.

[53] Werner Sautter. Fehleranalyse für die Gauss-Elimination zur Berechnung der Lösung minimaler Länge. *Numer. Math.*, 30:165–184, 1978.

[54] W. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrica.*, 31:1–10, 1966.

[55] H. R. Schwartz. Tridiagonalizing of a symmetric band matrix. *Numer. Math.*, 12:231–241, 1968. See also Wilkinson and Reinsch, 1971, 273–283.

[56] G. W. Stewart. An updating algorithm for subspace tracking. *IEEE Trans. Signal Process.*, 40:1535–1541, 1992.

[57] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35:4:551–556, 1993.

[58] G. W. Stewart. Updating a rank-revealing ULV decomposition. *SIAM J. Matrix Anal. Appl.*, 14:494–499, 1993.

[59] G. W. Stewart. The QLP approximation to the singular value decomposition. *SIAM J. Sci. Comput.*, 20:4:1336–1348, 1999.

[60] S. M. Stiegler. Gauss and the invention of least squares. *Ann. Statist.*, 9:465–474, 1981.

[61] Ji-guang Sun and Zheng Sun. Optimal backward perturbation bounds for underdetermined systems. *SIAM J. Matrix Anal. Appl.*, 18:393–402, 1997.

[62] Lloyd N. Trefethen and III David Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, PA, 1997.

[63] Sabine Van Huffel and Joos Vandewalle. *The Total Least Squares Problem; Computational Aspects and Analysis*. SIAM, Philadelphia, PA, 1991.

[64] Bertil Waldén, Rune Karlsson, and Ji guang. Sun. Optimal backward perturbation bounds for the linear least squares problem. *Numer. Linear Algebra Appl.*, 2:271–286, 1995.

[65] Per-Åke Wedin. Perturbation theory for pseudo-inverses. *BIT*, 9:217–232, 1973.

[66] Svante Wold, Axel Ruhe, Herman Wold, and W. J. Dunn. The collinearity problem in linear regression, the partial least squares (pls) approach to generalized inverses. *SIAM J. Sci. Statist. Comput.*, 5:735–743, 1984.

[67] M. Zelen. Linear estimation and related topics. In J. Todd, editor, *Survey of Numerical Analysis*, pages 558–584. McGraw-Hill, New York, 1962.

Index

algorithm
  classical Gram–Schmidt, 34
  Givens rotations, 45
  Householder QR, 49
  Householder reflection, 44
  IRLS, 112
  MGS
    least squares by, 41
    minimum norm solution by, 42
    Orthogonal projection by, 41
  modified Gram–Schmidt, 36
augmented linear system, 90
augmented system, 90–92, 96

B-splines
  cubic, 81
banded matrix
  bidiagonal reduction, 72
  of standard form, 80
bandwidth
  row, 20
bidiagonal reduction, 69–72
  banded matrix, 72
bidiagonalization
  for TLS, 107–109
block angular form, 85–87
  doubly bordered, 85
block angular problem
  QR algorithm, 87
block triangular form, 83
  coarse decomposition, 83
  fine decomposition, 83

CGS, *see* classical Gram–Schmidt
Cholesky factorization, 19
column pivoting, 50
column scaling, 13
  optimal, 23
condition estimation, 56–57
condition number
  general matrix, 11
covariance matrix, 3, 4, 21
  block angular problem, 87
  estimate, 21

data least squares problem, 104
direct elimination
  method of, 101
distribution function, 2

elementary reflector, 43
error
  componentwise estimate, 57
errors-in-variable model, 103
expected value, 2

filter factor, 62
Fischer's theorem, 8
flop count
  Householder QR, 58
  normal equations, 19
  QR factorization, 49, 77
    banded, 80
  reduction to bidiagonal form, 70
fundamental subspaces, 6

Gauss–Markoff's theorem, 3
generalized inverse, 6–7
Givens rotation, 45
Gram–Schmidt
  classical, 34
  modified, 36
  orthogonalization, 31–42

Householder reflector, 43

Householder vector, 43
hyperbolic rotations, 97

indefinite least squares problem, 96
inverse
  left, 16
IRLS, *see* iteratively reweighted least squares
iteratively reweighted least squares, 112–113

KKT-system, 90
Kronecker
  least squares problem, 87–89

Lagrange multipliers, 90
latent root regression, 103
least squares
  banded problems, 20–21, 79–81
  characterization of solution, 16–18
  general problem, 4
  principle of, 2
  problem, 1
  solution, 1
  total, 103–111
  with linear equality constraints, 100
least squares problem
  damped, 61
  indefinite, 96
  Kronecker, 87–89
  slightly overdetermined, 27
  stiff, 93
  weighted, 93
left-inverse, 15
linear model
  general univariate, 4
  standard, 3
linear regression, 19
linear system
  augmented, 90
  underdetermined, 91
LU factorization, 25
  of rectangular matrix, 7

matrix
  idempotent, 32
  orthogonal, 32, 43
  unitary, 32
matrix approximation, 9–10
mean, 112
median, 112
MGS, *see* modified Gram–Schmidt
midrange, 112
minimax characterization
  of singular values, 8
minimum distance
  between matrices, 9
minimum norm solution, 2
Moore–Penrose inverse, 5

normal equations, 17
  accuracy of, 21–25
  generalized, 91
  iterative refinement, 24
  method of, 18–21
  scaling of, 23
normalized residual, 21
null space method, 91, 101
nullspace
  numerical, 9
    from SVD, 9
numerical rank, 9
  by SVD, 59–61

oblique projector, 32
orthogonal, 31
  complement, 31
  matrix, 32, 43
  projector, 32
orthogonal projections, 6
orthogonal projector, 31
orthogonal regression, 98–99
orthogonality
  loss of, 36–40
orthonormal, 31

Partial least squares method, 76
Penrose conditions, 5
perturbation
  component-wise, 13
  of least squares solution, 11–13

Peters–Wilkinson method, 25–27
plane rotations, 45
polar decomposition, 10
projection
    oblique, 33
projector
    orthogonal, 32
pseudo-inverse, 5
    Bjerhammar, 16
    Kronecker product, 88
    solution, 5, 6
pseudo-inverse solution
    by LU factorization, 7

QR decomposition
    Kronecker product, 88
    row pivoting, 94
    row sorting, 94
QR factorization, 35, 47
    backward stability, 49, 55
    column pivoting, 50
    complete, 64
    for rank deficient problems, 57, 59
    of banded matrix, 82
    rank revealing, 66

range space method, 91
regularization, 61–62
    filter factor, 62
residual
    normalized, 21
right-inverse, 15

saddle-point system, 90
signature matrix, 96
singular value decomposition, 7–61
sparse matrix
    block angular form, 85–87
    block triangular form, 83
SVD, *see* singular value decomposition
    and pseudo-inverse, 4–7
    Kronecker product, 89
SVD solution
    truncated, 60

TLS, *see* total least squares
TLS problem
    scaled, 104
total least squares, 103–111
    by SVD, 104–105
    conditioning, 105
    generalized, 109
    mixed, 110
    multidimensional, 110
truncated SVD, 59–61
TSVD, *see* truncated SVD

ULV decomposition, 68
underdetermined problem, 2
underdetermined system
    general solution, 54
    minimum norm solution, 54
URV decomposition, 67

variance, 2
vector
    orthogonal, 31
    orthonormal, 31

weighted problem
    condition number, 93

Contents

9 Matrix Eigenvalue Problems **1**

9.1 Basic Properties . . . 1
9.1.1 Introduction . . . 1
9.1.2 Complex Matrices . . . 2
9.1.3 Theoretical Background . . . 3
9.1.4 Invariant Subspaces . . . 5
Review Questions . . . 9
Problems . . . 10
9.2 Canonical Forms and Matrix Functions . . . 11
9.2.1 The Schur Normal Form . . . 11
9.2.2 Sylvester's Equation and Jordan's Canonical Form . . . 14
9.2.3 Convergence of Matrix Power Series . . . 18
9.2.4 Matrix Functions . . . 21
9.2.5 Non-Negative Matrices . . . 28
9.2.6 Finite Markov Chains . . . 29
Review Questions . . . 32
Problems and Computer Exercises . . . 33
9.3 Perturbation Theory and Eigenvalue Bounds . . . 35
9.3.1 Gerschgorin's Theorems . . . 35
9.3.2 Perturbation Theorems . . . 38
9.3.3 Hermitian Matrices . . . 41
9.3.4 Rayleigh quotient and residual bounds . . . 43
9.3.5 Residual bounds for SVD . . . 46
Review Questions . . . 47
Problems . . . 47
9.4 The Power Method . . . 49
9.4.1 The Simple Power Method . . . 49
9.4.2 Deflation . . . 51
9.4.3 Spectral Transformation and Inverse Iteration . . . 52
9.4.4 Eigenvectors by Inverse Iteration . . . 53
9.4.5 Rayleigh Quotient Iteration . . . 55
9.4.6 Subspace Iteration . . . 56
Review Questions . . . 58
Problems . . . 58

9.5 Jacobi Methods . . . 59
9.5.1 Jacobi Methods for Real Symmetric Matrices . . . 59
9.5.2 Jacobi Methods for Computing the SVD. . . . 62
Review Questions . . . 65
Problems . . . 65
9.6 Transformation to Condensed Form . . . 67
9.6.1 Introduction . . . 67
9.6.2 Unitary Elementary Transformations . . . 67
9.6.3 Reduction to Hessenberg Form . . . 69
9.6.4 Reduction to Symmetric Tridiagonal Form . . . 72
9.6.5 A Divide and Conquer Algorithm . . . 74
9.6.6 Spectrum Slicing . . . 75
Review Questions . . . 78
Problems . . . 78
9.7 The LR and QR Algorithms . . . 79
9.7.1 The Basic LR and QR Algorithms . . . 79
9.7.2 Convergence of the Basic QR Algorithm . . . 82
9.7.3 QR Algorithm for Hessenberg Matrices . . . 84
9.7.4 QR Algorithm for Symmetric Tridiagonal Matrices . 89
9.7.5 QR-SVD algorithms for Bidiagonal Matrices . . . 92
9.7.6 Singular Values by Spectrum Slicing . . . 99
Review Questions . . . 99
Problems . . . 100
9.8 Subspace Methods for Large Eigenvalue Problems . . . 101
9.8.1 The Rayleigh–Ritz Procedure . . . 101
9.8.2 Subspace Iteration for Hermitian Matrices . . . 103
9.8.3 Krylov Subspaces . . . 105
9.8.4 The Lanczos Process . . . 108
9.8.5 Golub–Kahan Bidiagonalization. . . . 111
9.8.6 Arnoldi's Method. . . . 112
Review Questions . . . 113
Problems . . . 113
9.9 Generalized Eigenvalue Problems . . . 113
9.9.1 Introduction . . . 113
9.9.2 Canonical Forms . . . 114
9.9.3 Reduction to Standard Form . . . 115
9.9.4 Methods for Generalized Eigenvalue Problems . . . 117
9.9.5 The Generalized SVD. . . . 118
9.9.6 The CS Decomposition. . . . 120
Review Questions . . . 121
Problems . . . 122

Bibliography **125**

Index **129**

Chapter 9
Matrix Eigenvalue Problems

9.1 Basic Properties

9.1.1 Introduction

Eigenvalues and eigenvectors are a standard tool in the mathematical sciences and in scientific computing. Eigenvalues give information about the behavior of evolving systems governed by a matrix or operator. The problem of computing eigenvalues and eigenvectors of a matrix occurs in many settings in physics and engineering. Eigenvalues are useful in analyzing resonance, instability, and rates of growth or decay with applications to, e.g., vibrating systems, airplane wings, ships, buildings, bridges and molecules. Eigenvalue decompositions also play an important part in the analysis of many numerical methods. Further, singular values are closely related to an eigenvalues a symmetric matrix.

In this chapter we treat numerical methods for computing eigenvalues and eigenvectors of matrices. In the first three sections we briefly review the classical theory needed for the proper understanding of the numerical methods treated in the later sections. In particular Section 9.1 gives a brief account of basic facts of the matrix eigenvalue problem, Section 9.2 treats the classical theory of canonical forms and matrix functions. Section 9.3 is devoted to the localization of eigenvalues and perturbation results for eigenvalues and eigenvectors.

Section 9.5 treats the Jacobi methods for the real symmetric eigenvalue problem and the SVD. These methods have advantages for parallel implementation and are potentially very accurate. The power method and its modifications are treated in Section 9.4. Transformation to condensed form described in Section 9.4 often is a preliminary step in solving the eigenvalue problem. Followed by the QR algorithm this constitutes the current method of choice for computing eigenvalues and eigenvectors of small to medium size matrices, see Section 9.7. This method can also be adopted to compute singular values and singular vectors although the numerical implementation is often far from trivial, see Section 9.7.

In Section 9.8 we briefly discuss some methods for solving the eigenvalue prob-

lem for large sparse matrices. Finally, in Section 9.9 we consider the generalized eigenvalue problem $Ax = \lambda Bx$, and the generalized SVD.

9.1.2 Complex Matrices

In developing the theory for the matrix eigenvalue problem it often is more relevant to work with complex vectors and matrices. This is so because a real unsymmetric matrix can have complex eigenvalues and eigenvectors. We therefore introduce the vector space $\mathbf{C}^{n\times m}$ of all complex $n \times m$ matrices whose components are complex numbers.

Most concepts and operations in Section 7.2 carry over from the real to the complex case in a natural way. Addition and multiplication of vectors and matrices follow the same rules as before. The Hermitian inner product of two vectors x and y in $\mathbf{C}^n$ is defined as

$$(x, y) = x^H y = \sum_{k=1}^{n} \bar{x}_k y_k, \tag{9.1.1}$$

where $x^H = (\bar{x}_1, \ldots, \bar{x}_n)$ and $\bar{x}_i$ denotes the complex conjugate of x_i. Hence $(x, y) = \overline{(y, x)}$, and $x \perp y$ if $x^H y = 0$. The Euclidean length of a vector x thus becomes

$$\|x\|_2 = (x, x)^{1/2} = \sum_{k=1}^{n} |x_k|^2.$$

The set of complex $m \times n$ matrices is denoted by $\mathbf{C}^{m\times n}$. If $A = (a_{ij}) \in \mathbf{C}^{m\times n}$ then by definition its **adjoint** matrix $A^H \in \mathbf{C}^{n\times m}$ satisfies

$$(x, A^H y) = (Ax, y).$$

By using coordinate vectors for x and y it follows that $A^H = \bar{A}^T$, that is, A^H is the conjugate transpose of A. It is easily verified that $(AB)^H = B^H A^H$. In particular, if α is a scalar $\alpha^H = \bar{\alpha}$.

A matrix $A \in \mathbf{C}^{n\times n}$ is called self-adjoint or **Hermitian** if $A^H = A$. A Hermitian matrix has analogous properties to a real symmetric matrix. If A is Hermitian, then $(x^H Ax)^H = x^H Ax$ is real, and A is called positive definite if

$$x^H Ax > 0, \quad \forall x \in \mathbf{C}^n, \quad x \neq 0.$$

A square matrix U is **unitary** if $U^H U = I$. From (9.1.1) we see that a unitary matrix preserves the Hermitian inner product

$$(Ux, Uy) = (x, U^H Uy) = (x, y).$$

In particular the 2-norm is invariant under unitary transformations, $\|Ux\|_2^2 = \|x\|_2^2$. Hence, unitary matrices corresponds to real orthogonal matrices. Note that in every case, the new definition coincides with the old when the vectors and matrices are real.

9.1.3 Theoretical Background

Of central importance in the study of matrices $A \in \mathbf{C}^{n\times n}$ are the special vectors whose directions are not changed when multiplied by A. A complex scalar λ such that

$$Ax = \lambda x, \qquad x \neq 0, \tag{9.1.2}$$

is called an **eigenvalue** of A and x is an **eigenvector** of A. When an eiegnvalue is known, the determination of the corresponding eigenvector(s) requires the solution of a linear homogenous system $(A - \lambda I)x = 0$. Clearly, if x is an eigenvector so is αx for any scalar $\alpha \neq 0$.

It follows that λ is an eigenvalue of A if and only if the system $(A - \lambda I)x = 0$ has a nontrivial solution $x \neq 0$, or equivalently if and only if the matrix $A - \lambda I$ is singular. Hence the eigenvalues satisfy the **characteristic equation**

$$p_n(\lambda) = \det(A - \lambda I) = \begin{vmatrix} a_{11}-\lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22}-\lambda & \cdots & a_{2n} \\ \vdots & \cdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn}-\lambda \end{vmatrix} = 0 \tag{9.1.3}$$

The set $\lambda(A) = \{\lambda_i\}_{i=1}^n$ of all eigenvalues of A is called the **spectrum**[1] of A. The polynomial $p_n(\lambda) = \det(A - \lambda I)$ is called the **characteristic polynomial** of the matrix A. Expanding the determinant in (9.1.3) it follows that $p(\lambda)$ has the form

$$p_n(\lambda) = (a_{11} - \lambda)(a_{22} - \lambda)\cdots(a_{nn} - \lambda) + q(\lambda), \tag{9.1.4}$$
$$= (-1)^n(\lambda^n - \xi_{n-1}\lambda^{n-1} - \cdots \xi_0). \tag{9.1.5}$$

where $q(\lambda)$ has degree at most $n-2$. Thus, by the fundamental theorem of algebra the matrix A has exactly n eigenvalues λ_i, $i = 1, 2, \ldots n$, counting multiple roots according to their multiplicities, and we can write

$$p(\lambda) = (\lambda_1 - \lambda)(\lambda_2 - \lambda)\cdots(\lambda_n - \lambda).$$

Using the relation between roots and coefficients of an algebraic equation we obtain

$$p(0) = \lambda_1\lambda_2\cdots\lambda_n = \det(A), \tag{9.1.6}$$

Further, using the relation between roots and coefficients of an algebraic equation we obtain

$$\lambda_1 + \lambda_2 + \cdots + \lambda_n = \operatorname{trace}(A). \tag{9.1.7}$$

where $\operatorname{trace}(A) = a_{11} + a_{22} + \cdots + a_{nn}$ is the **trace** of the matrix A. This relation is useful for checking the accuracy of a computed spectrum.

[1] From Latin verb *specere* meaning "to look".

Theorem 9.1.1.
Let $A \in \mathbf{C}^{n\times n}$. Then

$$\lambda(A^T) = \lambda(A), \quad \lambda(A^H) = \bar{\lambda}(A).$$

Proof. Since $\det(A^T - \lambda I)^T = \det(A - \lambda I)^T = \det(A - \lambda I)$ it follows that A^T and A have the same characteristic polynomial and thus same set of eigenvalues. For the second part note that $\det(A^H - \bar{\lambda} I) = \det(A - \lambda I)^H$ is zero if and only if $\det(A - \lambda I)$ is zero. □

By the above theorem, if λ is an eigenvalue of A then $\bar{\lambda}$ is an eigenvalue of A^H, i.e., $A^H y = \overline{\lambda} y$ for some vector $y \neq 0$, or equivalently

$$y^H A = \lambda y^H, \qquad y \neq 0. \tag{9.1.8}$$

Here y is called a **left** eigenvector of A, and consequently if $Ax = \lambda x$, x is also called a **right** eigenvector of A. For a Hermitian matrix $A^H = A$ and thus $\overline{\lambda} = \lambda$, i.e., λ is real. In this case the left and right eigenvectors can be chosen to coincide.

Theorem 9.1.2.
Let λ_i and λ_j be two distinct eigenvalues of $A \in \mathbf{C}^{n\times n}$, and let y_i and x_j be left and right eigenvectors corresponding to λ_i and λ_j respectively. Then $y_i^H x_j = 0$, i.e., y_i and x_j are orthogonal.

Proof. By definition we have

$$y_i^H A = \lambda_i y_i^H, \qquad Ax_j = \lambda_j x_j.$$

Multiplying the first equation with x_j from the right and the second with y_i^H from the left and subtracting we obtain $(\lambda_i - \lambda_j) y_i^H x_j = 0$. Since $\lambda_i \neq \lambda_j$ the theorem follows. □

Definition 9.1.3.
Denote the eigenvalues of the matrix $A \in \mathbf{C}^{n\times n}$ by $\lambda_i|$, $i = 1 : n$. The **spectral radius** *of A is is the maximal absolute value of the eigenvalues of A*

$$\rho(A) = \max_i |\lambda_i|. \tag{9.1.9}$$

The **spectral abscissa** *is the maximal real part of the eigenvalues of A*

$$\alpha(A) = \max_i \Re\lambda_i. \tag{9.1.10}$$

If X is any square nonsingular matrix and

$$\tilde{A} = X^{-1} A X, \tag{9.1.11}$$

then $\tilde{A}$ is said to be similar to A and (9.1.11) is called a **similarity transformation** of A. Similarity of matrices is an equivalence transformation, i.e., if A is similar to B and B is similar to C then A is similar to C.

Theorem 9.1.4.
If A and B are similar, then A and B have the same characteristic polynomial, and hence the same eigenvalues. Further, if $B = X^{-1}AX$ and y is an eigenvector of B corresponding to λ then Xy is an eigenvector of A corresponding to λ.

Proof. We have

$$\begin{aligned}\det(B-\lambda I) &= \det(X^{-1}AX-\lambda I) = \det(X^{-1}(A-\lambda I)X)\\ &= \det\left(X^{-1}\right)\det(A-\lambda I)\det(X) = \det(A-\lambda I).\end{aligned}$$

Further, from $AX = XB$ it follows that $AXy = XBy = \lambda Xy$. □

Let $Ax_i = \lambda_i x_i$, $i = 1, \ldots, n$. It is easily verified that these n equations are equivalent to the single matrix equation

$$AX = X\Lambda, \qquad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n),$$

where $X = (x_1, \ldots, x_n)$ is a matrix of right eigenvectors of A If the eigenvectors are linearly independent then X is nonsingular and we have

$$X^{-1}AX = \Lambda. \tag{9.1.12}$$

This similarity transformation by X transforms A to diagonal form and A is said to be **diagonalizable**.

From (9.1.12) it follows that $X^{-1}A = \Lambda X^{-1}$, which shows that *the rows of X^{-1} are left eigenvectors y_i^H*. We can also write $A = X\Lambda X^{-1} = X\Lambda Y^H$, or

$$A = \sum_{i=1}^{n} \lambda_i P_i, \quad P_i = x_i y_i^H. \tag{9.1.13}$$

Since $Y^H X = I$ it follows that the left and right eigenvectors are biorthogonal, $y_i^H x_j = 0$, $i \neq j$, and $y_i^H x_i = 1$. Hence P_i is a projection ($P_i^2 = P_i$) and (9.1.13) is called the **spectral decomposition** of A. The decomposition (9.1.13) is essentially unique. If λ_{i_1} is an eigenvalue of multiplicity m and $\lambda_{i_1} = \lambda_{i_2} = \cdots = \lambda_{i_m}$, then the vectors $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$ can be chosen as any basis for the null space of $A - \lambda_{i_1} I$.

9.1.4 Invariant Subspaces

Suppose that for a matrix $X \in \mathbf{C}^{n\times k}$, $\operatorname{rank}(X) = k \leq n$, it holds that

$$AX = XB, \qquad B \in \mathbf{C}^{k\times k}.$$

Any vector $x \in \mathcal{R}(X)$ can be written $x = Xz$ for some vector $z \in \mathbf{C}^k$. Thus $Ax = AXz = XBz \in \mathcal{R}(X)$ and $\mathcal{R}(X)$ is called a **right invariant subspace**. If $By = \lambda y$, it follows that

$$AXy = XBy = \lambda Xy,$$

and so *any eigenvalue λ of B is also an eigenvalue of A and Xy a corresponding eigenvector*. Note that any set of right eigenvectors spans a right invariant subspace.

Similarly, if $Y^HA = BY^H$, where $Y \in \mathbf{C}^{n\times k}$, rank $(Y) = k \le n$, then $\mathcal{R}(Y)$ is a **left invariant** subspace. If $v^HB = \lambda v^H$ it follows that

$$v^HY^HA = v^HBY^H = \lambda v^HY^H,$$

and so λ is an eigenvalue of A and Yv is a left eigenvector.

Definition 9.1.5.

A matrix $A \in \mathbf{R}^{n\times n}$, is said to be **reducible** *if for some permutation matrix P, P^TAP has the form*

$$P^TAP = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix}, \tag{9.1.14}$$

where B and C, are square submatrices, or if $n = 1$ and $A = 0$. Otherwise A is called **irreducible**.

The concept of a reducible matrix can be illustrated using some elementary notions from the theory of graphs. The **directed graph** of a matrix A is constructed as follows: Let $P_1, \ldots, P_n$ be n distinct points in the plane called **nodes**. For each $a_{ij} \neq 0$ in A we connect node P_i to node P_j by means of directed **edge** from node i to node j. (Compare the definition of an undirected graph of a matrix in Def. 6.5.2.) It can be shown that a matrix A is irreducible if and only if its graph is **connected** in the following sense. Given any two distinct nodes P_i and P_j there exists a path $P_i = P_{i_1}, P_{i_2}, \ldots, P_{i_p} = P_j$ along directed edges from P_i to P_j. Note that the graph of a matrix A is the same as the graph of P^TAP, where P is a permutation matrix; only the labeling of the node changes.

Assume that a matrix A is reducible to the form (9.1.14), where $B \in \mathbf{R}^{r\times r}$, $B \in \mathbf{R}^{s\times s}$ $(r + s = n)$. Then we have

$$\tilde{A}\begin{pmatrix} I_r \\ 0 \end{pmatrix} = \begin{pmatrix} I_r \\ 0 \end{pmatrix} B, \qquad (\,0 \quad I_s\,)\,\tilde{A} = D\,(\,0 \quad I_s\,),$$

that is, the first r unit vectors span a right invariant subspace, and the s last unit vectors span a left invariant subspace of $\tilde{A}$. It follows that the spectrum of A equals the union of the spectra of B and D.

If B and D are reducible they can be reduced in the same way. Continuing in this way until the diagonal blocks are irreducible we obtain a block upper triangular matrix

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ 0 & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & A_{NN} \end{pmatrix}, \tag{9.1.15}$$

where each diagonal block A_{ii} is square.

Theorem 9.1.6.

Assume that the matrix A can be reduced by a permutation to the block upper triangular form (9.1.15). Then $\lambda(A) = \bigcup_{i=1}^{N} \lambda(A_{ii})$, where $\lambda(A)$ denotes the

spectrum of A. In particular the eigenvalues of a triangular matrix are its diagonal elements.

Many important numerical methods for computing eigenvalues and eigenvectors of a matrix A perform a *sequence of similarity transformations* to transform A into a matrix of simpler form. With $A_0 = A$ one computes

$$A_k = P_k^{-1} A_{k-1} P_k, \quad k = 1, 2, ...$$

The matrix A_k is similar to A and the eigenvectors x of A and y of A_k are related by $x = P_1 P_2 \cdots P_k y$. The eigenvalues of a triangular matrix equal its diagonal elements. Hence if the matrix A can be transformed by successive similarities to triangular form, then its eigenvalues are trivial to determine.

Let $AX_1 = X_1 B$, for some $X_1 \in \mathbf{R}^{n\times p}$ of rank p, and $B \in \mathbf{R}^{p\times p}$. Then $\mathcal{R}(X_1)$ is a right invariant subspace of A. Let $X_2 \in \mathbf{R}^{n\times(n-p)}$ be such that $X = (X_1, X_2)$ is invertible. Then we have

$$X^{-1}AX = X^{-1}(AX_1, AX_2) = (X^{-1}X_1 B, X^{-1}AX_2) = \begin{pmatrix} B & T_{12} \\ 0 & T_{22} \end{pmatrix} \tag{9.1.16}$$

that is, $X^{-1}AX$ is reducible. Hence, if a set of eigenvalues of A and a basis X_1 for a corresponding right invariant are known, then we can find the remaining eigenvalues of A from T_{22}. This process is called **deflation** and is a powerful tool for computation of eigenvalues and eigenvectors. Note that if $X_1 = Q_1$ has orthonormal columns, then $X = (Q_1, Q_2)$ in (9.1.16) can be chosen as an orthogonal matrix.

A matrix A may not have a full set of n linearly independent eigenvectors. However, it holds:

Theorem 9.1.7.

Let $x_1, \ldots, x_k$ be eigenvectors of $A \in \mathbf{C}^{n\times n}$ corresponding to distinct eigenvalues $\lambda_1, \ldots, \lambda_k$. Then the vectors $x_1, \ldots, x_k$ are linearly independent. In particular if all the eigenvalues of a matrix A are distinct then A has a complete set of linearly independent eigenvectors and hence A is diagonalizable.

Proof. Assume that only the vectors $x_1 \ldots, x_p$, $p < k$, are linearly independent and that $x_{p+1} = \gamma_1 x_1 + \cdots + \gamma_p x_p$. Then $Ax_{p+1} = \gamma_1 Ax_1 + \cdots + \gamma_p Ax_p$, or

$$\lambda_{p+1} x_{p+1} = \gamma_1 \lambda_1 x_1 + \cdots + \gamma_p \lambda_p x_p.$$

It follows that $\sum_{i=1}^{p} \gamma_i(\lambda_i - \lambda_{p+1})x_i = 0$. Since $\gamma_i \neq 0$ for some i and $\lambda_i - \lambda_{p+1} \neq 0$ for all i, this contradicts the assumption of linear independence. Hence we must have $p = k$ linearly independent vectors. □

Let $\lambda_1, \cdots, \lambda_k$ be the distinct zeros of $p(\lambda)$ and let σ_i be the multiplicity of λ_i, $i = 1, ..., k$. The integer σ_i is called the **algebraic multiplicity** of the eigenvalue λ_i and

$$\sigma_1 + \sigma_2 + \cdots + \sigma_k = n.$$

To every distinct eigenvalue corresponds at least one eigenvector. All the eigenvectors corresponding to the eigenvalue λ_i form a linear subspace $L(\lambda_i)$ of $\mathbf{C}^n$ of dimension

$$\rho_i = n - \text{rank}(A - \lambda_i I). \tag{9.1.17}$$

The integer ρ_i is called the **geometric multiplicity** of λ_i, and specifies the maximum number of linearly independent eigenvectors associated with λ_i. The eigenvectors are not in general uniquely determined.

Theorem 9.1.8.

For the geometric and algebraic multiplicity the inequality $\rho(\lambda) \leq \sigma(\lambda)$ holds.

Proof. Let $\bar{\lambda}$ be an eigenvalue with geometric multiplicity $\rho = \rho(\bar{\lambda})$ and let $x_1, \ldots, x_\rho$ be linearly independent eigenvectors associated with $\bar{\lambda}$. If we put $X_1 = (x_1, \ldots, x_\rho)$ then we have $AX_1 = \bar{\lambda}X_1$. We now let $X_2 = (x_{\rho+1}, \cdots, x_n)$ consist of $n - \rho$ more vectors such that the matrix $X = (X_1, X_2)$ is nonsingular. Then it follows that the matrix $X^{-1}AX$ must have the form

$$X^{-1}AX = \begin{pmatrix} \bar{\lambda}I & B \\ 0 & C \end{pmatrix}$$

and hence the characteristic polynomial of A, or $X^{-1}AX$ is

$$p(\lambda) = (\bar{\lambda} - \lambda)^\rho \det(C - \lambda I).$$

Thus the algebraic multiplicity of $\bar{\lambda}$ is at least equal to ρ. □

If $\rho(\lambda) < \sigma(\lambda)$ then λ is said to be a **defective eigenvalue**. A matrix with at least one defective eigenvalue is **defective**, otherwise it is **nondefective**. The eigenvectors of a nondefective matrix A span the space $\mathbf{C}^n$ and A is said to have a complete set of eigenvectors. A matrix is nondefective if and only if it is diagonalizable.

Example 9.1.1.

The matrix $\bar{\lambda}I$, where I is a unit matrix of dimension n has the characteristic polynomial $p(\lambda) = (\bar{\lambda} - \lambda)^n$ and hence $\lambda = \bar{\lambda}$ is an eigenvalue of algebraic multiplicity equal to n. Since rank $(\bar{\lambda}I - \bar{\lambda} \cdot I) = 0$, there are n linearly independent eigenvectors associated with this eigenvalue. Clearly any vector $x \in \mathbf{C}^n$ is an eigenvector.

Now consider the nth order matrix

$$J_n(\bar{\lambda}) = \begin{pmatrix} \bar{\lambda} & 1 & & \\ & \bar{\lambda} & \ddots & \\ & & \ddots & 1 \\ & & & \bar{\lambda} \end{pmatrix}. \tag{9.1.18}$$

Also this matrix has the characteristic polynomial $p(\lambda) = (\bar{\lambda} - \lambda)^n$. However, since rank $(J_n(\bar{\lambda}) - \bar{\lambda} \cdot I) = n - 1$, $J_n(\bar{\lambda})$ has only one right eigenvector $x = (1, 0, ..., 0)^T$.

Similarly it has only one left eigenvector $y = (0, \ldots, 0, 1)^T$, and the eigenvalue $\lambda = \bar{\lambda}$ is defective. A matrix of this form is called a **Jordan block**, see Theorem 9.2.8.

For any nonzero vector $v_1 = v$, define a sequence of vectors by

$$v_{k+1} = Av_k = A^k v_1. \tag{9.1.19}$$

Let v_{m+1} be the first of these vectors that can be expressed as a linear combination of the preceding ones. (Note that we must have $m \leq n$.) Then for some polynomial p of degree m

$$p(\lambda) = c_0 + c_1\lambda + \cdots + \lambda^m$$

we have $p(A)v = 0$, i.e., p annihilates v. Since p is the polynomial of minimal degree that annihilates v it is called the **minimal polynomial** and m the **grade** of v with respect to A.

Of all vectors v there is at least one for which the degree is maximal, since for any vector $m \leq n$. If v is such a vector and q its minimal polynomial, then it can be shown that $q(A)x = 0$ for any vector x, and hence

$$q(A) = \gamma_0 I + \gamma_1 A + \cdots + \gamma_{s-1}A^{s-1} + A^s = 0.$$

This polynomial p is the **minimal polynomial** for the matrix A, see Section 9.2.2.

Consider the Kronecker product $C = A \otimes B$ of $A \in \mathbf{R}^{n\times n}$ and $B \in \mathbf{R}^{m\times m}$ as defined in Sec. 7.5.5 The eigenvalues and eigenvectors of C can be expressed in terms of the eigenvalues and eigenvectors of A and B. Assume that $Ax_i = \lambda_i x_i$, $i = 1, \ldots, n$, and $By_j = \mu_j y_j$, $j = 1, \ldots, m$. Then, using equation (7.5.26), we obtain

$$(A \otimes B)(x_i \otimes y_j) = (Ax_i) \otimes (By_j) = \lambda_i\mu_j(x_i \otimes y_j). \tag{9.1.20}$$

This shows that the nm eigenvalues of $A \otimes B$ are $\lambda_i\mu_j$, $i = 1, \ldots, n$, $j = 1, \ldots, m$, and $x_i \otimes y_j$ are the corresponding eigenvectors. If A and B are diagonalizable, $A = X^{-1}\Lambda_1 X$, $B = Y^{-1}\Lambda_2 Y$, then

$$(A \otimes B) = (X^{-1} \otimes Y^{-1})(\Lambda_1 \otimes \Lambda_2)(X \otimes Y),$$

and thus $A \otimes B$ is also diagonalizable.

The matrix

$$(I_m \otimes A) + (B \otimes I_n) \in \mathbf{R}^{nm\times nm} \tag{9.1.21}$$

is the **Kronecker sum** of A and B. Since

$$\begin{aligned}\big[(I_m \otimes A) + (B \otimes I_n)\big](y_j \otimes x_i) &= y_j \otimes (Ax_i) + (By_j) \otimes x_i \\ &= (\lambda_i + \mu_j)(y_j \otimes x_i).\end{aligned} \tag{9.1.22}$$

the nm eigenvalues of the Kronecker sum equal the sum of all pairs of eigenvalues of A and B

Review Questions

1. How are the eigenvalues and eigenvectors of A affected by a similarity transformation?

2. What is meant by a (right) invariant subspace of A? Describe how a basis for an invariant subspace can be used to construct a similarity transformation of A to block triangular form. How does such a transformation simplify the computation of the eigenvalues of A?

3. What is meant by the algebraic multiplicity and the geometric multiplicity of an eigenvalue of A? When is a matrix said to be defective?

Problems

1. A matrix $A \in \mathbf{R}^{n\times n}$ is called nilpotent if $A^k = 0$ for some $k > 0$. Show that a nilpotent matrix can only have 0 as an eigenvalue.

2. Show that if λ is an eigenvalue of a unitary matrix U then $|\lambda| = 1$.

3. Let $A \in \mathbf{R}^{m\times n}$ and $B \in \mathbf{R}^{n\times m}$. Show that

$$X^{-1}\begin{pmatrix} AB & 0 \\ B & 0 \end{pmatrix} X = \begin{pmatrix} 0 & 0 \\ B & BA \end{pmatrix}, \qquad X = \begin{pmatrix} I & A \\ 0 & I \end{pmatrix}.$$

Conclude that the nonzero eigenvalues of $AB \in \mathbf{R}^{m\times m}$ and $BA \in \mathbf{R}^{n\times n}$ are the same.

4. (a) Let $A = xy^T$, where x and y are vectors in $\mathbf{R}^n$, $n \geq 2$. Show that 0 is an eigenvalue of A with multiplicity at least $n-1$, and that the remaining eigenvalue is $\lambda = y^T x$.

(b) What are the eigenvalues of a Householder reflector $P = I - 2uu^T$, $\|u\|_2 = 1$?

5. What are the eigenvalues of a Givens' rotation

$$R(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}?$$

When are the eigenvalues real?

6. An upper Hessenberg matrix is called unreduced if all its subdiagonal elements are nonzero. Show that if $H \in \mathbf{R}^{n\times n}$ is an unreduced Hessenberg matrix, then $\operatorname{rank}(H) \geq n-1$, and that therefore if H has a multiple eigenvalue it must be defective.

7. Let $A \in \mathbf{C}^{n\times n}$ be an Hermitian matrix, λ an eigenvalue of A, and z the corresponding eigenvector. Let $A = S + iK$, $z = x + iy$, where S, K, x, y are real. Show that λ is a double eigenvalue of the real symmetric matrix

$$\begin{pmatrix} S & -K \\ K & S \end{pmatrix} \in \mathbf{R}^{2n\times 2n},$$

and determine two corresponding eigenvectors.

8. Show that the matrix

$$K_n = \begin{pmatrix} -a_1 & -a_2 & \cdots & -a_{n-1} & -a_n \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

has the characteristic polynomial

$$p(\lambda) = (-1)^n(\lambda^n + a_1\lambda^{n-1} + \cdots + a_{n-1}\lambda + a_n).$$

K_n is called the **companion matrix** of $p(\lambda)$. Determine the eigenvectors of K_n corresponding to an eigenvalue λ, and show that there is only one eigenvector even when λ is a multiple eigenvalue.

Remark: The term companion matrix is sometimes used for slightly different matrices, where the coefficients of the polynomial appear, e.g., in the last row or in the last column.

9. Draw the graphs $G(A)$, $G(B)$ and $G(C)$, where

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

Show that A and C are irreducible but B is reducible.

9.2 Canonical Forms and Matrix Functions

Using similarity transformations it is possible to transform a matrix into one of several canonical forms, which reveal its eigenvalues and gives information about the eigenvectors. These canonical forms are useful also for extending analytical functions of one variable to matrix arguments.

9.2.1 The Schur Normal Form

The computationally most useful of the canonical forms is the triangular, or **Schur normal form**.

Theorem 9.2.1. Schur Normal Form.

Given $A \in \mathbf{C}^{n\times n}$ there exists a unitary matrix $U \in \mathbf{C}^{n\times n}$ such that

$$U^H AU = T = D + N, \tag{9.2.1}$$

where T is upper triangular, N strictly upper triangular, $D = \operatorname{diag}(\lambda_1, \cdots, \lambda_n)$, and $\lambda_i, i = 1, ..., n$ are the eigenvalues of A. Furthermore, U can be chosen so that the eigenvalues appear in arbitrary order in D.

Proof. The proof is by induction on the order n of the matrix A. For $n = 1$ the theorem is trivially true. Assume the theorem holds for all matrices of order $n-1$. We will show that it holds for any matrix $A \in \mathbf{C}^{n\times n}$.

Let λ be an arbitrary eigenvalue of A. Then, $Ax = \lambda x$, for some $x \neq 0$ and we let $u_1 = x/\|x\|_2$. Then we can always find $U_2 \in \mathbf{C}^{n\times n-1}$ such that $U = (u_1,\ U_2)$ is a unitary matrix. Since $AU = A(u_1, U_2) = (\lambda u_1,\ AU_2)$ we have

$$U^H AU = \begin{pmatrix} u_1^H \\ U_2^H \end{pmatrix} AU = \begin{pmatrix} \lambda u_1^H u_1 & u_1^H AU_2 \\ \lambda U_2^H u_1 & U_2^H AU_2 \end{pmatrix} = \begin{pmatrix} \lambda & w^H \\ 0 & B \end{pmatrix}.$$

Here B is of order $n-1$ and by the induction hypothesis there exists a unitary matrix $\tilde{U}$ such that $\tilde{U}^H B\tilde{U} = \tilde{T}$. Then

$$\overline{U}^H A\overline{U} = T = \begin{pmatrix} \lambda & w^H\tilde{U} \\ 0 & \tilde{T} \end{pmatrix}, \quad \overline{U} = U\begin{pmatrix} 1 & 0 \\ 0 & \tilde{U} \end{pmatrix},$$

where $\overline{U}$ is unitary. From the above it is obvious that we can choose U to get the eigenvalues of A arbitrarily ordered on the diagonal of T. □

The advantage of the Schur normal form is that it can be obtained using a numerically stable unitary transformation. The eigenvalues of A are displayed on the diagonal. The columns in $U = (u_1, u_2, \ldots, u_n)$ are called **Schur vectors**. It is easy to verify that the nested sequence of subspaces

$$S_k = \text{span}[u_1, \ldots, u_k], \qquad k = 1, \ldots, n,$$

are invariant subspaces. However, of the Schur vectors in general only u_1 is an eigenvector.

If the matrix A is real, we would like to restrict ourselves to real similarity transformations, since otherwise we introduce complex elements in $U^{-1}AU$. If A has complex eigenvalues, then A obviously cannot be reduced to triangular form by a real orthogonal transformation. For a real matrix A the eigenvalues occur in complex conjugate pairs, and it is possible to reduce A to block triangular form T, with 1×1 and 2×2 diagonal blocks, in which the 2×2 blocks correspond to pairs of complex conjugate eigenvalues. T is then said to be in **quasi-triangular** form.

Theorem 9.2.2. The Real Schur Form.

Given $A \in \mathbf{R}^{n\times n}$ there exists a real orthogonal matrix $Q \in \mathbf{R}^{n\times n}$ such that

$$Q^T AQ = T = D + N, \tag{9.2.2}$$

where T is real block upper triangular, D is block diagonal with 1×1 and 2×2 blocks, and where all the 2×2 blocks have complex conjugate eigenvalues.

Proof. Let A have the complex eigenvalue $\lambda \neq \bar{\lambda}$ corresponding to the eigenvector x. Then, since $A\bar{x} = \bar{\lambda}\bar{x}$, also $\bar{\lambda}$ is an eigenvalue with eigenvector $\bar{x} \neq x$, and $\mathcal{R}(x, \bar{x})$ is an invariant subspace of dimension 2. Let

$$X_1 = (x_1, x_2), \quad x_1 = x + \bar{x}, \quad x_2 = i(x - \bar{x})$$

be a real basis for this invariant subspace. Then $AX_1 = X_1 M$ where $M \in \mathbf{R}^{2\times 2}$ has eigenvalues λ and $\bar{\lambda}$. Let $X_1 = Q\begin{pmatrix} R \\ 0 \end{pmatrix} = Q_1 R$ be the QR decomposition of X_1. Then $AQ_1R = Q_1RM$ or $AQ_1 = Q_1P$, where $P = RMR^{-1} \in \mathbf{R}^{2\times 2}$ is similar to M. Using (9.1.16) with $X = Q$, we find that

$$Q^T AQ = \begin{pmatrix} P & W^H \\ 0 & B \end{pmatrix}.$$

where P has eigenvalues λ and $\bar{\lambda}$. An induction argument completes the proof. □

We now introduce a class of matrices for which the Schur normal form is diagonal.

Definition 9.2.3.
A matrix $A \in \mathbf{C}^{\mathbf{n}\times\mathbf{n}}$ is said to be **normal** *if*

$$A^H A = AA^H. \tag{9.2.3}$$

If A is normal then for unitary U so is $U^H AU$, since

$$(U^H AU)^H U^H AU = U^H (A^H A)U = U^H (AA^H)U = U^H AU(U^H AU)^H.$$

It follows that the upper triangular matrix T in the Schur normal form is normal,

$$T^H T = TT^H, \qquad T = \begin{pmatrix} \lambda_1 & t_{12} & \dots & t_{1n} \\ & \lambda_2 & \dots & t_{2n} \\ & & \ddots & \vdots \\ & & & \lambda_n \end{pmatrix},$$

Equating the $(1,1)$-element on both sides of the equation $T^H T = TT^H$ we get $|\lambda_1|^2 = |\lambda_1|^2 + \sum_{j=2}^n |t_{1j}|^2$, and so $t_{1j} = 0$, $j = 2, \dots, n$. In the same way it can be shown that all the other nondiagonal elements in T vanishes, and so T is diagonal.

Important classes of normal matrices are Hermitian ($A = A^H$), skew-Hermitian ($A^H = -A$), unitary ($A^{-1} = A^H$) and circulant matrices (see Problem 9.1.10). Hermitian matrices have real eigenvalues, skew-Hermitian matrices have imaginary eigenvalues, and unitary matrices have eigenvalues on the unit circle.

Theorem 9.2.4.
A matrix $A \in \mathbf{C}^{n\times n}$ is normal, $A^H A = AA^H$, if and only if A can be unitarily diagonalized, i.e., there exists a unitary matrix $U \in \mathbf{C}^{n\times n}$ such that

$$U^H AU = D = \operatorname{diag}(\lambda_1, \cdots, \lambda_n).$$

Proof. If A is normal, then it follows from the above that the matrix T in the Schur normal form is diagonal. If on the other hand A is unitarily diagonalizable then we immediately have that

$$A^H A = UD^H DU^H = UDD^H U^H = AA^H.$$

□

It follows in particular that any Hermitian matrix may be decomposed into

$$A = U\Lambda U^H = \sum_{i=1}^n \lambda_i u_i u_i^H. \tag{9.2.4}$$

with λ_i real. In the special case that A is real and symmetric we can take U to be real and orthogonal, $U = Q = (q_1, \ldots, q_n)$, where q_i are orthonormal eigenvectors. Note that in (9.2.4) $u_i u_i^H$ is the unitary projection matrix that projects unitarily onto the eigenvector u_i. We can also write $A = \sum_j \lambda P_j$, where the sum is taken over the *distinct* eigenvalues of A, and P_j projects $\mathbf{C^n}$ unitarily onto the eigenspace belonging to λ_j. (This comes closer to the formulation given in functional analysis.)

Note that although U in the Schur normal form (9.2.1) is not unique, $\|N\|_F$ is independent of the choice of U, and

$$\Delta_F^2(A) \equiv \|N\|_F^2 = \|A\|_F^2 - \sum_{i=1}^{n} |\lambda_i|^2.$$

The quantity $\Delta_F(A)$ is called the **departure from normality** of A.

9.2.2 Sylvester's Equation and Jordan's Canonical Form

Let the matrix A have the block triangular form

$$A = \begin{pmatrix} B & C \\ 0 & D \end{pmatrix}, \tag{9.2.5}$$

where B and D are square. Suppose that we wish to reduce A to **block diagonal** form by a similarity transformation of the form

$$P = \begin{pmatrix} I & Q \\ 0 & I \end{pmatrix}, \qquad P^{-1} = \begin{pmatrix} I & -Q \\ 0 & I \end{pmatrix}.$$

This gives the result

$$P^{-1}AP = \begin{pmatrix} I & -Q \\ 0 & I \end{pmatrix} \begin{pmatrix} B & C \\ 0 & D \end{pmatrix} \begin{pmatrix} I & Q \\ 0 & I \end{pmatrix} = \begin{pmatrix} B & C - QD + BQ \\ 0 & D \end{pmatrix}.$$

The result is a block diagonal matrix if and only if $BQ - QD = -C$. This equation, which is a linear equation in the elements of Q, is called **Sylvester's equation**[2]

We will investigate the existence and uniqueness of solutions to the general Sylvester equation

$$AX - XB = C, \quad X \in \mathbf{R}^{n\times m}, \tag{9.2.6}$$

where $A \in \mathbf{R}^{n\times n}$, $B \in \mathbf{R}^{m\times m}$. We prove the following result.

Theorem 9.2.5.

The matrix equation (9.2.6) *has a unique solution if and only if*

$$\lambda(A) \cap \lambda(B) = \emptyset.$$

Proof. From Theorem 9.2.1 follows the existence of the Schur decompositions

$$U_1^H A U_1 = S, \qquad U_2^H B U_2 = T,$$

[2]James Joseph Sylvester English mathematician (1814–1893) considered the homogenous case in 1884.

where S and T are upper triangular and U_1 and U_2 are unitary matrices. Using these decompositions (9.2.6) can be reduced to

$$SY - YT = F, \qquad Y = U_1^H X U_2, \quad F = U_1^H C U_2.$$

Expanding this equation by columns gives

$$S\,(\,y_1 \quad y_2 \quad y_3 \cdots) - (\,y_1 \quad y_2 \quad y_3 \cdots) \begin{pmatrix} t_{11} & t_{12} & t_{13} \cdots \\ 0 & t_{22} & t_{33} \cdots \\ 0 & 0 & t_{33} \cdots \\ \vdots & \vdots & \vdots \end{pmatrix} = (\,f_1 \quad f_2 \quad f_3 \cdots)\,. \tag{9.2.7}$$

The first column of the system (9.2.7) has the form

$$Sy_1 - t_{11}y_1 = (S - t_{11}I)y_1 = d_1.$$

Here t_{11} is an eigenvalue of T and hence is *not* an eigenvalue of S. Therefore the triangular matrix $S - t_{11}I$ is not singular and we can solve for y_1. Now suppose that we have found $y_1, \ldots, y_{k-1}$. From the kth column of the system

$$(S - t_{kk}I)y_k = d_k + \sum_{i=1}^{k} t_{ik}y_i.$$

Here the right hand side is known and, by the argument above, the triangular matrix $S - t_{kk}I$ nonsingular. Hence it can be solved for y_k. The proof now follows by induction. □

If we have an algorithm for computing the Schur decompositions this proof gives an algorithm for solving the Sylvester equation. It involves solving m triangular equations and requires $O(mn^2)$ operations.

An important special case of (9.2.6) is the **Lyapunov equation**

$$AX + XA^H = C. \tag{9.2.8}$$

Here $B = -A^H$, and hence by Theorem 9.2.5 this equation has a unique solution if and only if the eigenvalues of A satisfy $\lambda_i + \bar{\lambda}_j \neq 0$ for all i and j. Further, if $C^H = C$ the solution X is Hermitian. In particular, if all eigenvalues of A have negative real part, then all eigenvalues of $-A^H$ have positive real part, and the assumption is satisfied.

We have seen that a given block triangular matrix (9.2.5) can be transformed by a similarity transformation to block diagonal form provided that B and C have disjoint spectra. The importance of this contruction is that it cann be applied recursively.

If A is not normal, then the matrix T in its Schur normal form cannot be diagonal. To transform T to a form closer to a diagonal matrix we have to use *non-unitary similarities*. By Theorem 9.2.1 we can order the eigenvalues so that in the Schur normal form

$$D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.$$

We now show how to obtain the following block diagonal form:

Theorem 9.2.6. Block Diagonal Decomposition.

Let the distinct eigenvalues of A be $\lambda_1, \cdots, \lambda_k$, and in the Schur normal form let $D = \operatorname{diag}(D_1, \ldots, D_k)$, $\quad D_i = \lambda_i I, \quad i = 1, \ldots, k$. Then there exists a nonsingular matrix Z such that

$$Z^{-1}U^H AUZ = Z^{-1}TZ = \operatorname{diag}(\lambda_1 I + N_1, \cdots, \lambda_k I + N_k),$$

where N_i, $i = 1, \ldots, k$ are strictly upper triangular. In particular, if the matrix A has n distinct eigenvalues the matrix D diagonal.

Proof. Consider first the matrix $T = \begin{pmatrix} \lambda_1 & t \\ 0 & \lambda_2 \end{pmatrix} \in \mathbf{C}^{2\times 2}$, where $\lambda_1 \neq \lambda_2$. Perform the similarity transformation

$$M^{-1}TM = \begin{pmatrix} 1 & -m \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 & t \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} 1 & m \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \lambda_1 & m(\lambda_1 - \lambda_2) + t \\ 0 & \lambda_2 \end{pmatrix}.$$

where M is an upper triangular elementary elimination matrix, see Section 7.3.5. By taking $m = t/(\lambda_2 - \lambda_1)$, we can annihilate the off-diagonal element in T.

In the general case let t_{ij} be an element in T outside the block diagonal. Let M_{ij} be a matrix which differs from the unit matrix only in the (i,j)th element, which is equal to m_{ij}. Then as above we can choose m_{ij} so that the element (i,j) is annihilated by the similarity transformation $M_{ij}^{-1}TM_{ij}$. Since T is upper triangular this transformation will not affect any already annihilated off-diagonal elements in T with indices (i', j') if $j' - i' < j - i$. Hence, we can annihilate all elements t_{ij} outside the block diagonal in this way, starting with the elements on the diagonal closest to the main diagonal and working outwards. For example, in a case with 3 blocks of orders $2, 2, 1$ the elements are eliminated in the order

$$\begin{pmatrix} \times & \times & 2 & 3 & 4 \\ & \times & 1 & 2 & 3 \\ & & \times & \times & 2 \\ & & & \times & 1 \\ & & & & \times \end{pmatrix}.$$

Further details of the proof is left to the reader. □

A matrix which does not have n linearly independent eigenvectors is defective and cannot be similar to a diagonal matrix. We now state without proof the following fundamental Jordan Canonical Form[3] For a proof based on the block diagonal decomposition in Theorem 9.2.6, see Fletcher and Sorensen [12, 1983].

Theorem 9.2.7. Jordan Canonical Form.

If $A \in \mathbf{C}^{n\times n}$, then there is a nonsingular matrix $X \in \mathbf{C}^{n\times n}$, such that

$$X^{-1}AX = J = \operatorname{diag}(J_{m_1}(\lambda_1), \cdots, J_{m_t}(\lambda_t)), \tag{9.2.9}$$

[3] Marie Ennemond Camille Jordan (1838–1922), French mathematician, professor at École Polytechnique and Collége de France. Jordan made important contributions to finte group theory, linear and multilinear algebra as well as differential equations.

where

$$J_{m_i}(\lambda_i) = \begin{pmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{pmatrix} = \lambda_i I + S \in \mathbf{C}^{m_i \times m_i}, \quad m_i \geq 1,$$

The numbers $m_1, \ldots, m_t$ are unique and $\sum_{i=1}^{t} m_i = n$. To each Jordan block $J_{m_i}(\lambda_i)$ there corresponds exactly one eigenvector. Hence the number of Jordan blocks corresponding to a multiple eigenvalue λ equals the geometric multiplicity of λ.

The form (9.2.9) is called the Jordan canonical form of A, and is unique up to the ordering of the Jordan blocks. Note that the same eigenvalue may appear in several different Jordan blocks. A matrix for which this occurs is called **derogatory**. The Jordan canonical form has the advantage that it displays all eigenvalues and eigenvectors of A explicitly. A serious disadvantage is that the Jordan canonical form is not in general a continuous function of the elements of A. For this reason the Jordan canonical form of a nondiagonalizable matrix may be very difficult to determine numerically.

Example 9.2.1.

Consider the matrices of the form

$$J_m(\lambda, \epsilon) = \begin{pmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ \epsilon & & & \lambda \end{pmatrix} \in \mathbf{C}^{m \times m}.$$

The matrix $J_m(\lambda, 0)$ has an eigenvalue equal to λ of multiplicity m, and is in Jordan canonical form. For any $\epsilon > 0$ the matrix $J_m(\lambda, \epsilon)$ has m distinct eigenvalues μ_i, $i = 1, \ldots, m$, which are the roots of the equation $(\lambda - \mu)^m - (-1)^m \epsilon = 0$. Hence $J_m(\lambda, \epsilon)$ is diagonalizable for any $\epsilon \neq 0$, and its eigenvalues λ_i satisfy$|\lambda_i - \lambda| = |\epsilon|^{1/m}$. For example, if $m = 10$ and $\epsilon = 10^{-10}$, then the perturbation is of size 0.1.

If $X = (x_1, x_2, \ldots, x_n)$ is the matrix in (9.2.9), then

$$Ax_1 = \lambda_1 x_1, \quad Ax_{i+1} = \lambda_1 x_{i+1} + x_i, \quad i = 1, \ldots, m_1 - 1.$$

The vectors $x_2, \ldots, x_{m_1}$ are called **principal vectors** of the matrix A. Similar relations holds for the other Jordan blocks.

The minimal polynomial of A can be read off from its Jordan canonical form. Consider a Jordan block $J_m(\lambda) = \lambda I + N$ of order m and put $q(z) = (z - \lambda)^j$. Then we have $q(J_m(\lambda)) = N^j = 0$ for $j \geq m$. The minimal polynomial of a matrix A with the *distinct* eigenvalues $\lambda_1, \ldots, \lambda_k$ then has the form

$$q(z) = (z - \lambda_1)^{m_1} (z - \lambda_2)^{m_2} \cdots (z - \lambda_k)^{m_k}, \tag{9.2.10}$$

where m_j is the highest dimension of any Jordan box corresponding to the eigenvalue λ_j, $j = 1, \ldots, k$.

As a corollary we obtain **Cayley–Hamilton theorem**, which states that the characteristic polynomial $p(z)$ of a matrix A satisfies $p(A) = 0$. The polynomials

$$\pi_i(z) = \det\left(zI - J_{m_i}(\lambda_i)\right) = (z - \lambda_i)^{m_i}$$

are called **elementary divisors** of A. They divide the characteristic polynomial of A. The elementary divisors of the matrix A are all linear if and only if the Jordan canonical form is diagonal.

We end with an approximation theorem due to Bellman, which sometimes makes it possible to avoid the complication of the Jordan canonical form.

Theorem 9.2.8.

Let $A \in \mathbf{C}^{n\times n}$ be a given matrix. Then for any $\epsilon > 0$ there exists a matrix B with $\|A - B\|_2 \leq \epsilon$, such that B has n distinct eigenvalues. Hence, the class of diagonalizable matrices is dense in $\mathbf{C}^{n\times n}$.

Proof. Let $X^{-1}AX = J$ be the Jordan canonical form of A. Then, by a slight extension of Example 9.2.1 it follows that there is a matrix $J(\delta)$ with distinct eigenvalues such that $\|J - J(\delta)\|_2 = \delta$. (Show this!) Take $B = XJ(\delta)X^{-1}$. Then

$$\|A - B\|_2 \leq \epsilon, \quad \epsilon = \delta\|X\|_2\|X^{-1}\|_2.$$

□

9.2.3 Convergence of Matrix Power Series

We start with a definition of the limit of a sequence of matrices:

Definition 9.2.9.

An infinite sequence of matrices $A_1, A_2, \ldots$ is said to converge to a matrix A, $\lim_{n\to\infty} A_n = A$, if

$$\lim_{n\to\infty} \|A_n - A\| = 0.$$

From the equivalence of norms in a finite dimensional vector space it follows that convergence is independent of the choice of norm. The particular choice $\|\cdot\|_\infty$ shows that convergence of vectors in $\mathbf{R}^n$ is equivalent to convergence of the n sequences of scalars formed by the components of the vectors. By considering matrices in $\mathbf{R}^{m\times n}$ as vectors in $\mathbf{R}^{mn}$ the same conclusion holds for matrices.

An infinite sum of matrices is defined by:

$$\sum_{k=0}^{\infty} B_k = \lim_{n\to\infty} S_n, \quad S_n = \sum_{k=0}^{n} B_k.$$

In a similar manner we can define $\lim_{z\to\infty} A(z), A'(z)$, etc., for **matrix-valued functions** of a complex variable $z \in \mathbf{C}$.

Theorem 9.2.10.
If $\|\cdot\|$ is any matrix norm, and $\sum_{k=0}^{\infty}\|B_k\|$ is convergent, then $\sum_{k=0}^{\infty}B_k$ is convergent.

Proof. The proof follows from the triangle inequality $\|\sum_{k=0}^{n}B_k\| \le \sum_{k=0}^{n}\|B_k\|$ and the Cauchy condition for convergence. (Note that the converse of this theorem is not necessarily true.) □

A power series $\sum_{k=0}^{\infty}B_k z^n, z \in \mathbf{C}$, has a *circle of convergence* in the z-plane which is equivalent to the smallest of the circles of convergence corresponding to the series for the matrix elements. In the interior of the convergence circle, formal operations such as term-wise differentiation and integration with respect to z are valid for the element series and therefore also for matrix series.

We now investigate the convergence of matrix power series. First we prove a theorem which is also of fundamental importance for the theory of convergence of iterative methods studied in Chapter 10. We first recall the the following result:

Lemma 9.2.11. *For any consistent matrix norm*

$$\rho(A) \le \|A\|, \tag{9.2.11}$$

where $\rho(A) = \max_i |\lambda_i(A)|$ is the **spectral radius** *of A.*

Proof. If λ is an eigenvalue of A then there is a nonzero vector x such that $\lambda x = Ax$. Taking norms we get $|\lambda|\,\|x\| \le \|A\|\|x\|$. Dividing with $\|x\|$ the result follows. □

We now return to the question of convergence of matrix series.

Theorem 9.2.12.
If the infinite series $f(z) = \sum_{k=0}^{\infty}a_k z^k$ has radius of convergence r, then the matrix series $f(A) = \sum_{k=0}^{\infty}a_k A^k$ converges if $\rho < r$, where $\rho = \rho(A)$ is the spectral radius of A. If $\rho > r$, then the matrix series diverges; the case $\rho = r$ is a "questionable case".

Proof. By Theorem 9.2.10 the matrix series $\sum_{k=0}^{\infty}a_k A^k$ converges if the series $\sum_{k=0}^{\infty}|a_k|\|A^k\|$ converges. By Theorem 9.2.13 for any $\epsilon > 0$ there is a matrix norm such that $\|A\|_T = \rho + \epsilon$. If $\rho < r$ then we can choose r_1 such that $\rho(A) \le r_1 < r$, and we have

$$\|A^k\|_T \le \|A\|_T^k \le (\rho+\epsilon)^k = O(r_1^k).$$

Here $\sum_{k=0}^{\infty}|a_k|r_1^k$ converges, and hence $\sum_{k=0}^{\infty}|a_k|\|A^k\|$ converges. If $\rho > r$, let $Ax = \lambda x$ with $|\lambda| = \rho$. Then $A^k x = \lambda^k x$, and since $\sum_{k=0}^{\infty}a_k\lambda^k$ diverges $\sum_{k=0}^{\infty}a_k A^k$ cannot converge. □

Theorem 9.2.13.

Given a matrix $A \in \mathbf{R}^{n\times n}$ with spectral radius $\rho = \rho(A)$. Denote by $\|\cdot\|$ any l_p-norm, $1 \le p \le \infty$, and set $\|A\|_T = \|T^{-1}AT\|$. Then the following holds:

(a) *If A has no defective eigenvalues with absolute value ρ then there exists a nonsingular matrix T such that*

$$\|A\|_T = \rho.$$

(b) *If A has a defective eigenvalue with absolute value ρ then for every $\epsilon > 0$ there exists a nonsingular matrix $T(\epsilon)$ such that*

$$\|A\|_{T(\epsilon)} \le \rho + \epsilon.$$

In this case, the condition number $\kappa(T(\epsilon)) \to \infty$ like ϵ^{1-m^*} as $\epsilon \to 0$, where $m^* > 1$ is the largest order of a Jordan block belonging to an eigenvalue λ with $|\lambda| = \rho$.

Proof. If A is diagonalizable, we can simply take T as the diagonalizing transformation. Then clearly $\|A\|_T = \|D\| = \rho$, where $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. In the general case, we first bring A to Jordan canonical form, $X^{-1}AX = J$, where

$$J = \operatorname{diag}\big(J_1(\lambda_1), \cdots, J_t(\lambda_t)\big), \quad J_i(\lambda_i) = \lambda_i I + N_i \in \mathbf{C}^{m_i\times m_i}, \quad m_i \ge 1,$$

and $J_i(\lambda_i)$ is a Jordan block. We shall find a diagonal matrix $D = \operatorname{diag}(D_1, \ldots, D_t)$, such that a similarity transformation with $T = XD$, $K = T^{-1}AT = D^{-1}JD$ makes K close to the diagonal of J. Note that $\|A\|_T = \|K\|$, and

$$K = \operatorname{diag}(K_1, K_2, \ldots, K_t), \quad K_i = D_i^{-1}J_i(\lambda_i)D_i.$$

If $m_i = 1$, we set $D_i = 1$, hence $\|K_i\| = |\lambda_i|$. Otherwise we choose

$$D_i = \operatorname{diag}\left(1, \delta_i, \delta_i^2, \ldots, \delta_i^{m_i-1}\right), \quad \delta_i > 0. \tag{9.2.12}$$

Then $K_i = \lambda_i I + \delta_i N_i$, and $\|K\| = \max_i(\|K_i\|)$. (Verify this!) We have $\|N_i\| \le 1$, because $N_i x = (x_2, x_3, ..., x_{m_i}, 0)^T$, so $\|N_i x\| \le \|x\|$ for all vectors x. Hence,

$$\|K_i\| \le |\lambda_i| + \delta_i. \tag{9.2.13}$$

If $m_i > 1$ and $|\lambda_i| < \rho$, we choose $\delta_i = \rho - |\lambda_i|$, hence $\|K_i\| \le \rho$. This proves case (a).

In case (b), $m_i > 1$ for at least one eigenvalue with $|\lambda_i| = \rho$. Let $M = \{i : |\lambda_i| = \rho\}$, and choose $\delta_i = \epsilon$, for $i \in M$. Then by (9.2.13) $\|K_i\| \le \rho + \epsilon$, for $i \in M$, while $\|K_i\| \le \rho$, for $i \notin M$. Hence $\|K\| = \max_i \|K_i\| = \rho + \epsilon$, and the first part of statement (b) now follows.

With $T(\epsilon) = XD(\epsilon)$, we have that

$$\kappa\big(D(\epsilon)\big)/\kappa(X) \le \kappa\big(T(\epsilon)\big) \le \kappa\big(D(\epsilon)\big)\kappa(X).$$

When $|\lambda_i| = \rho$ we have $\delta_i = \epsilon$, and it follows from (9.2.12) that $\kappa(D_i)$ grows like ϵ^{1-m_i}. Since $\kappa(D) = \max_i \kappa(D_i)$, and for $|\lambda_i| < \rho$ the condition numbers of D_i are bounded, this proves the second part of statement (b). □

Note that $1/\kappa(T) \le \|A\|_T/\|A\| \le \kappa(T)$. For every natural number n, we have, in case (a), $\|A^n\|_T \le \|A\|_T^n = \rho(A)^n$. Hence

$$\|A^n\|_p \le \kappa(T)\|A^n\|_T \le \kappa(T)\rho^n.$$

In case (b), the same holds, if ρ, T are replaced by, respectively, $\rho+\epsilon, T(\epsilon)$. See also Problem 9.

If only statement (b) is needed, a more elementary proof can be found by a similar argument applied to the Schur canonical form instead of the Jordan canonical form. Since X is unitary in this case, one has a better control of the condition numbers, which is of particular importance in some applications to partial differential equations, where one needs to apply this kind of theorem to a *family of matrices* instead of just one individual matrix. This leads to the famous *matrix theorems of Kreiss*, see Theorems 13.8.6–13.8.7.

For some classes of matrices, an efficient (or rather efficient) norm can be found more easily than by the construction used in the proof of Theorem 9.2.13 This may have other advantages as well, e.g., a better conditioned T. Consider, for example, the weighted max-norm

$$\|A\|_w = \|T^{-1}AT\|_\infty = \max_i \sum_j |a_{ij}|w_j/w_i,$$

where $T = \operatorname{diag}(w_1, \ldots, w_n) > 0$, and $\kappa(T) = \max w_i / \min w_i$. We then note that if we can find a positive vector w such that $|A|w \le \alpha w$, then $\|A\|_w \le \alpha$.

9.2.4 Matrix Functions

The **matrix exponential** e^{At}, where A is a constant matrix, can be defined by the series expansion

$$e^{At} = I + At + \frac{1}{2!}A^2t^2 + \frac{1}{3!}A^3t^3 + \cdots.$$

This series converges for all A and t since the radius of convergence of the power series $\sum_{k=0}^{\infty} \|A\|^k t^k/k!$ is infinite. The series can thus be differentiated everywhere and

$$\frac{d}{dt}(e^{At}) = A + A^2t + \frac{1}{2!}A^3t^2 + \cdots = Ae^{At}.$$

Hence $y(t) = e^{At}c \in \mathbf{R}^n$ solves the initial value problem for the linear system of ordinary differential equations with constant coefficients

$$dy(t)/dt = Ay(t), \quad y(0) = c. \tag{9.2.14}$$

Such systems occurs in many physical, biological, and economic processes.

Other functions, for example, $\sin(z)$, $\cos(z)$, $\log(z)$, can be similarly defined for matrix arguments from their Taylor series representation. In general, if $f(z)$ is an analytic function with Taylor expansion $f(z) = \sum_{k=0}^{\infty} a_k z^k$, then we define

$$f(A) = \sum_{k=0}^{\infty} a_k A^k.$$

We now turn to the question of how to define analytic functions of matrices in general. If the matrix A is diagonalizable, $A = X\Lambda X^{-1}$, we define

$$f(A) = X\text{diag}\big(f(\lambda_1), \ldots, f(\lambda_n)\big)X^{-1} = Xf(\Lambda)X^{-1}. \tag{9.2.15}$$

This expresses the matrix function $f(A)$ in terms of the function f evaluated at the spectrum of A and is often the most convenient way to compute $f(A)$.

For the case when A is not diagonalizable we first give an explicit form for the kth power of a Jordan block $J_m(\lambda) = \lambda I + N$. Since $N^j = 0$ for $j \geq m$ we get using the binomial theorem

$$J_m^k(\lambda) = (\lambda I + N)^k = \lambda^k I + \sum_{p=1}^{\min(m-1,k)} \binom{k}{p} \lambda^{k-p} N^p, \quad k \geq 1.$$

Since an analytic function can be represented by its Taylor series we are led to the following definition:

Definition 9.2.14.

Suppose that the analytic function $f(z)$ is regular for $z \in D \subset \mathbf{C}$, where D is a simply connected region, which contains the spectrum of A in its interior. Let

$$A = XJX^{-1} = X\text{diag}\big(J_{m_1}(\lambda_1), \cdots, J_{m_t}(\lambda_t)\big)X^{-1}$$

be the Jordan canonical form of A. We then define

$$f(A) = X\,\text{diag}\Big(f\big(J_{m_1}(\lambda_1)\big), \cdots, f\big(J_{m_t}(\lambda_t)\big)\Big)X^{-1}. \tag{9.2.16}$$

where the analytic function f of a Jordan block is

$$f(J_m) = f(\lambda)I + \sum_{p=1}^{m-1} \frac{1}{p!} f^{(p)}(\lambda) N^p. \tag{9.2.17}$$

If A is diagonalizable, $A = X^{-1}\Lambda X$, then for the exponential function we have,

$$\|e^A\|_2 = \kappa(X)e^{\alpha(A)},$$

where $\alpha(A) = \max_i \Re\lambda_i$ is the **spectral abscissa** of A and $\kappa(X)$ denotes the condition number of the eigenvector matrix. If A is normal, then V is orthogonal and $\kappa(V) = 1$.

One can show that for every non-singular matrix T it holds

$$f(T^{-1}AT) = T^{-1}f(A)T. \tag{9.2.18}$$

With this definition, the theory of analytic functions of a matrix variable closely follows the theory of a complex variable. If $\lim_{n\to\infty} f_n(z) = f(z)$ for $z \in D$, then $\lim_{n\to\infty} f_n(J(\lambda_i)) = f(J(\lambda_i))$. Hence *if the spectrum of A lies in the interior of D* then $\lim_{n\to\infty} f_n(A) = f(A)$. This allows us to deal with operations involving limit processes.

The following important theorem can be obtained, which shows that Definition 9.2.14 is consistent with the more restricted definition (by a power series) given in Theorem 9.2.12.

Theorem 9.2.15.

All identities which hold for analytic functions of one complex variable z for $z \in D \subset \mathbf{C}$, where D is a simply connected region, also hold for analytic functions of one matrix variable A if the spectrum of A is contained in the interior of D. The identities also hold if A has eigenvalues on the boundary of D, provided these are not defective.

Example 9.2.2.

We have, for example,

$$\cos^2 A + \sin^2 A = I, \qquad \forall A;$$

$$\ln(I - A) = -\sum_{n=1}^{\infty} \frac{1}{n} A^n, \qquad \rho(A) < 1;$$

$$\int_0^{\infty} e^{-st} e^{At} dt = (sI - A)^{-1}, \qquad Re(\lambda_i) < Re(s);$$

Further, if $f(z)$ is analytic inside C, and if the whole spectrum of A is inside C, we have (cf. Problem 9)

$$\frac{1}{2\pi i} \int_C (zI - A)^{-1} f(z) dz = f(A).$$

Observe also that, for two arbitrary analytic functions f and g, which satisfy the condition of the definition, $f(A)g(A) = g(A)f(A)$. However, when several non-commutative matrices are involved, one can no longer use the usual formulas for analytic functions.

Example 9.2.3.

$e^{(A+B)t} = e^{At}e^{Bt}$ for all t if and only if $BA = AB$. We have

$$e^{At}e^{Bt} = \sum_{p=0}^{\infty} \frac{A^p t^p}{p!} \sum_{q=0}^{\infty} \frac{B^q t^q}{q!} = \sum_{n=0}^{\infty} \frac{t^n}{n!} \sum_{p=0}^{n} \binom{n}{p} A^p B^{n-p}.$$

This is in general not equivalent to

$$e^{(A+B)t} = \sum_{n=0}^{\infty} \frac{t^n}{n!}(A+B)^n.$$

The difference between the coefficients of $t^2/2$ in the two expressions is

$$(A+B)^2 - (A^2 + 2AB + B^2) = BA - AB \neq 0, \qquad \text{if } BA \neq AB.$$

Conversely, if $BA = AB$, then it follows by induction that the binomial theorem holds for $(A+B)^n$, and the two expressions are equal.

Because of its key role in the solution of differential equations methods for computing the matrix exponential and investigation of its qualitative behavior has been studied extensively. A wide variety of methods for computing e^A have been proposed; see Moler and Van Loan [35]. Consider the 2 by 2 upper triangular matrix

$$A = \begin{pmatrix} \lambda & \alpha \\ 0 & \mu \end{pmatrix}.$$

The exponential of this matrix is

$$e^{tA} = \begin{cases} \begin{pmatrix} e^{\lambda t} & \alpha \dfrac{e^{\lambda t} - e^{\mu t}}{\lambda - \mu} \\ 0 & e^{\mu t} \end{pmatrix}, & \text{if } \lambda \neq \mu, \\ \begin{pmatrix} e^{\lambda t} & \alpha t e^{\lambda t} \\ 0 & e^{\mu t} \end{pmatrix}, & \text{if } \lambda = \mu \end{cases}. \tag{9.2.19}$$

When $|\lambda - \mu|$ is small, but not negligible neither of these two expressions are suitable, since severe cancellation will occur in computing the divided difference in the (1,2)-element in (9.2.19). When the same type of difficulty occurs in non-triangular problems of larger size the cure is by no means easy!

Another property of e^{At} that does not occur in the scalar case is illustrated next.

Example 9.2.4. Consider the matrix

$$A = \begin{pmatrix} -1 & 4 \\ 0 & -2 \end{pmatrix}.$$

Since $\max\{-1, -2\} = -1 < 0$ it follows that $\lim_{t\to\infty} e^{tA} = 0$. In Figure 9.2.1 we have plotted $\|e^{tA}\|_2$ as a function of t. The curve has a **hump** illustrating that as t increases some of the elements in e^{tA} first increase before they start to decay.

One of the best methods to compute e^A, the method of scaling and squaring, uses the fundamental relation

$$e^A = (e^{A/m})^m, \quad m = 2^s$$

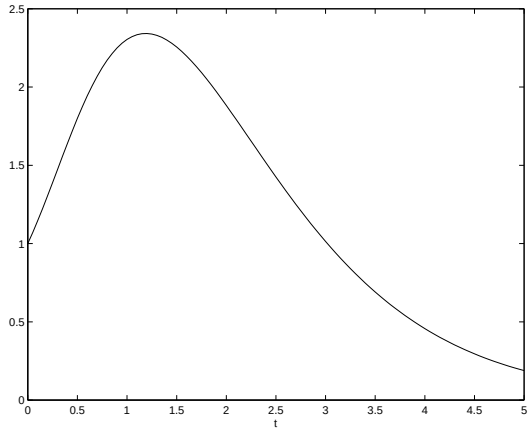


Figure 9.2.1. $\|e^{tA}\|$ *as a function of* t *for the matrix in Example 9.2.4.*

of the exponential function. Here the exponent s is chosen so that $e^{A/m}$ can be reliably computed, e.g. from a Taylor or Padé approximation. Then $e^A = (e^{A/m})^{2^s}$ can be formed by squaring the result s times.

Instead of the Taylor series it is advantageous to use the diagonal Padé approximation of e^x.

$$r_{m,m}(z) = \frac{P_{m,m}(z)}{Q_{m,m}(z)} = \frac{\sum_{j=0}^{m} p_j z^j}{\sum_{j=0}^{n} q_j z^j}, \tag{9.2.20}$$

which are known explicitly for all m. We have

$$p_j = \frac{(2m-j)!\, m!}{(2m)!\,(m-j)!j!}, \qquad q_j = (-1)^j p_j, \quad j = 0:m. \tag{9.2.21}$$

with the error

$$e^z - \frac{P_{m,m}(z)}{Q_{m,m}(z)} = (-1)^k \frac{(m!)^2}{(2m)!(2m+1)!} z^{2m+1} + O(z^{2m+2}). \tag{9.2.22}$$

Note that $P_{m,m}(z) = Q_{m,m}(-z)$, which reflects the property that $e^{-z} = 1/e^z$. The coefficients satisfy the recursion

$$p_0 = 1, \quad p_{j+1} = \frac{m-j}{(2m-j)(j+1)} p_j, \quad j = 0:m-1. \tag{9.2.23}$$

To evaluate a digonal Padé approximant of even degree m we can write

$$\begin{aligned} P_{2m,2m}(A) &= p_{2m}A^{2m} + \cdots + p_2 A^2 + p_0 I \\ &\quad + A(p_{2m-1}A^{2m-2} + \cdots + p_3 A^2 + p_1 I) = U + V. \end{aligned}$$

This can be evaluated with $m+1$ matrix multiplications by forming $A^2, A^4, \ldots, A^{2m}$. Then $Q_{2m}(A) = U - V$ needs no extra matrix multiplications. For an approximation

of odd degree $2m+1$ we write

$$P_{2m+1,2m+1}(A) = A(p_{2m+1}A^{2m} + \cdots + p_3A^2 + p_1I) + p_{2m}A^{2m-2} + \cdots + p_2A^2 + p_0I = U + V.$$

This can be evalauted with the same number of matrix multiplications and $Q_{2m+1}(A) = -U + V$. The final division $P_{k,m}(A)/Q_{m,m}(A)$ is performed by solving

$$Q_{m,m}(A)r_{m,m}(A) = P_{m,m}(A)$$

for $r_{m,m}(A)$ using Gaussian elimination.

The function `expm` in MATLAB uses a scaling such that $2^{-s}\|A\| < 1/2$ and a diagonal Padé approximant of degree $2m = 6$

$$P_{6,6}(z) = 1 + \frac{1}{2}z + \frac{5}{44}z^2 + \frac{1}{66}z^3 + \frac{1}{792}z^4 + \frac{1}{15840}z^5 + \frac{1}{665280}z^6.$$

```
function E = expmv(A);
% EXPMV computes the exponential
% of the matrix A
% Compute scaling parameter
[f,e] = log2(norm(A,'inf'));
s = max(0,e+1);
A = A/2^s;
X = A;
d = 2; c = 1/d;
E = eye(size(A)) + c*A;
D = eye(size(A)) - c*A;
m = 8; p = 1;
for k = 2:m
  d = d*(k*(2*m-k+1))/(m-k+1)
  c = 1/d;
  X = A*X;
  cX = c*X;
  E = E + cX;
  if p, D = D + c*X;
  else, D = D - c*X; end
  p = ~p;
end
E = D\E;
for k = 1:s, E = E*E; end
```

It can be shown ([35, Appendix A]) that then $r_{mm}(2^{-s}A)^{2^s} = e^{A+E}$, where

$$\frac{\|E\|}{\|A\|} < 2^3(2^{-s}\|A\|)^{2m}\frac{(m!)^2}{(2m)!(2m+1)!}.$$

For s and m chosen as in MATLAB this gives $\|E\|/\|A\| < 3.4 \cdot 10^{-16}$, which is close to the unit roundoff in IEEE double precision $2^{-53} = 1.11 \cdot 10^{-16}$. Note that

this backward error result does not guarantee an accurate result. If the problem is inherently sensitive to perturbations the error can be large.

The analysis does not take roundoff errors in the squaring phase into consideration. This is the weak point of this approach. We have

$$\|A^2 - fl(A^2)\| \le \gamma_n \|A\|^2, \quad \gamma_n = \frac{nu}{1-nu}$$

but since possibly $\|A^2\| \ll \|A\|^2$ this is not satisfactory and shows the danger in matrix squaring. If a higher degree Padé approximation is chosen then the number of squarings can be reduced. Choices suggested in the literature (N. J. Higham [25]) are $m = 8$, with $2^{-s}\|A\| < 1.5$ and $m = 13$, with $2^{-s}\|A\| < 5.4$.

Given a square matrix $A \in \mathbf{C}^{n\times n}$ a matrix X such that

$$X^2 = A, \tag{9.2.24}$$

is called a square root of A and denoted by $X = A^{1/2}$. Unlike a square root of a scalar, the square root of a matrix may not exist. For example, it is easy to verify that the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

has no square root. A sufficient condition for A to have a square root is that it has at least $n-1$ nonzero eigenvalues. We assume in the following that this condition is satisfied. If A is nonsingular and has s distinct eigenvalues then it has precisely 2^s square roots that are expressible as polynomials in the matrix A. If some eigenvalues appear in more than one Jordan block then there are infinitely many additional square roots, none of which can be expressed as a polynomial in A. For example, any Householder matrix is a square root of the identity matrix.

There is a **principal square root** of particular interest, namely the one whose eigenvalues lie in the right half plane. To make this uniquely defined we map any eigenvalue on the negative real axis to the positive imaginary axis. The principal square root, when it exists, is a polynomial in the original matrix. When A is symmetric positive definite the principal square root is the unique symmetric positive definite square root.

To compute the principal square root we first determine the Schur decomposition

$$A = QSQ^H,$$

where Q is unitary and S upper triangular. If U is an upper triangular square root of S, then $X = QUQ^H$ is a square root of A. If A is a normal matrix then $S = \operatorname{diag}(\lambda_i)$ and we can just take $U = \operatorname{diag}(\lambda_i^{1/2})$. Otherwise, from the relation $S = U^2$ we get

$$s_{ij} = \sum_{k=i}^{j} u_{ik}u_{kj}, \quad i \le j. \tag{9.2.25}$$

This gives a recurrence relation for determining the elements in U. For the diagonal elements in U we have

$$u_{ii} = s_{ii}^{1/2}, \quad i = 1:n. \tag{9.2.26}$$

Further

$$u_{ij} = \left(s_{ij} - \sum_{k=i+1}^{j-1} u_{ik} u_{kj} \right) \Big/ (u_{ii} + u_{jj}). \quad i < j. \tag{9.2.27}$$

Hence, the elements in U can be determined computed from (9.2.27), for example, one diagonal at a time. Since whenever $s_{ii} = s_{jj}$ we take $u_{ii} = u_{jj}$ this recursion does not break down. (Recall we assumed that at most one diagonal element of S is zero.)

If we let $\bar{U}$ be the computed square root of S then it can be shown that

$$\bar{U}^2 = S + E, \qquad \|E\| \le c(n)u(\|S\| + \|U\|^2),$$

where u is the unit roundoff and $c(n)$ a small constant epending on n. If we define

$$\alpha = \|A^{1/2}\|^2 / \|A\|,$$

then we have

$$\|E\| \le c(n)u(1+\alpha)\|S\|.$$

To study the conditioning of the square root we let $\tilde{X}$ be an approximation to the square root of A and look for a correction E such that $X = \tilde{X} + E$. Expanding $(\tilde{X} + E)^2 = A$ and neglecting the term E^2 we get

$$\tilde{X}E + E\tilde{X} = A - \tilde{X}^2.$$

We remark that for real matrices an analogue algorithm can be developed, which uses the real Schur decomposition and only exploys real arithmetic.

9.2.5 Non-Negative Matrices

Non-negative matrices arise in many applications and play an important role in, e.g., queuing theory, stochastic processes, and input-output analysis.

Definition 9.2.16. *A matrix $A \in \mathbf{R}^{n\times n}$ is called non-negative if $a_{ij} \ge 0$ for each i and j and* **positive** *if $a_{ij} > 0$ for $i, j = 1 : n$. Similarly, a vector $x \in \mathbf{R}^n$ is called non-negative if $x_i \ge 0$ $i = 1 : n$ and* **positive** *if $x_i > 0$ $i = 1 : n$.*

Theorem 9.2.17. *Let $A \in \mathbf{R}^{n\times n}$ be a square nonnegative matrix and let $s = Ae$, $e = (1\ 1\ \cdots\ 1)^T$ be the vector of row sums of A. Then*

$$\min_i s_i \le \rho(A) \le \max_i s_i = \|A\|_1. \tag{9.2.28}$$

For the class of nonnegative and irreducible matrices (see (Def. 9.1.5)) the following classical theorem holds.

Theorem 9.2.18. (Perron–Frobenius Theorem)

If $A > 0$ then $r = \rho(A)$ is a simple eigenvalue and there are no other eigenvalue of modulus $\rho(A)$.

If $A \geq 0$ is irreducible then $\rho(A)$ is a simple eigenvalue and

(i) *A has a positive eigenvector x corresponding to the eigenvalue $\rho(A)$ and any nonnegative eigenvector of A is a multiple of x;*

(ii) *The eigenvalues of modulus $\rho(A)$ are all simple. If there are m eigenvalues of modulus ρ, they must be of the form*

$$\lambda_k = \rho e^{\frac{2k\pi i}{m}}, \quad k = 0 : m-1.$$

(iii) *$\rho(A)$ increases when any entry of A increases.*

Proof. See, e.g., Gantmacher [15, 1959], Vol. II or [4, pp. 27,32]. A simpler proof of some of these results is found in Strang [46, 1988, [*p.* 271]. □

Perron[4] (1907) proved the first part of this theorem for $A > 0$. Later Frobenius (1912) extended most of Perron's result to the class of nonnegative irreducible matrices.

9.2.6 Finite Markov Chains

A **Markov chain**[5] is a probabilistic process in which the future development is completely determined by the present state and not at all in the way it arose. Markov chains serve as models for describing systems that can be in a number of different states $s_1, s_2, s_3, \ldots$. At each time step the system moves from state s_i to state s_j with probability q_{ij}. Such processes have many applications in the physical, biological and social sciences. The Markov chain is finite if the number of states is finite.

Definition 9.2.19. *A matrix $Q \in \mathbf{R}^{n\times n}$ is called* **row stochastic matrix** *if it satisfies*

$$q_{ij} \geq 0, \qquad \sum_{1\leq j\leq n} q_{ij} = 1, \quad i,j = 1:n. \tag{9.2.29}$$

It is called **doubly stochastic** *if in addition*

$$\sum_{1\leq i\leq n} q_{ij} = 1, \tag{9.2.30}$$

[4] German mathematician (1880–1975).

[5] Named after Russian mathematician Andrej Andreevic Markov (1856–1922), who introduced them in 1908,

In a finite Markov chain there are a finite number of states s_i, $i = 1 : n$. The nonnegative matrix Q with elements equal to the transition probabilities q_{ij} is a row stochastic matrix. From (9.2.29) it follows that

$$Qe = e, \quad e = (\,1 \quad 1 \quad \dots \quad 1\,)^T, \tag{9.2.31}$$

i.e. e is a right eigenvector of Q corresponding to the eigenvalue $\lambda = 1$. From Theorem 9.2.17 it follows that $\rho(Q) = 1$.

The vector $p = (\,p_1 \quad p_2 \quad \dots \quad p_n\,)^T$, where $p_i \geq 0$, $e^T p = 1$ is the probability that the system is at state i, is called the **state vector** of the Markov chain. Let p^k denote the state vector at time step k. Then $p^{(k+1)} = Q^T p^k$, and

$$p^k = (Q^k)^T p^0, \quad k = 1, 2, \dots.$$

An important problem is to find the **stationary distribution** p of a Markov chain. A state vector p of a Markov chain is said to be **stationary** if

$$Q^T p = p, \quad e^T p = 1. \tag{9.2.32}$$

Hence p is a *left* eigenvector of Q corresponding to the eigenvalue $\lambda = 1 = \rho(Q)$. It follows that p solves the singular homogeneous linear system

$$(I - Q^T)p = 0. \tag{9.2.33}$$

From the Perron–Frobenius Theorem it follows that if Q is irreducible then $\lambda = 1$ is a simple eigenvalue of Q and there is a unique eigenvector p satisfying (9.2.32). If $Q > 0$, then there is no other eigenvalue with modulus $\rho(Q)$ and we have the following result:

Theorem 9.2.20. *Assume that a Markov chain has a positive transition matrix. Then, independent of the initial state vector,*

$$\lim_{k\to\infty} p^k = p,$$

where p satisfies (9.2.32).

If Q is not positive then the Markov chain may not converge to a stationary state.

Example 9.2.5. Consider a Markov chain with two states for which state 2 is always transformed into state 1 and state 2 into state 1. The corresponding transition matrix

$$Q = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

with two eigenvalues of modolus $\rho(Q)$, $\lambda_1 = 1$ and $\lambda_2 = -1$. Here Q is symmetric and its left eigenvalue equals $p = (0.5\ 0.5)^T$. However, for any initial state different from p the state will oscillate and not converge.

This example can be generalized by considering a Markov chain with m states and taking Q equal to the permutation matrix corresponding to a cyclic shift. Then Q has m eigenvalues on the unit circle in the complex plane.

The theory of Markov chains for general reducible nonnegative transition matrices Q is much more complicated. It is then neccessary to classify the states. We say that a state s_i has access to a state s_j if it is possible to move from state s_i to s_j in a finite number of steps. If also s_j has access to s_i s_i and s_j are said to communicate. This is an equivalence relation on the set of states and partitions the states into classes. If a class of states has access to no other class it is called **final**. If a final class contains a single state then the state is called **absorbing**.

Suppose that Q has been permuted to its block triangular form

$$Q = \begin{pmatrix} Q_{11} & 0 & \dots & 0 \\ Q_{21} & Q_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ Q_{s1} & Q_{s2} & \dots & Q_{ss} \end{pmatrix} \tag{9.2.34}$$

where the diagonal blocks Q_{ii} are square and irreducible. Then these blocks correspond to the classes associated with the corresponding Markov chain. The class associated with Q_{ii} is final if and only if $Q_{ij} = 0$, $j = 1 : i-1$. If the matrix Q is irreducible then the corresponding matrix chain contains a single class of states.

Example 9.2.6. Suppose there is an epidemic in which every month 10% of those who are well become sick and of those who are sick 20% dies, and the rest become well. This can be modeled by the Markov process with three states dead, sick,well, and transition matrix

$$Q = \begin{pmatrix} 1 & 0 & 0 \\ 0.1 & 0 & 0.9 \\ 0 & 0.2 & 0.8 \end{pmatrix}.$$

Then the left eigenvector is $p = e_1 = (\,1 \quad 0 \quad 0\,)^T$, i.e. in the stationary distribution all are dead. Clearly the class dead is absorbing!

We now describe a way to force a Markov chain to become irreducible.

Example 9.2.7 (Eldén).

Let $Q \in \mathbf{R}^{n\times n}$ be a row stohastic matrix and set

$$P = \alpha Q + (1-\alpha)\frac{1}{n} e e^T, \quad \alpha > 0,$$

where e is a vector of all ones. Then $P > 0$ and since $e^T e = n$ we have $Pe = (1-\alpha)e + \alpha e = 1$, so P is row stochastic. From the Perron–Frobenius Theorem it follows that there is no other eigenvalue of P with modulus 1

We now show that if the eigenvalues of Q equal $1, \lambda_2, \lambda_3, \dots, \lambda_n$ then the eigenvalues of P are $1, \alpha\lambda_2, \alpha\lambda_3, \dots, \alpha\lambda_n$.

Proceeding as in the proof of the Schur normal form (Theorem 9.2.1) we define the orthogonal matrix $U = (u_1\ U_2)$, where $u_1 = e/\sqrt{n}$. Then

$$U^TQU = U^T\,(\,Q^Tu_1 \quad Q^TU_2\,) = U^T\,(\,u_1 \quad Q^TU_2\,)$$
$$= \begin{pmatrix} u_1^Tu_1 & u_1^TQ^TU_2 \\ U_2^Tu_1 & U_2^TQ^TU_2 \end{pmatrix} = \begin{pmatrix} 1 & v^T \\ 0 & T \end{pmatrix}.$$

This is a similarity transformation so T has eigenvalues $\lambda_2, \lambda_3, \ldots, \lambda_n$. Further $U^Te = \sqrt{n}e_1$ so that $U^Tee^TU = ne_1e_1^T$, and we obtain

$$U^TPU = U^T\left(\alpha Q + (1-\alpha)\frac{1}{n}ee^T\right)U$$
$$= \alpha\begin{pmatrix} 1 & v^T \\ 0 & T \end{pmatrix} + (1-\alpha)\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & \alpha v^T \\ 0 & \alpha T \end{pmatrix}.$$

The result now follows.

Review Questions

1. What is the Schur normal form of a matrix $A \in \mathbf{C}^{n\times n}$?
(b)What is meant by a normal matrix? How does the Schur form simplify for a normal matrix?

2. How can the class of matrices which are diagonalizable by unitary transformations be characterized?

3. What is meant by a defective eigenvalue? Give a simple example of a matrix with a defective eigenvalue.

4. Define the matrix function e^A. Show how this can be used to express the solution to the initial value problem $y'(t) = Ay(t)$, $y(0) = c$?

5. What can be said about the behavior of $\|A^k\|$, $k \gg 1$, in terms of the spectral radius and the order of the Jordan blocks of A? (See Problem 8.)

6. (a) Given a square matrix A. Under what condition does there exist a vector norm, such that the corresponding operator norm $\|A\|$ equals the spectral radius? If A is diagonalizable, mention a norm that has this property.
(b) What can you say about norms that come close to the spectral radius, when the above condition is not satisfied? What sets the limit to their usefulness?

7. Show that

$$\lim_{t\to\infty}\frac{1}{t}\ln\|e^{At}\| = \max_{\lambda\in\lambda(A)}\mathrm{Re}(\lambda), \qquad \lim_{t\to 0}\frac{1}{t}\ln\|e^{At}\| = \mu(A).$$

8. Prove the Cayley-Hamilton theorem for a diagonalizable matrix. Then generalize to an arbitrary matrix, either as in the text or by using Bellman's approximation theorem, (Theorem 9.2.5).

9. Give an example of a matrix, for which the minimal polynomial has a lower degree than the characteristic polynomial. Is the characteristic polynomial always divisible by the minimal polynomial?

10. Under what conditions can identities which hold for analytic functions of complex variable(s) be generalized to analytic functions of matrices?

11. (a) Show that any permutation matrix is doubly stochastic.

(b) What are the eigenvalues of matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ = & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}?$$

12. Suppose that P and Q are row stochastic matrices.

(a) Show that $\alpha P + (1 - \alpha Q)$ is a row stochastic matrix.

(b) Show that PQ is a row stochastic matrix.

Problems and Computer Exercises

1. Find a similarity transformation $X^{-1}AX$ that diagonalizes the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1+\epsilon \end{pmatrix}, \quad \epsilon > 0.$$

How does the transformation X behave as ϵ tends to zero?

2. Show that the Sylvester equation (9.2.6) can be written as the linear system

$$(I_m \otimes A - B^T \otimes I_n)\text{vec}(X) = \text{vec}(C), \tag{9.2.35}$$

where $\otimes$ denotes the Kronecker product and $\text{vec}(X)$ is the column vector obtained by stacking the column of X on top of each other.

3. (a) Let $A \in \mathbf{R}^{n\times n}$, and consider the matrix polynomial

$$p(A) = a_0A^n + a_1A^{n-1} + \cdots + a_nI \in \mathbf{R}^{n\times n}.$$

Show that if $Ax = \lambda x$ then $p(\lambda)$ is an eigenvalue and x an associated eigenvector of $p(A)$.

(b) Show that the same is true in general for an analytic function $f(A)$. Verify (9.2.18). Also construct an example, where $p(A)$ has other eigenvectors in addition to those of A.

4. Show that the series expansion

$$(I - A)^{-1} = I + A + A^2 + A^3 + \ldots$$

converges if $\rho(A) < 1$.

5. (a) Let $\|\cdot\|$ be a consistent matrix norm, and ρ denote the spectral radius. Show that

$$\lim_{k\to\infty} \|A^k\|^{1/k} = \rho(A).$$

(b) Show that

$$\lim_{t\to\infty} \frac{\ln\|e^{At}\|}{t} = \max_{\lambda\in\lambda(A)} \Re(\lambda).$$

Hint: Assume, without loss of generality, that A is in its Jordan canonical form.

6. Show that the eigenvalues λ_i of a matrix A satisfy the inequalities

$$\sigma_{\min}(A) \le \min_i |\lambda_i| \le \max_i |\lambda_i| \sigma_{\max}(A).$$

Hint: Use the fact that the singular values of A and its Schur decomposition $Q^TAQ = \operatorname{diag}(\lambda_i) + N$ are the same.

7. Show that Sylvester's equation (9.2.6) can be written as an equation in standard matrix-vector form,

$$\big((I\otimes A) + (-B^T\otimes I)\big)x = c,$$

where the vectors $x, c \in \mathbf{R}^{nm}$ are obtained from $X = (x_1,\dots,x_m)$ and $C = (c_1,\dots,c_m)$ by

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}, \qquad c = \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix}.$$

Then use (9.1.19) to give an independent proof that Sylvester's equation has a unique solution if and only if $\lambda_i - \mu_j \neq 0$, $i = 1,\dots,n$, $j = 1,\dots,m$.

8. Show that

$$e^A \otimes e^B = e^{B\oplus A},$$

where $\oplus$ denotes the Kronecker sum.

9. (a) Show that if $A = \begin{pmatrix} \lambda_1 & 1 \\ 0 & \lambda_2 \end{pmatrix}$ and $\lambda_1 \neq \lambda_2$ then

$$f(A) = \begin{pmatrix} f(\lambda_1) & \dfrac{f(\lambda_1) - f(\lambda_2)}{\lambda_1 - \lambda_2} \\ 0 & f(\lambda_2) \end{pmatrix}.$$

Comment on the numerical use of this expression when $\lambda_2 \to \lambda_1$.

(b) For $A = \begin{pmatrix} 0.5 & 1 \\ 0 & 0.6 \end{pmatrix}$, show that $\ln(A) = \begin{pmatrix} -0.6931 & 1.8232 \\ 0 & 0.5108 \end{pmatrix}$.

10. (a) Compute e^A, where

$$A = \begin{pmatrix} -49 & 24 \\ -64 & 31 \end{pmatrix},$$

using the method of scaling and squaring. Scale the matrix so that $\|A/2^s\|_\infty < 1/2$ and approximate the exponential of the scaled matrix by a Padé approximation of order (4,4).

(b) Compute the eigendecomposition $A = X\Lambda X^{-1}$ and obtain $e^A = Xe^\Lambda X^{-1}$. Compare the result with that obtained in (a).

11. Show that an analytic function of the matrix A can be computed by Newton's interpolation formula, i.e.,

$$f(A) = f(\lambda_1)I + \sum_{j=1}^{n^*} f(\lambda_1, \lambda_2, \ldots, \lambda_j)(A - \lambda_1 I)\cdots(A - \lambda_j I)$$

where λ_j, $j = 1, 2, \ldots, n^*$ are the distinct eigenvalues of A, each counted with the same multiplicity as in the minimal polynomial. Thus, n^* is the degree of the minimal polynomial of A.

12. We use the notation of Theorem 9.2.13. For a given n, show by an appropriate choice of ϵ that $\|A^n\|_p \leq Cn^{m^*-1}\rho^n$, where C is independent of n. Then derive the same result from the Jordan Canonical form.

Hint: See the comment after Theorem 9.2.13.

13. Let C be a closed curve in the complex plane, and consider the function,

$$\phi_C(A) = \frac{1}{2\pi i}\int_C (zI - A)^{-1}dz,$$

If the whole spectrum of A is inside C then, by Example 9.2.2, $\phi_C(A) = I$. What is $\phi_C(A)$, when only part of the spectrum (or none of it) is inside C? Is it generally true that $\phi_C(A)^2 = \phi_C(A)$?

Hint: First consider the case, when A is a Jordan block.

9.3 Perturbation Theory and Eigenvalue Bounds

Methods for computing eigenvalues and eigenvectors are subject to roundoff errors. The best we can demand of an algorithm in general is that it yields approximate eigenvalues of a matrix A that are the exact eigenvalues of a slightly perturbed matrix $A + E$. In order to estimate the error in the computed result we need to know the effects of the perturbation E on the eigenvalues and eigenvectors of A. Such results are derived in this section.

9.3.1 Gerschgorin's Theorems

In 1931 the Russian mathematician published a seminal paper [17] on how to obtain estimates of all eigenvalues of a complex matrix. His results can be used both to locate eigenvalues and to derive perturbation results.

Theorem 9.3.1.

All the eigenvalues of the matrix $A \in \mathbf{C}^{n\times n}$ lie in the union of the **Gerschgorin disks** *in the complex plane*

$$\mathcal{D}_i = \{z \mid |z - a_{ii}| \le r_i\}, \qquad r_i = \sum_{j=1, j\neq i}^{n} |a_{ij}|, \qquad i = 1, 2, \ldots, n. \tag{9.3.1}$$

Proof. If λ is an eigenvalue there is an eigenvector $x \neq 0$ such that $Ax = \lambda x$, or

$$(\lambda - a_{ii})x_i = \sum_{j=1, j\neq i}^{n} a_{ij}x_j, \quad i = 1, \ldots, n.$$

Choose i so that $|x_i| = \|x\|_\infty$. Then

$$|\lambda - a_{ii}| \le \sum_{j=1, j\neq i}^{n} \frac{|a_{ij}||x_j|}{|x_i|} \le r_i. \tag{9.3.2}$$

□

The Gerschgorin theorem is very useful for getting crude estimates for eigenvalues of matrices, and can also be used to get accurate estimates for the eigenvalues of a nearly diagonal matrix. Since A and A^T have the same eigenvalues we can, in the non-Hermitian case, obtain more information about the location of the eigenvalues simply by applying the theorem also to A^T .

From (9.3.2) it follows that if the ith component of the eigenvector is maximal, then λ lies in the ith disk. Otherwise the Gerschgorin theorem does not say in *which* disks the eigenvalues lie. Sometimes it is possible to decide this as the following theorem shows.

Theorem 9.3.2.

If the union $\mathcal{M}$ of k Gerschgorin disks $\mathcal{D}_i$ is disjoint from the remaining disks, then $\mathcal{M}$ contains precisely k eigenvalues of A.

Proof. Consider for $t \in [0, 1]$ the family of matrices

$$A(t) = tA + (1 - t)D_A, \quad D_A = \operatorname{diag}(a_{ii}).$$

The coefficients in the characteristic polynomial are continuous functions of t, and hence also the eigenvalues $\lambda(t)$ of $A(t)$ are continuous functions of t. Since $A(0) = D_A$ and $A(1) = A$ we have $\lambda_i(0) = a_{ii}$ and $\lambda_i(1) = \lambda_i$. For $t = 0$ there are exactly k eigenvalues in $\mathcal{M}$. For reasons of continuity an eigenvalue $\lambda_i(t)$ cannot jump to a subset that does not have a continuous connection with a_{ii} for $t = 1$. Therefore also k eigenvalues of $A = A(1)$ lie in $\mathcal{M}$. □

Example 9.3.1.
The matrix

$$A = \begin{pmatrix} 2 & -0.1 & 0.05 \\ 0.1 & 1 & -0.2 \\ 0.05 & -0.1 & 1 \end{pmatrix},$$

with eigenvalues $\lambda_1 = 0.8634$, $\lambda_2 = 1.1438$, $\lambda_3 = 1.9928$, has the Gerschgorin disks

$$\mathcal{D}_1 = \{z \mid |z-2| \le 0.15\}; \quad \mathcal{D}_2 = \{z \mid |z-1| \le 0.3\}; \quad \mathcal{D}_3 = \{z \mid |z-1| \le 0.15\}.$$

Since the disk $\mathcal{D}_1$ is disjoint from the rest of the disks, it must contain precisely one eigenvalue of A. The remaining two eigenvalues must lie in $\mathcal{D}_2 \cup \mathcal{D}_3 = \mathcal{D}_2$.

There is another useful sharpening of Gerschgorin's Theorem in case the matrix A is irreducible, cf. Def. 9.1.5.

Theorem 9.3.3.
If A is irreducible then each eigenvalue λ lies in the **interior** *of the union of the Gerschgorin disks, unless it lies on the boundary of* **all** *Gerschgorin disks.*

Proof. If λ lies on the boundary of the union of the Gerschgorin disks, then we have

$$|\lambda - a_{ii}| \ge r_i, \quad \forall i. \tag{9.3.3}$$

Let x be a corresponding eigenvector and assume that $|x_{i_1}| = \|x\|_\infty$. Then from the proof of Theorem 9.3.1 and (9.3.3) it follows that $|\lambda - a_{i_1 i_1}| = r_{i_1}$. But (9.3.2) implies that equality can only hold here if for any $a_{i_1 j} \neq 0$ it holds that $|x_j| = \|x\|_\infty$. If we assume that $a_{i_1,i_2} \neq 0$ then it follows that $|\lambda - a_{i_2 i_2}| = r_{i_2}$. But since A is irreducible for any $j \neq i$ there is a path $i = i_1, i_2, \ldots, i_p = j$. It follows that λ must lie on the boundary of all Gerschgorin disks. ∎

Example 9.3.2. Consider the real, symmetric matrix

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \in \mathbf{R}^{n \times n}.$$

Its Gerschgorin disks are

$$|z-2| \le 2, \quad i = 2, \ldots, n-1, \qquad |z-2| \le 1, \quad i = 1, n,$$

and it follows that all eigenvalues of A satisfy $\lambda \ge 0$. Since zero is on the boundary of the union of these disks, but *not* on the boundary of all disks, zero cannot be an eigenvalue of A. Hence all eigenvalues are *strictly* positive and A is positive definite.

9.3.2 Perturbation Theorems

In the rest of this section we consider the sensitivity of eigenvalue and eigenvectors to perturbations.

Theorem 9.3.4. (Bauer–Fike.)
Let the matrix $A \in \mathbf{C}^{n\times n}$ be diagonalizable, $X^{-1}AX = D = diag(\lambda_1, \cdots, \lambda_n)$, and let μ be an eigenvalue to $A + E$. Then for any p-norm

$$\min_{1\le i\le n} |\mu - \lambda_i| \le \kappa_p(X)\|E\|_p. \tag{9.3.4}$$

where $\kappa_p(X) = \|X^{-1}\|_p \|X\|_p$ is the condition number of the eigenvector matrix.

Proof. We can assume that μ is not an eigenvalue of A, since otherwise (9.3.4) holds trivially. Since μ is an eigenvalue of $A + E$ the matrix $A + E - \mu I$ is singular and so is also

$$X^{-1}(A + E - \mu I)X = (D - \mu I) + X^{-1}EX.$$

Then there is a vector $z \neq 0$ such that

$$(D - \mu I)z = -X^{-1}EXz.$$

Solving for z and taking norms we obtain

$$\|z\|_p \le \kappa_p(X)\|(D - \mu I)^{-1}\|_p\|E\|_p\|z\|_p.$$

The theorem follows by dividing by $\|z\|_p$ and using the fact that for any p-norm $\|(D - \mu I)^{-1}\|_p = 1/\min_{1\le i\le n} |\lambda_i - \mu|$. ∎

The Bauer–Fike theorem shows that $\kappa_p(X)$ is an upper bound for the condition number of the eigenvalues of a *diagonalizable matrix A*. In particular if A is normal we know from the Schur Canonical Form (Theorem 9.2.1) that we can take $X = U$ to be a unitary matrix. Then we have $\kappa_2(X) = 1$, which shows the important result that *the eigenvalues of a normal matrix are perfectly conditioned, also if they have multiplicity greater than one.* On the other hand, for a matrix A which is close to a defective matrix the eigenvalues can be very ill-conditioned, see Example 9.2.1, and the following example.

Example 9.3.3.

Consider the matrix $A = \begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix}$, $0 < \epsilon$ with eigenvector matrix

$$X = \begin{pmatrix} 1 & 1 \\ \sqrt{\epsilon} & -\sqrt{\epsilon} \end{pmatrix}, \qquad X^{-1} = \frac{0.5}{\sqrt{\epsilon}} \begin{pmatrix} \sqrt{\epsilon} & 1 \\ \sqrt{\epsilon} & -1 \end{pmatrix}.$$

If $\epsilon \ll 1$ then

$$\kappa_\infty(X) = \|X^{-1}\|_\infty\|X\|_\infty = \frac{1}{\sqrt{\epsilon}} + 1 \gg 1.$$

Note that in the limit when $\epsilon \to 0$ the matrix A is not diagonalizable.

In general a matrix may have a mixture of well-conditioned and ill-conditioned eigenvalues. Therefore it is useful to have perturbation estimates for the individual eigenvalues of a matrix A. We now derive first order estimates for simple eigenvalues and corresponding eigenvectors.

Theorem 9.3.5.

Let λ_j be a simple eigenvalue of A and let x_j and y_j be the corresponding right and left eigenvector of A,

$$Ax_j = \lambda_j x_j, \qquad y_j^H A = \lambda_j y_j^H.$$

Then for sufficiently small ϵ the matrix $A + \epsilon E$ has a simple eigenvalue $\lambda_j(\epsilon)$ such that,

$$\lambda_j(\epsilon) = \lambda_j + \epsilon \frac{y_j^H E x_j}{y_j^H x_j} + O(\epsilon^2). \tag{9.3.5}$$

Proof. Since λ_j is a simple eigenvalue there is a $\delta > 0$ such that the disk $\mathcal{D} = \{\mu \| |\mu - \lambda_j| < \delta\}$ does not contain any eigenvalues of A other than λ_j. Then using Theorem 9.3.2 it follows that for sufficiently small values of ϵ the matrix $A + \epsilon E$ has a simple eigenvalue $\lambda_j(\epsilon)$ in $\mathcal{D}$. If we denote a corresponding eigenvector $x_j(\epsilon)$ then

$$(A + \epsilon E)x_j(\epsilon) = \lambda_j(\epsilon)x_j(\epsilon).$$

Using results from function theory, it can be shown that $\lambda_j(\epsilon)$ and $x_j(\epsilon)$ are analytic functions of ϵ for $\epsilon < \epsilon_0$. Differentiating with respect to ϵ and putting $\epsilon = 0$ we get

$$(A - \lambda_j I)x_j'(0) + Ex_j = \lambda_j'(0)x_j. \tag{9.3.6}$$

Since $y_j^H(A - \lambda_j I) = 0$ we can eliminate $x_j'(0)$ by multiplying this equation with y_j^H and solve for $\lambda_j'(0) = y_j^H E x_j / y_j^H x_j$. ∎

If $\|E\|_2 = 1$ we have $|y_j^H E x_j| \le \|x_j\|_2 \|y_j\|_2$ and E can always be chosen so that equality holds. If we also normalize so that $\|x_j\|_2 = \|y_j\|_2 = 1$, then $1/s(\lambda_j)$, where

$$s(\lambda_j) = |y_j^H x_j| \tag{9.3.7}$$

can be taken as *the condition number of the simple eigenvalue* λ_j. Note that $s(\lambda_j) = \cos\theta(x_j, y_j)$, where $\theta(x_j, y_j)$ is the acute angle between the left and right eigenvector corresponding to λ_j. If A is a normal matrix we get $s(\lambda_j) = 1$.

The above theorem shows that for perturbations in A of order ϵ, a simple eigenvalue λ of A will be perturbed by an amount approximately equal to $\epsilon/s(\lambda)$. If λ is a defective eigenvalue, then there is no similar result. *Indeed, if the largest Jordan block corresponding to λ is of order k, then perturbations to λ of order $\epsilon^{1/k}$ can be expected.* Note that for a Jordan box we have $x = e_1$ and $y = e_m$ and so $s(\lambda) = 0$ in (9.3.7).

Example 9.3.4.

Consider the perturbed diagonal matrix

$$A + \epsilon E = \begin{pmatrix} 1 & \epsilon & 2\epsilon \\ \epsilon & 2 & \epsilon \\ \epsilon & 2\epsilon & 2 \end{pmatrix}.$$

Here A is diagonal with left and right eigenvector equal to $x_i = y_i = e_i$. Thus $y_i^H E x_i = e_{ii} = 0$ and the first order term in the perturbation of the simple eigenvalues are zero. For $\epsilon = 10^{-3}$ the eigenvalues of $A + E$ are

$$0.999997, \quad 1.998586, \quad 2.001417.$$

Hence the perturbation in the simple eigenvalue λ_1 is of order 10^{-6}. Note that the Bauer–Fike theorem would predict perturbations of order 10^{-3} for all three eigenvalues.

We now consider the perturbation of an eigenvector x_j corresponding to a simple eigenvalue λ_j. Assume that the matrix A is diagonalizable and that $x_1, \ldots, x_n$ are linearly independent eigenvectors. Then we can write

$$x_j(\epsilon) = x_j + \epsilon x_j'(0) + O(\epsilon^2), \qquad x_j'(0) = \sum_{k \neq j} c_{kj} x_k,$$

where we have normalized $x_j(\epsilon)$ to have unit component along x_j. Substituting the expansion of $x_j'(0)$ into (9.3.6) we get

$$\sum_{k \neq j} c_{kj}(\lambda_k - \lambda_j) x_k + E x_j = \lambda_j'(0) x_j.$$

Multiplying by y_i^H and using $y_i^H x_j = 0$, $i \neq j$, we obtain

$$c_{ij} = \frac{y_i^H E x_j}{(\lambda_j - \lambda_i) y_i^H x_i}, \quad i \neq j. \tag{9.3.8}$$

Hence, the sensitivity of the eigenvectors also depend on the separation $\delta_j = \min_{i \neq j} |\lambda_i - \lambda_j|$ between λ_j and the rest of the eigenvalues of A. If several eigenvectors corresponds to a multiple eigenvalue these are not uniquely determined, which is consistent with this result. Note that even if the individual eigenvectors are sensitive to perturbations it may be that an invariant subspace containing these eigenvectors is well determined.

To measure the accuracy of computed invariant subspaces we need to introduce the largest angle between two subspaces.

Definition 9.3.6. *Let $\mathcal{X}$ and $\mathcal{Y} = \mathcal{R}(Y)$ be two subspaces of $\mathbf{C}^n$ of dimension k. Define the largest angle between these subspaces to be*

$$\theta_{\max}(\mathcal{X}, \mathcal{Y}) = \max_{\substack{x \in \mathcal{X} \\ \|x\|_2 = 1}} \min_{\substack{y \in \mathcal{Y} \\ \|y\|_2 = 1}} \theta(x, y). \tag{9.3.9}$$

where $\theta(x, y)$ is the acute angle between x and y.

The quantity $\sin\theta_{\max}(\mathcal{X}, \mathcal{Y})$ defines a distance between the two subspaces $\mathcal{X}$ and $\mathcal{Y}$. If X and Y are orthonormal matrices such that $\mathcal{X} = \mathcal{R}(X)$ and $\mathcal{Y} = \mathcal{R}(Y)$, then it can be shown (see Golub and Van Loan [21]) that

$$\theta(\mathcal{X}, \mathcal{Y}) = \arccos \sigma_{min}(X^H Y). \tag{9.3.10}$$

9.3.3 Hermitian Matrices

We have seen that the eigenvalues of Hermitian, and real symmetric matrices are all real, and from Theorem 9.3.5 it follows that these eigenvalues are perfectly conditioned. For this class of matrices it is possible to get more informative perturbation bounds, than those given above. In this section we give several classical theorems. They are all related to each other, and the interlace theorem dates back to Cauchy, 1829. We assume in the following that the eigenvalues of A have been ordered in decreasing order $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

In the particular case of a Hermitian matrix the extreme eigenvalues λ_1 and λ_n can be characterized by

$$\lambda_1 = \max_{\substack{x \in C^n \\ x \neq 0}} \rho(x), \quad \lambda_n = \min_{\substack{x \in C^n \\ x \neq 0}} \rho(x).$$

The following theorem gives an important extremal characterization also of the intermediate eigenvalues of a Hermitian matrix.

Theorem 9.3.7. Fischer's Theorem.

Let the Hermitian matrix A have eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ ordered so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Then

$$\lambda_i = \max_{\dim(\mathcal{S})=i} \min_{\substack{x \in \mathcal{S} \\ x \neq 0}} \frac{x^H A x}{x^H x} \tag{9.3.11}$$

$$= \min_{\dim(\mathcal{S})=n-i+1} \max_{\substack{x \in \mathcal{S} \\ x \neq 0}} \frac{x^H A x}{x^H x}. \tag{9.3.12}$$

where $\mathcal{S}$ denotes a subspace of $\mathbf{C}^n$.

Proof. See Stewart [43, 1973, p. 314]. □

The formulas (9.3.11) and (9.3.12) are called the max-min and the min-max characterization, respectively. They can be used to establish an important relation between the eigenvalues of two Hermitian matrices A and B, and their sum $C = A + B$.

Theorem 9.3.8.

Let $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$, $\beta_1 \geq \beta_2 \geq \cdots \geq \beta_n$, and $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_n$ be the eigenvalues of the Hermitian matrices A, B, and $C = A + B$. Then

$$\alpha_i + \beta_1 \geq \gamma_i \geq \alpha_i + \beta_n, \quad i = 1, 2, \ldots, n. \tag{9.3.13}$$

Proof. Let $x_1, x_2, \ldots, x_n$ be an orthonormal system of eigenvectors of A corresponding to $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_n$, and let $\mathcal{S}$ be the subspace of $\mathbf{C}^n$ spanned by $x_1, \ldots, x_i$. Then by Fischer's theorem

$$\gamma_i \geq \min_{\substack{x \in \mathcal{S} \\ x \neq 0}} \frac{x^H C x}{x^H x} \geq \min_{\substack{x \in \mathcal{S} \\ x \neq 0}} \frac{x^H A x}{x^H x} + \min_{\substack{x \in \mathcal{S} \\ x \neq 0}} \frac{x^H B x}{x^H x} = \alpha_i + \min_{\substack{x \in \mathcal{S} \\ x \neq 0}} \frac{x^H B x}{x^H x} \geq \alpha_i + \beta_n.$$

This is the last inequality of (9.3.12). The first equality follows by applying this result to $A = C + (-B)$. □

The theorem implies that when B is added to A all of its eigenvalues are changed by an amount which lies between the smallest and greatest eigenvalues of B. If the matrix rank $(B) < n$, the result can be sharpened, see Parlett [38, Section 10-3]. An important case is when $B = \pm z z^T$ is a rank one matrix. Then B has only one nonzero eigenvalue equal to $\rho = \pm \|z\|_2^2$. In this case the perturbed eigenvalues will satisfy the relations

$$\lambda_i' - \lambda_i = m_i \rho, \quad 0 \leq m_i, \quad \sum m_i = 1. \tag{9.3.14}$$

Hence all eigenvalues are shifted by an amount which lies between zero and ρ.

An important application is to get bounds for the eigenvalues λ_i' of $A + E$, where A and E are Hermitian matrices. Usually the eigenvalues of E are not known, but from

$$\max\{|\lambda_1(E)|, |\lambda_n(E)|\} = \rho(E) = \|E\|_2$$

it follows that

$$|\lambda_i - \lambda_i'| \leq \|E\|_2. \tag{9.3.15}$$

Note that this result also holds for *large perturbations*.

A related result is the **Wielandt–Hoffman theorem** which states that

$$\sqrt{\textstyle\sum_{i=1}^n |\lambda_i - \lambda_i'|^2} \leq \|E\|_F. \tag{9.3.16}$$

An elementary proof of this result is given by Wilkinson [52, Section 2.48].

Another important result that follows from Fischer's Theorem is the following theorem, due to Cauchy, which relates the eigenvalues of a principal submatrix to the eigenvalues of the original matrix.

Theorem 9.3.9. Interlacing Property.

Let A_{n-1} be a principal submatrix of order $n-1$ of a Hermitian matrix $A_n \in \mathbf{C}^{n \times n}$, Then, the eigenvalues of A_{n-1}, $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_{n-1}$ interlace the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ of A_n, that is

$$\lambda_i \geq \mu_i \geq \lambda_{i+1}, \quad i = 1, \ldots, n-1. \tag{9.3.17}$$

Proof. Without loss of generality we assume that A_{n-1} is the leading principal submatrix of A,

$$A_n = \begin{pmatrix} A_{n-1} & a^H \\ a & \alpha \end{pmatrix}.$$

Consider the subspace of vectors $\mathcal{S}' = \{x \in \mathbf{C}^n, x \perp e_n\}$. Then with $x \in \mathcal{S}'$ we have $x^H A_n x = (x')^H A_{n-1} x'$, where $x^H = ((x')^H, 0)$. Using the minimax characterization (9.3.11) of the eigenvalue λ_i it follows that

$$\lambda_i = \max_{\dim(\mathcal{S})=i} \min_{\substack{x\in\mathcal{S}\\ x\neq 0}} \frac{x^H A_n x}{x^H x} \geq \max_{\substack{dim(\mathcal{S})=i\\ \mathcal{S}\perp e_n}} \min_{\substack{x\in\mathcal{S}\\ x\neq 0}} \frac{x^H A_n x}{x^H x} = \mu_i.$$

The proof of the second inequality $\mu_i \geq \lambda_{i+1}$ is obtained by a similar argument applied to $-A_n$. □

Since any principal submatrix of a Hermitian matrix also is Hermitian, this theorem can be used recursively to get relations between the eigenvalues of A_{n-1} and A_{n-2}, A_{n-2} and A_{n-3}, etc.

9.3.4 Rayleigh quotient and residual bounds

We make the following definition.

Definition 9.3.10.
The **Rayleigh quotient** *of a nonzero vector $x \in \mathbf{C}^n$ is the (complex) scalar*

$$\rho(x) = \rho(A, x) = \frac{x^H A x}{x^H x}. \tag{9.3.18}$$

The Rayleigh quotient plays an important role in the computation of eigenvalues and eigenvectors. The Rayleigh quotient is a homogeneous function of x, $\rho(\alpha x) = \rho(x)$ for all scalar $\alpha \neq 0$.

Definition 9.3.11.
The **field of values** *of a matrix A is the set of all possible Rayleigh quotients*

$$F(A) = \{\rho(A, x) \mid x \in C^n\}.$$

For any unitary matrix U we have $F(U^H A U) = F(A)$. From the Schur canonical form it follows that there is no restriction in assuming A to be upper triangular, and, if normal, then diagonal. Hence for a normal matrix A

$$\rho(x) = \sum_{i=1}^{n} \lambda_i |\xi_i|^2 \Big/ \sum_{i=1}^{n} |\xi_i|^2,$$

that is any point in $F(A)$ is a weighted mean of the eigenvalues of A. Thus for a normal matrix the field of values coincides with the convex hull of the eigenvalues. In the special case of a Hermitian matrix the field of values equals the segment $[\lambda_1, \lambda_n]$ of the real axis.

In general the field of values of a matrix A may contain complex values even if its eigenvalues are real. However, the field of values will always contain the convex hull of the eigenvalues.

Let x and A be given and consider the problem

$$\min_{\mu} \|Ax - \mu x\|_2^2.$$

This is a linear least squares problem for the unknown μ. The normal equations are $x^H x\mu = x^H Ax$. Hence the minimum is attained for $\rho(x)$, the Rayleigh quotient of x.

When A is Hermitian the gradient of $\frac{1}{2}\rho(x)$ is

$$\frac{1}{2}\nabla\rho(x) = \frac{Ax}{x^H x} - \frac{x^H Ax}{(x^H x)^2}x = \frac{1}{x^H x}(Ax - \rho x),$$

and hence the Rayleigh quotient $\rho(x)$ is stationary if and only if x is an eigenvector of A.

Suppose we have computed by some method an approximate eigenvalue/eigenvector pair (σ, v) to a matrix A. In the following we derive some error bounds depending on the **residual vector**

$$r = Av - \sigma v.$$

Since $r = 0$ if (σ, v) are an exact eigenpair it is reasonable to assume that the size of the residual r measures the accuracy of v and σ. We show a simple backward error bound:

Theorem 9.3.12.

Let $\bar{\lambda}$ and $\bar{x}$, $\|\bar{x}\|_2 = 1$, be a given approximate eigenpair of $A \in \mathbf{C}^{n\times n}$, and $r = A\bar{x} - \bar{\lambda}\bar{x}$ be the corresponding residual vector. Then $\bar{\lambda}$ and $\bar{x}$ is an exact eigenpair of the matrix $A + E$, where

$$E = -r\bar{x}^H, \qquad \|E\|_2 = \|r\|_2. \tag{9.3.19}$$

Proof. We have $(A + E)\bar{x} = (A - r\bar{x}^H/\bar{x}^H\bar{x})\bar{x} = A\bar{x} - r = \bar{\lambda}\bar{x}$. □

It follows that given an approximate eigenvector $\bar{x}$ a good eigenvalue approximation is the Rayleigh quotient $\rho(\bar{x})$, since this choice minimizes the error bound in Theorem 9.3.12.

By combining Theorems 9.3.4 and 9.3.12 we obtain for a Hermitian matrix A the very useful a posteriori error bound

Corollary 9.3.13. *Let A be a Hermitian matrix. For any $\bar{\lambda}$ and any unit vector $\bar{x}$ there is an eigenvalue of λ of A such that*

$$|\lambda - \bar{\lambda}| \le \|r\|_2, \qquad r = A\bar{x} - \bar{\lambda}\bar{x}. \tag{9.3.20}$$

For a fixed $\bar{x}$, the error bound is minimized by taking $\bar{\lambda} = \bar{x}^T A\bar{x}$.

This shows that $(\bar{\lambda}, \bar{x})$ ($\|\bar{x}\|_2 = 1$) is a numerically acceptable eigenpair of the Hermitian matrix A if $\|A\bar{x} - \lambda\bar{x}\|_2$ is of order machine precision.

For a Hermitian matrix A, the Rayleigh quotient $\rho(x)$ may be a far more accurate approximate eigenvalue than x is an approximate eigenvector. The following theorem shows that if an eigenvector is known to precision ϵ, the Rayleigh quotient approximates the corresponding eigenvalue to precision ϵ^2.

Theorem 9.3.14.

Let the Hermitian matrix A have eigenvalues $\lambda_1, \ldots, \lambda_n$ and orthonormal eigenvectors $x_1, \ldots, x_n$. If the vector $x = \sum_{i=1}^n \xi_i x_i$, satisfies

$$\|x - \xi_1 x_1\|_2 \le \epsilon \|x\|_2. \tag{9.3.21}$$

then

$$|\rho(x) - \lambda_1| \le 2\|A\|_2 \epsilon^2. \tag{9.3.22}$$

Proof. Writing $Ax = \sum_{i=1}^n \xi_i \lambda_i x_i$, the Rayleigh quotient becomes

$$\rho(x) = \sum_{i=1}^n |\xi_i|^2 \lambda_i \Big/ \sum_{i=1}^n |\xi_i|^2 = \lambda_1 + \sum_{i=2}^n |\xi_i|^2 (\lambda_i - \lambda_1) \Big/ \sum_{i=1}^n |\xi_i|^2.$$

Using (9.3.21) we get $|\rho(x) - \lambda_1| \le \max_i |\lambda_i - \lambda_1| \epsilon^2$. Since the matrix A is Hermitian we have $|\lambda_i| \le \sigma_1(A) = \|A\|_2$, $i = 1, \ldots, n$, and the theorem follows. □

Stronger error bounds can be obtained if $\sigma = \rho(v)$ is known to be well separated from all eigenvalues except λ.

Theorem 9.3.15.

Let A be a Hermitian matrix with eigenvalues $\lambda(A) = \{\lambda_1, \ldots, \lambda_n\}$, x a unit vector and $\rho(x)$ its Rayleigh quotient. Let $Az = \lambda_\rho z$, where λ_ρ is the eigenvalue of A closest to $\rho(x)$. Define

$$\operatorname{gap}(\rho) = \min_{\lambda \in \lambda(A)} |\lambda - \rho|, \quad \lambda \ne \lambda_\rho. \tag{9.3.23}$$

Then it holds that

$$|\lambda_\rho - \rho(x)| \le \|Ax - x\rho\|_2^2 / \operatorname{gap}(\rho), \tag{9.3.24}$$

$$\sin\theta(x, z) \le \|Ax - x\rho\|_2 / \operatorname{gap}(\rho). \tag{9.3.25}$$

Proof. See Parlett [38, Section 11.7]. □

Example 9.3.5.

With $x = (1, 0)^T$ and

$$A = \begin{pmatrix} 1 & \epsilon \\ \epsilon & 0 \end{pmatrix}, \text{ we get } \rho = 1, \quad Ax - x\rho = \begin{pmatrix} 0 \\ \epsilon \end{pmatrix}.$$

From Corollary 9.3.13 we get $|\lambda-1| \leq \epsilon$, whereas Theorem 9.3.15 gives the improved bound $|\lambda - 1| \leq \epsilon^2/(1-\epsilon^2)$.

Often $\text{gap}(\sigma)$ is not known and the bounds in Theorem 9.3.15 are only theoretical. In some methods, e.g., the method of spectrum slicing (see Section 9.4.4) an interval around σ can be determined which contain no eigenvalues of A.

9.3.5 Residual bounds for SVD

The singular values of a matrix $A \in \mathbf{R}^{m\times n}$ equal the positive square roots of the eigenvalues of the symmetric matrix A^TA and AA^T. Another very useful relationship between the SVD of $A = U\Sigma V^T$ and a symmetric eigenvalue was given in Theorem 7.3.2. If A is square, then[6]

$$C = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} U & U \\ V & -V \end{pmatrix} \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} U & U \\ V & -V \end{pmatrix}^T \tag{9.3.26}$$

Using these relationships the theory developed for the symmetric (Hermitian) eigenvalue problem in Secs. 9.3.3–9.3.4 applies also to the singular value decomposition. For example, Theorems 8.3.3–8.3.5 are straightforward applications of Theorems 9.3.7–9.3.9.

We now consider applications of the Rayleigh quotient and residual error bounds given in Section 9.3.4. If u, v are unit vectors the Rayleigh quotient of C is

$$\rho(u,v) = \frac{1}{\sqrt{2}}(u^T, v^T) \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} u \\ v \end{pmatrix} = u^TAv, \tag{9.3.27}$$

From Corollary 9.3.13 we obtain the following error bound.

Theorem 9.3.16. *For any scalar α and unit vectors u, v there is a singular value σ of A such that*

$$|\sigma - \alpha| \leq \frac{1}{\sqrt{2}} \left\| \begin{pmatrix} Av - u\alpha \\ A^Tu - v\alpha \end{pmatrix} \right\|_2. \tag{9.3.28}$$

For fixed u, v this error bound is minimized by taking $\alpha = u^TAv$.

The following theorem is an application to Theorem 9.3.15.

Theorem 9.3.17.

Let A have singular values σ_i, $i = 1, \ldots, n$. Let u and v be unit vectors, $\rho = u^TAv$ the corresponding Rayleigh quotient, and

$$\delta = \frac{1}{\sqrt{2}} \left\| \begin{pmatrix} Av - u\rho \\ A^Tu - v\rho \end{pmatrix} \right\|_2$$

[6] This assumption is no restriction since we can always adjoin zero rows (columns) to make A square.

the residual norm. If σ_s is the closest singular value to ρ and $Au_s = \sigma_s v_s$, then

$$|\sigma_s - \rho(x)| \le \delta^2/\text{gap}\,(\rho), \tag{9.3.29}$$

$$\max\{\sin\theta(u_s, u), \sin\theta(v_s, v)\} \le \delta/\text{gap}\,(\rho). \tag{9.3.30}$$

where

$$\text{gap}\,(\rho) = \min_{i \ne s} |\sigma_i - \rho|. \tag{9.3.31}$$

Review Questions

1. State Gerschgorin's Theorem, and discuss how it can be sharpened.
2. Discuss the sensitivity to perturbations of eigenvalues and eigenvectors of a Hermitian matrix A.
3. Suppose that $(\bar{\lambda}, \bar{x})$ is an approximate eigenpair of A. Give a backward error bound. What can you say of the error in $\bar{\lambda}$ if A is Hermitian?
4. (a) Tell the minimax and maximin properties of the eigenvalues (of what kind of matrices?), and the related properties of the singular values (of what kind of matrices?).

   (b) Show how the theorems in (a) can be used for deriving an interlacing property for the eigenvalues of a matrix in $\mathbf{R}^{n\times n}$ (of what kind?) and the eigenvalues of its principal submatrix in $\mathbf{R}^{(n-1)\times(n-1)}$.

Problems

1. An important problem is to decide if all the eigenvalues of a matrix A have negative real part. Such a matrix is called **stable**. Show that if

   $$\text{Re}(a_{ii}) + r_i \le 0, \quad \forall i,$$

   and $\text{Re}(a_{ii}) + r_i < 0$ for at least one i, then the matrix A is stable if A is irreducible.
2. Suppose that the matrix A is real, and all Gerschgorin discs of A are distinct. Show that from Theorem 9.3.2 it follows that all eigenvalues of A are real.
3. Show that all eigenvalues to a matrix A lie in the union of the disks

   $$|z - a_{ii}| \le \frac{1}{d_i} \sum_{j=1, j\ne i}^{n} d_j |a_{ij}|, \qquad i = 1, 2, \ldots, n,$$

   where d_i, $i = 1, 2, \ldots, n$ are given positive scale factors.

   Hint: Use the fact that the eigenvalues are invariant under similarity transformations.

4. Let $A \in \mathbf{C}^{n\times n}$, and assume that $\epsilon = \max_{i\neq j} |a_{ij}|$ is small. Choose the diagonal matrix $D = \operatorname{diag}(\mu, 1, \ldots, 1)$ so that the first Gerschgorin disk of DAD^{-1} is as small as possible, without overlapping the other disks. Show that if the diagonal elements of A are distinct then

$$\mu = \frac{\epsilon}{\delta} + O(\epsilon^2), \quad \delta = \min_{i\neq 1} |a_{ii} - a_{11}|,$$

and hence the first Gerschgorin disk is given by

$$|\lambda - a_{11}| \leq r_1, \quad r_1 \leq (n-1)\epsilon^2/\delta + O(\epsilon^3).$$

5. Compute the eigenvalues of B and A, where

$$B = \begin{pmatrix} 0 & \epsilon \\ \epsilon & 0 \end{pmatrix}, \qquad A = \begin{pmatrix} 0 & \epsilon & 0 \\ \epsilon & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

Show that they interlace.

6. Use a suitable diagonal similarity and Gerschgorin's theorem to show that the eigenvalues of the tridiagonal matrix

$$T = \begin{pmatrix} a & b_2 & & & \\ c_2 & a & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a & b_n \\ & & & c_n & a \end{pmatrix}.$$

satisfy the inequality

$$|\lambda - a| < 2\sqrt{\max_i |b_i| \max_i |c_i|}.$$

7. Let A and B be square Hermitian matrices and

$$H = \begin{pmatrix} A & C \\ C^H & B \end{pmatrix}.$$

Show that for every eigenvalue $\lambda(B)$ of B there is an eigenvalue $\lambda(H)$ of H such that

$$|\lambda(H) - \lambda(B)| \leq (\|C^H C\|_2)^{1/2}.$$

Hint: Use the estimate (9.3.20).

8. (a) Let $D = \operatorname{diag}(d_i)$ and $z = (z_1, \ldots, z_n)^T$. Show that if $\lambda \neq d_i$, $i = 1, \ldots, n$, then

$$\det(D + \mu z z^T - \lambda I) = \det\big((D - \lambda I)(I + (D - \lambda I)^{-1}\mu z z^T)\big).$$

Using the identity $\det(I + xy^T) = 1 + y^T x$ conclude that the eigenvalues λ of $D + \mu z z^T$ are the roots of the **secular equation**

$$f(\lambda) = 1 + \mu \sum_{i=1}^{n} \frac{z_i^2}{d_i - \lambda} = 0.$$

(b) Show by means of Fischer's Theorem 9.3.8 that the eigenvalues λ_i interlace the elements d_i so that if , for example, $\mu \geq 0$ then

$$d_1 \leq \lambda_1 \leq d_2 \leq \lambda_2 \leq \cdots \leq d_n \leq \lambda_n.$$

9.4 The Power Method

9.4.1 The Simple Power Method

One of the oldest methods for computing eigenvalues and eigenvectors of a matrix is the **power method**. For a long time the power method was the only alternative for finding the eigenvalues of a general non-Hermitian matrix. It is still one of the few practical methods when the matrix A is very large and sparse. Although it is otherwise no longer much used in its basic form for computing eigenvalues it is central to the convergence analysis of many currently used algorithms. A variant of the power method is also a standard method for computing eigenvectors when an accurate approximation to the corresponding eigenvalue is known.

Let $A \in \mathbf{R}^{n\times n}$ and $q_0 \neq 0$ be a given starting vector. In the power method the sequence of vectors $q_1, q_2, \ldots$ is formed, where

$$q_k = Aq_{k-1}, \qquad k = 1, 2, \ldots.$$

It follows that $q_k = A^k q_0$, which explains the name of the method. Note that in general it would be much more costly to form the matrix A^k, than to perform the above sequence of matrix vector multiplications.

We assume in the following that the eigenvalues are ordered so that

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

To simplify the analysis of the power method assume that the matrix A is diagonalizable. Then the initial vector q_0 can be expanded along the eigenvectors x_i of A, $q_0 = \sum_{j=1}^{n} \alpha_j x_j$, and we have

$$q_k = \sum_{j=1}^{n} \lambda_j^k \alpha_j x_j = \lambda_1^k \Big(\alpha_1 x_1 + \sum_{j=2}^{n} \Big(\frac{\lambda_j}{\lambda_1}\Big)^k \alpha_j x_j \Big), \quad k = 1, 2, \cdots$$

If λ_1 is a unique eigenvalue of maximum magnitude, $|\lambda_1| > |\lambda_2|$, we say that λ_1 is a **dominant eigenvalue**. If $\alpha_1 \neq 0$, then

$$\frac{1}{\lambda_1^k} q_k = \alpha_1 x_1 + O\left(\Big|\frac{\lambda_2}{\lambda_1}\Big|^k \right), \tag{9.4.1}$$

and up to a factor λ_1^k the vector q_k will converge to a limit vector which is an eigenvector associated with the dominating eigenvalue λ_1. The *rate of convergence is linear and equals* $|\lambda_2|/|\lambda_1|$. One can show that this result holds also when A is not diagonalizable by writing q_0 as a linear combination of the vectors associated with the Jordan (or Schur) canonical form of A, see Theorem 9.2.7 (9.2.1).

In practice the vectors q_k have to be normalized in order to avoid overflow or underflow. Hence we modify the initial recursion as follows. Assume that $\|q_0\|$=1, and compute

$$\hat{q}_k = Aq_{k-1}, \quad \mu_k = \|\hat{q}_k\|, \quad q_k = \hat{q}_k/\mu_k, \quad k = 1, 2, \ldots \tag{9.4.2}$$

Then we have

$$q_k = \frac{1}{\gamma_k} A^k q_0, \quad \gamma_k = \mu_1 \cdots \mu_k,$$

and under the assumptions above q_k converges to a normalized eigenvector x_1. From equations (9.4.1) and (9.4.2) it follows that

$$\hat{q}_k = \lambda_1 q_{k-1} + O(|\lambda_2/\lambda_1|^k), \qquad \lim_{k\to\infty} \mu_k = |\lambda_1|. \tag{9.4.3}$$

An approximation to λ_1 can also be obtained from the ratio of elements in the two vectors $\hat{q}_k$ and q_{k-1}. The convergence, which is slow when $|\lambda_2| \approx |\lambda_1|$, can be accelerated by Aitken extrapolation.

If the matrix A is real symmetric (or Hermitian) its eigenvalues are real and the eigenvectors can be chosen so that $X = (x_1, \ldots, x_n)$ is real and orthogonal. Using (9.4.1) one can show that the Rayleigh quotient converges twice as fast as μ_k,

$$\lambda_1 = \rho(q_{k-1}) + O\Big(|\lambda_2/\lambda_1|^{2k}\Big), \quad \rho(q_{k-1}) = q_{k-1}^T A q_{k-1} = q_{k-1}^T \hat{q}_k. \tag{9.4.4}$$

Example 9.4.1.

The eigenvalues of the matrix

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 4 \end{pmatrix}$$

are (4.732051, 3, 1.267949), correct to 6 decimals. If we take $q_0 = (1, 1, 1)^T$ then we obtain the Rayleigh quotients ρ_k and errors $e_k = \lambda_1 - \rho_k$ given in the table below:

k	ρ_k	e_k	e_k/e_{k-1}
1	4.333333	0.398718	
2	4.627119	0.104932	0.263
3	4.694118	0.037933	0.361
4	4.717023	0.015027	0.396
5	4.729620	0.006041	0.402

The ratios of successive errors converge to $(\lambda_2/\lambda_1)^2 = 0.4019$.

The convergence of the power method depends on the assumption that $\alpha_1 \neq 0$, and hence we only can prove convergence for *almost all starting vectors*. Even when $\alpha_1 = 0$, rounding errors will tend to introduce a small component along x_1 in Aq_0,

and therefore the method converges in practice also in this case. Convergence of the power method can also be shown under the weaker assumption that $\lambda_1 = \lambda_2 = \cdots = \lambda_r$, and

$$|\lambda_r| > |\lambda_{r+1}| \geq \cdots \geq |\lambda_n|.$$

However, an inherent weakness in this case is that the limit vector will depend on the expansion of q_0 along $x_1, \cdots, x_r$, and q_k will converge to *one particular vector* in the invariant subspace span$[x_1, \ldots, x_r]$. To determine the whole dominating invariant subspace we will have to perform the power method with $p \geq r$ linearly independent starting vectors, see Section 9.4.6.

An attractive feature of the power method is that the matrix A is not explicitly needed. It suffices to be able to form the matrix times vector product Ay for any given vector y. If the matrix A is sparse the cost of one iteration step is proportional to the number of nonzero elements in A.

9.4.2 Deflation

The simple power method can be used for computing several eigenvalues and the associated eigenvectors by combining it with **deflation**. By that we mean a method that given an eigenvector x_1 and the corresponding eigenvalue λ_1 computes a matrix A_1 such that $\lambda(A) = \lambda_1 \cup \lambda(A_1)$. A way to construct such a matrix A_1 in a stable way was indicated in Section 9.1, see (9.1.16). However, this method has the drawback that even if A is sparse the matrix A_1 will in general be dense.

The following simple method for deflation is due to Hotelling. Suppose an eigenpair (λ_1, x_1), $\|x_1\|_2 = 1$, of a symmetric matrix A is known. If we define $A_1 = A - \lambda_1 x_1 x_1^H$, then from the orthogonality of the eigenvectors x_i we have

$$A_1 x_i = Ax_i - \lambda_1 x_1 (x_1^T x_i) = \begin{cases} 0, & \text{if } i = 1; \\ \lambda_i x_i, & \text{if } i \neq 1. \end{cases}$$

Hence the eigenvalues of A_1 are $0, \lambda_2, \ldots, \lambda_n$ with corresponding eigenvectors equal to $x_1, x_2, \ldots, x_n$. The power method can now be applied to A_1 to determine the dominating eigenvalue of A_1. Note that $A_1 = A - \lambda_1 x_1 x_1^T = (I - x_1 x_1^T)A = P_1 A$, where P_1 is an orthogonal projection.

When A is unsymmetric there is a corresponding deflation technique. Here it is necessary to have the left eigenvector y_1^T as well as the right x_1. If these are normalized so that $y_1^T x_1 = 1$, then we define A_1 by $A_1 = A - \lambda_1 x_1 y_1^T$. From the biorthogonality of the x_i and y_i we have

$$A_1 x_i = Ax_i - \lambda_1 x_1 (y_1^T x_i) = \begin{cases} 0, & \text{if } i = 1; \\ \lambda_i x_i, & \text{if } i \neq 1. \end{cases}$$

In practice an important advantage of this scheme is that it is not necessary to form the matrix A_1 explicitly. The power method, as well as many other methods, only requires that an operation of the form $y = A_1 x$ can be performed. This operation can be performed as

$$A_1 x = Ax - \lambda_1 x_1 (y_1^T x) = Ax - \tau x_1, \qquad \tau = \lambda_1 (y_1^T x).$$

Hence it suffices to have the vectors x_1, y_1 available as well as a procedure for computing Ax for a given vector x. Obviously this deflation procedure can be performed repeatedly, to obtain $A_2, A_3, \ldots$.

This deflation procedure has to be used with caution, since errors will accumulate. This can be disastrous in the nonsymmetric case, when the eigenvalues may be badly conditioned.

9.4.3 Spectral Transformation and Inverse Iteration

The simple power method has the drawback that convergence may be arbitrarily slow or may not happen at all. To overcome this difficulty we can use a **spectral transformation**, which we now describe. Let $p(x)$ and $q(x)$ be two polynomials such that $q(A)$ is nonsingular and define $r(A) = (q(A))^{-1}p(A)$. Then if A has an eigenvalue λ with corresponding eigenvector x it follows that $r(\lambda)$ is an eigenvalue of $r(A)$ with the same eigenvector x.

As a simple application of this assume that A is nonsingular and take $r(x) = 1/x$. Then the matrix $r(A) = A^{-1}$ has eigenvalues equal to $1/\lambda_i$. Hence from (9.4.3) it follows that if the eigenvalues of A satisfy

$$|\lambda_1| \geq \cdots \geq |\lambda_{n-1}| > |\lambda_n|$$

and the power method is applied to A^{-1}, then q_k will converge to the eigenvector x_n of A corresponding to λ_n. This is called **inverse iteration** , and was introduced by H. Wielandt in 1944.

Inverse iteration can also be applied to the matrix $A-\mu I$, where μ is a chosen **shift** of the spectrum. The eigenvalues of $(A - \mu I)^{-1}$ equal

$$\mu_j = (\lambda_j - \mu)^{-1}. \tag{9.4.5}$$

and the iteration can be written

$$(A - \mu I)\hat{q}_k = q_{k-1}, \qquad q_k = \hat{q}_k/\|\hat{q}_k\|_2, \qquad k = 1, 2, \ldots. \tag{9.4.6}$$

Note that there is no need to explicitly invert $A - \mu I$. Instead we compute a triangular factorization of $A - \mu I$, and in each step of (9.4.6) solve two triangular systems

$$L(U\hat{q}_k) = Pq_{k-1}, \qquad P(A - \mu I) = LU.$$

For a dense matrix A one step of the iteration (9.4.5) is therefore no more costly than one step of the simple power method. However, if the matrix is sparse the total number of nonzero elements in L and U may be much larger than in A. Note that if A is positive definite (or diagonally dominant) this property is in general not shared by the shifted matrix $(A - \mu I)$. Hence in general partial pivoting must be employed.

If μ is chosen sufficiently close to an eigenvalue λ_i, so that $|\lambda_i - \mu| \ll |\lambda_j - \mu|$, $\lambda_i \neq \lambda_j$ then $(\lambda_i - \mu)^{-1}$ is a dominating eigenvalue of B,

$$|\lambda_i - \mu|^{-1} \gg |\lambda_j - \mu|^{-1}, \qquad \lambda_i \neq \lambda_j. \tag{9.4.7}$$

Then q_k will converge fast to the eigenvector x_i, and an approximation $\bar{\lambda}_i$ to the eigenvalue λ_i of A is obtained from the Rayleigh quotient

$$\frac{1}{\lambda_i - \mu} \approx q_{k-1}^T (A - \mu I)^{-1} q_{k-1} = q_{k-1}^T \hat{q}_k,$$

where $\hat{q_k}$ satisfies $(A - \mu I)\hat{q_k} = q_{k-1}$. Thus

$$\bar{\lambda}_i = \mu + 1/(q_{k-1}^T \hat{q}_k). \tag{9.4.8}$$

An a posteriori bound for the error in the approximate eigenvalue $\bar{\lambda}_i$ of A can be obtained from the residual corresponding to $(\bar{\lambda}_i, \hat{q}_k)$, which equals

$$r_k = A\hat{q}_k - \Big(\mu + 1/(q_{k-1}^T \hat{q}_k)\Big)\hat{q}_k = q_{k-1} - \hat{q_k}/(q_{k-1}^T \hat{q}_k).$$

Then, by Theorem 9.3.12, $(\bar{\lambda}_i, \hat{q}_k)$ is an exact eigenpair of a matrix $A + E$, where $\|E\|_2 \leq \|r_k\|_2/\|\hat{q}_k\|_2$. If A is real symmetric then the error in the approximative eigenvalue $\hat{\lambda}_i$ of A is bounded by $\|r_k\|_2/\|\hat{q}_k\|_2$.

9.4.4 Eigenvectors by Inverse Iteration

After extensive developments by Wilkinson and others inverse iteration has become the method of choice for computing the associated eigenvector to an eigenvalue λ_i, for which an accurate approximation already is known. Often just *one step* of inverse iteration suffices.

Inverse iteration will in general converge faster the closer μ is to λ_i. However, if μ equals λ_i up to machine precision then $A - \mu I$ in (9.4.6) is numerically singular. It was long believed that inverse iteration was doomed to failure when μ was chosen too close to an eigenvalue. Fortunately this is not the case!

If Gaussian elimination with partial pivoting is used the computed factorization of $(A - \mu I)$ will satisfy

$$P(A + E - \mu I) = \bar{L}\bar{U},$$

where $\|E\|_2/\|A\|_2 = f(n)O(u)$, and u is the unit roundoff and $f(n)$ a modest function of n (see Theorem 6.6.5). Since the rounding errors in the solution of the triangular systems usually are negligible the computed q_k will nearly satisfy

$$(A + E - \mu I)\hat{q}_k = q_{k-1}.$$

This shows that the inverse power method will give an approximation to an eigenvector of a slightly perturbed matrix $A + E$, independent of the ill-conditioning of $(A - \mu I)$.

To decide when a computed vector is a numerically acceptable eigenvector corresponding to an approximate eigenvalue we can apply the simple a posteriori error bound in Theorem 9.3.12 to inverse iteration. By (9.4.6) q_{k-1} is the residual vector corresponding to the approximate eigenpair $(\mu, \hat{q}_k)$. Hence, where u is the unit roundoff, $\hat{q}_k$ is a numerically acceptable eigenvector if

$$\|q_{k-1}\|_2/\|\hat{q}_k\|_2 \leq u\|A\|_2. \tag{9.4.9}$$

Example 9.4.2.

The matrix $A = \begin{pmatrix} 1 & 1 \\ 0.1 & 1.1 \end{pmatrix}$ has a simple eigenvalue $\lambda_1 = 0.7298438$ and the corresponding normalized eigenvector is $x_1 = (0.9653911, -0.2608064)^T$. We take $\mu = 0.7298$ to be an approximation to λ_1, and perform one step of inverse iteration, starting with $q_0 = (1, 0)^T$ we get

$$A - \mu I = LU = \begin{pmatrix} 1 & 0 \\ 0.37009623 & 1 \end{pmatrix} \begin{pmatrix} 0.2702 & 1 \\ 0 & 0.0001038 \end{pmatrix}$$

and $\hat{q}_1 = 10^4(1.3202568, -0.3566334)^T$, $q_1 = (0.9653989, -0.2607777)^T$, which agrees with the correct eigenvector to more than four decimals. From the backward error bound it follows that 0.7298 and q_1 is an exact eigenpair to a matrix $A + E$, where $\|E\|_2 \leq 1/\|\hat{q}_1\|_2 = 0.73122 \cdot 10^{-4}$.

Inverse iteration is a useful algorithm for calculation of specified eigenvectors corresponding to well separated eigenvalues for dense matrices. In order to save work in the triangular factorizations the matrix is usually first reduced to Hessenberg or real tridiagonal form, by the methods described in Section 9.6.

It is quite tricky to develop inverse iteration into a reliable algorithm in case the eigenvalues are not well separated. When A is symmetric and eigenvectors corresponding to multiple or very close eigenvalues are required, special steps have to be taken to ensure orthogonality of the eigenvectors. In the nonsymmetric case the situation can be worse in particular if the eigenvalue is defective or very ill-conditioned. Then, unless a suitable initial vector is used inverse iteration may not produce a numerically acceptable eigenvector. Often a random vector with elements from a uniform distribution in $[-1, 1]$ will work.

Example 9.4.3.

The matrix

$$A = \begin{pmatrix} 1 + \epsilon & 1 \\ \epsilon & 1 + \epsilon \end{pmatrix}$$

has eigenvalues $\lambda = (1 + \epsilon) \pm \sqrt{\epsilon}$. Assume that $|\epsilon| \approx u$, where u is the machine precision. Then the eigenpair $\lambda = 1$, $x = (1, 0)^T$ is a numerically acceptable eigenpair of A, since it is exact for the matrix $A + E$, where

$$E = -\begin{pmatrix} \epsilon & 0 \\ \epsilon & \epsilon \end{pmatrix}, \qquad \|E\|_2 < \sqrt{3}u.$$

If we perform one step of inverse iteration starting from the acceptable eigenvector $q_0 = (1, 0)^T$ then we get

$$\hat{q}_1 = \frac{1}{1 - \epsilon} \begin{pmatrix} -1 \\ 1 \end{pmatrix}.$$

No growth occurred and it can be shown that $(1, q_1)$ *is not* an acceptable eigenpair of A. If we carry out one more step of inverse iteration we will again get an acceptable eigenvector!

Equation (9.3.19) gives an expression for the backward error E of the computed eigenpair. An error bound can then be obtained by applying the perturbation analysis of Section 9.3. In the Hermitian case the eigenvalues are perfectly conditioned, and the error bound equals $\|E\|_2$. In general the sensitivity of an eigenvalue λ is determined by $1/s(\lambda) = 1/|y^H x|$, where x and y are right and left unit eigenvector corresponding to λ, see Section 9.3.2. If the power method is applied also to A^H (or in inverse iteration to $(A^H - \mu I)^{-1}$) we can generate an approximation to y and hence estimate $s(\lambda)$.

9.4.5 Rayleigh Quotient Iteration

A natural variation of the inverse power method is to vary the shift μ in each iteration. The previous analysis suggests choosing a shift equal to the Rayleigh quotient of the current eigenvector approximation. This leads to the **Rayleigh Quotient Iteration (RQI)**:

Let q_0, $\|q_0\|_2 = 1$, be a given starting vector, and for $k = 1, 2, \ldots$ compute

$$\big(A - \rho(q_{k-1})I\big)\hat{q}_k = q_{k-1}, \quad \rho(q_{k-1}) = q_{k-1}^T A q_{k-1}, \tag{9.4.10}$$

and set $q_k = \hat{q}_k/\|\hat{q}_k\|_2$. Here $\rho(q_{k-1})$ is the Rayleigh quotient of q_{k-1}.

RQI can be used to improve a given approximate eigenvector. It can also be used to find an eigenvector of A starting from any unit vector q_0, but then we cannot say to which eigenvector $\{q_k\}$ will converge. There is also a possibility that some unfortunate choice of starting vector will lead to endless cycling. However, it can be shown that such cycles are unstable under perturbations so this will not occur in practice.

In the RQI a new triangular factorization must be computed of the matrix $A - \rho(q_{k-1})I$ for each iteration step, which makes this algorithm much more expensive than ordinary inverse iteration. However, if the matrix A is, for example, of Hessenberg (or tridiagonal) form the extra cost is small. If the RQI converges towards an eigenvector corresponding to a *simple* eigenvalue then it can be shown that convergence is quadratic. More precisely, it can be shown that

$$\eta_k \le c_k \eta_{k-1}^2, \quad \eta_k = \|Aq_k - \rho(q_k)q_k\|_2,$$

where c_k changes only slowly, see Stewart [43, 1973, Section 7.2].

If the matrix A is real and symmetric (or Hermitian), then the situation is even more satisfactory because of the result in Theorem 9.3.14. This theorem says that if an eigenvector is known to precision ϵ, the Rayleigh quotient approximates the corresponding eigenvalue to precision ϵ^2. This leads to *cubic convergence* for the RQI for real symmetric (or Hermitian) matrices. Also, in this case it is no longer necessary to assume that the iteration converges to an eigenvector corresponding to a simple eigenvalue. Indeed, it can be shown that the for Hermitian matrices RQI has *global convergence*, i.e., it converges from *any starting vector* q_0. A key fact in the proof is that *the norm of the residuals always decrease*, $\eta_{k+1} \le \eta_k$, for all k, see Parlett [38, Section 4.8].

9.4.6 Subspace Iteration

A natural generalization of the power method is to iterate *simultaneously* with several vectors. Let $Z_0 = S = (s_1, \ldots, s_p) \in \mathbf{R}^{n\times p}$, be an initial matrix of rank $p > 1$. If we compute a sequence of matrices $\{Z_k\}$, from

$$Z_k = AZ_{k-1}, \qquad k = 1, 2, \ldots, \tag{9.4.11}$$

then it holds

$$Z_k = A^k S = (A^k s_1, \ldots, A^k s_p). \tag{9.4.12}$$

In applications A is often a very large sparse matrix and $p \ll n$.

At first it is not clear that we gain much by iterating with several vectors. If A has a dominant eigenvalue λ_1 *all the columns* of Z_k will converge to a scalar multiple of the dominant eigenvector x_1. Hence Z_k will be close to a matrix of numerical rank one.

We first note that we are really computing a sequence of subspaces. If $\mathcal{S} = \text{span}\,(S)$ the iteration produces the subspaces $A^k\mathcal{S} = \text{span}\,(A^k S)$. Hence the problem is that the basis $A^k s_1, \ldots, A^k s_p$ of this subspace becomes more and more ill-conditioned. This can be avoided by be maintaining orthogonality between the columns as follows: Starting with a matrix Q_0 with orthogonal columns we compute

$$Z_k = AQ_{k-1} = Q_k R_k, \qquad k = 1, 2, \ldots, \tag{9.4.13}$$

where $Q_k R_k$ is the QR decomposition of Z_k. Here Q_k can be computed, e.g., by Gram-Schmidt orthogonalization of Z_k. The iteration (9.4.13) is also called **orthogonal iteration**. Note that R_k plays the rule of a normalizing matrix. We have $Q_1 = Z_1 R_1^{-1} = AQ_0 R_1^{-1}$. Similarly it can be shown by induction that

$$Q_k = A^k Q_0 (R_k \cdots R_1)^{-1}. \tag{9.4.14}$$

It is important to note that if $Z_0 = Q_0$, then both iterations (9.4.11) and (9.4.13) will generate the same sequence of subspaces. $\mathcal{R}(A^k Q_0) = \mathcal{R}(Q_k)$. However, in orthogonal iteration an orthogonal bases for the subspace is calculated at each iteration. (Since the iteration (9.4.11) is less costly it is sometimes preferable to perform the orthogonalization in (9.4.13) only occasionally when needed.)

The method of orthogonal iteration overcomes several of the disadvantages of the power method. In particular it allows us to determine a dominant invariant subspace of a multiple eigenvalue.

Assume that the eigenvalues of A satisfy

$$|\lambda_1| \geq \cdots \geq |\lambda_p| > |\lambda_{p+1}| \geq \cdots \geq |\lambda_n| \tag{9.4.15}$$

and let

$$\begin{pmatrix} U_1^H \\ U_2^H \end{pmatrix} A(U_1\ U_2) = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}, \tag{9.4.16}$$

be a Schur decomposition of A, where

$$\text{diag}\,(T_{11}) = (\lambda_1, \cdots, \lambda_p)^H.$$

Then the subspace $\mathcal{U}_1 = \mathcal{R}(U_1)$ is a **dominant** invariant subspace of A. It can be shown that almost always the subspaces $\mathcal{R}(Q_k)$ in orthogonal iteration (9.4.13) converge to $\mathcal{U}_1$ when $k \to \infty$.

Theorem 9.4.1.

Let $\mathcal{U}_1 = \mathcal{R}(U_1)$ be a dominant invariant subspace of A defined in (9.4.16). *Let $\mathcal{S}$ be a p-dimensional subspace of $\mathbf{C}^n$ such that $\mathcal{S} \cap \mathcal{U}_1^{\perp} = \{0\}$. Then there exists a constant C such that*

$$\theta_{\max}(A^k\mathcal{S}, \mathcal{U}_1) \le C|\lambda_{p+1}/\lambda_p|^k.$$

where $\theta_{\max}(\mathcal{X}, \mathcal{Y})$ denotes the largest angle between the two subspaces (see Definition 9.3.6).

Proof. See Golub and Van Loan [21, pp. 333]. □

If we perform subspace iteration on p vectors, we are simultaneously performing subspace iteration on a nested sequence of subspaces

$$\text{span}\,(s_1), \quad \text{span}\,(s_1, s_2), \ldots, \quad \text{span}\,(s_1, s_2, \ldots, s_p).$$

This is also true for orthogonal iteration since this property is not changed by the orthogonalization procedure. Hence Theorem 9.4.1 shows that whenever $|\lambda_{q+1}/\lambda_q|$ is small for some $q \le p$, the convergence to the corresponding dominant invariant subspace of dimension q will be fast.

We now show that there is a duality between direct and inverse subspace iteration.

Lemma 9.4.2. (Watkins [1982])

Let $\mathcal{S}$ and $\mathcal{S}^{\perp}$ be orthogonal complementary subspaces of $\mathbf{C}^n$. Then for all integers k the spaces $A^k\mathcal{S}$ and $(A^H)^{-k}\mathcal{S}^{\perp}$ are also orthogonal.

Proof. Let $x \perp y \in \mathbf{C}^n$. Then $(A^k x)^H (A^H)^{-k} y = x^H y = 0$ and thus $A^k x \perp (A^H)^{-k} y$. □

This duality property means that the two sequences

$$S, AS, A^2S, \ldots, \qquad S^{\perp}, (A^H)^{-1}S^{\perp}, (A^H)^{-2}S^{\perp}, \ldots$$

are equivalent in that they yield orthogonal complements! This result will be important in Section 9.7.1 for the understanding of the QR algorithm.

Approximations to eigenvalues of A can be obtained from eigenvalues of the sequence of matrices

$$B_k = Q_k^T A Q_k = Q_k^T Z_{k+1} \in \mathbf{R}^{p \times p}. \tag{9.4.17}$$

Note that B_k is a generalized Rayleigh quotient, see Section 9.8.1– 9.8.2. Finally, both direct and inverse orthogonal iteration can be performed using a sequence of shifted matrices $A - \mu_k I$, $k = 0, 1, 2, \ldots$.

Review Questions

1. Describe the power method and its variants. Name at least one important application of the shifted inverse power method.
2. If the Rayleigh Quotient Iteration converges to a simple eigenvalue of a general matrix A, what is the asymptotic rate of convergence? If A is Hermitian, what can you say then?
3. Describe how the power method can be generalized to simultaneously iterating with several starting vector.

Problems

1. Let $A \in \mathbf{R}^{n\times n}$ be a symmetric matrix with eigenvalues satisfying $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_{n-1} > \lambda_n$. Show that the choice $\mu = (\lambda_2 + \lambda_n)/2$ gives fastest convergence towards the eigenvector corresponding to λ_1 in the power method applied to $A - \mu I$. What is this rate of convergence?
2. The matrix A has one real eigenvalue $\lambda = \lambda_1$ and another $\lambda = -\lambda_1$. All remaining eigenvalues satisfy $|\lambda| < |\lambda_1|$. Generalize the simple power method so that it can be used for this case.
3. (a) Compute the residual vector corresponding to the last eigenpair obtained in Example 9.4.1, and give the corresponding backward error estimate.

   (b) Perform Aitken extrapolation on the Rayleigh quotient approximations in Example 9.4.1 to compute an improved estimate of λ_1.
4. The symmetric matrix
$$A = \begin{pmatrix} 14 & 7 & 6 & 9 \\ 7 & 9 & 4 & 6 \\ 6 & 4 & 9 & 7 \\ 9 & 6 & 7 & 15 \end{pmatrix}$$
   has an eigenvalue $\lambda \approx 4$. Compute an improved estimate of λ with one step of inverse iteration using the factorization $A - 4I = LDL^T$.
5. For a symmetric matrix $A \in \mathbf{R}^{n\times n}$ it holds that $\sigma_i = |\lambda_i|$, $i = 1, \ldots, n$. Compute with inverse iteration using the starting vector $x = (1, -2,\ 1)^T$ the smallest singular value of the matrix
$$A = \begin{pmatrix} 1/5 & 1/6 & 1/7 \\ 1/6 & 1/7 & 1/8 \\ 1/7 & 1/8 & 1/9 \end{pmatrix}$$
   with at least two significant digits.
6. The matrix
$$A = \begin{pmatrix} 1 & 1 \\ \epsilon & 1+\epsilon \end{pmatrix}$$
   has two simple eigenvalues close to 1 if $\epsilon > 0$. For $\epsilon = 10^{-3}$ and $\epsilon = 10^{-6}$ first compute the smallest eigenvalue to six decimals, and then perform inverse iteration to determine the corresponding eigenvectors. Try as starting vectors both $x = (1, 0)^T$ and $x = (0, 1)^T$.

9.5 Jacobi Methods

9.5.1 Jacobi Methods for Real Symmetric Matrices

Jacobi's[7] method is one of the oldest methods for solving the eigenvalue problem for real symmetric (or Hermitian) matrices. It is at least three times slower than the QR algorithm, to be described in the next section. However, Jacobi's method is easily parallelized and there are problems, for which it should be prefered.

Jacobi's method is an efficient method when one has to solve eigenvalue problems for a sequence of matrices, differing only slightly from each other, or, equivalently, for computing eigenvalues of a nearly diagonal matrix. Jacobi's method, with a proper stopping criterion, can be shown to compute *all eigenvalues of symmetric positive definite matrices with uniformly better relative accuracy, than any algorithms which first reduces the matrix to tridiagonal form.* Note that, although the QR algorithm is backward stable (see Section 9.7), high relative accuracy can only be guaranteed for the larger eigenvalues (those near $\|A\|$ in magnitude).

The Jacobi method solves the eigenvalue problem for $A \in \mathbf{R}^{n\times n}$ by employing a sequence of similarity transformations

$$A_0 = A, \qquad A_{k+1} = J_k^T A_k J_k \tag{9.5.1}$$

such that the sequence of matrices A_k, $k = 1, 2, ...$ tends to a diagonal form. For each k, J_k is chosen as a plane rotations $J_k = G_{pq}(\theta)$, defined by a pair of indices (p, q), $p < q$, called the pivot pair. The angle θ is chosen so that the off-diagonal elements $a_{pq} = a_{qp}$ are reduced to zero, i.e. by solving a 2×2 subproblems. We note that only the entries in rows and columns p and q of A will change, and since symmetry is preserved only the upper triangular part of each A needs to be computed.

To construct the Jacobi transformation J_k we consider the symmetric 2×2 eigenvalue problem for the principal submatrix A_{pq} formed by rows and columns p and q. For simplicity of notation we rename $A_{k+1} = A'$ and $A_k = A$. Hence we want to determine $c = \cos\theta$, $s = \sin\theta$ so that

$$\begin{pmatrix} l_p & 0 \\ 0 & l_q \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}. \tag{9.5.2}$$

Equating the off-diagonal elements we obtain (as $a_{pq} = a_{qp}$)

$$0 = (a_{pp} - a_{qq})cs + a_{pq}(c^2 - s^2), \tag{9.5.3}$$

which shows that the angle θ satisfies

$$\tau \equiv \cot 2\theta = (a_{qq} - a_{pp})/(2a_{pq}), \qquad a_{pq} \neq 0. \tag{9.5.4}$$

The two diagonal elements a_{pp} and a_{qq} are transformed as follows,

$$a'_{pp} = c^2 a_{pp} - 2csa_{pq} + s^2 a_{qq} = a_{pp} - ta_{pq},$$
$$a'_{qq} = s^2 a_{pp} + 2csa_{pq} + c^2 a_{qq} = a_{qq} + ta_{pq}.$$

[7] Carl Gustf Jacob Jacobi (1805–1851), German mathematician. Jacobi joined the faculty of Berlin university in 1825. Like Euler, he was a profilic calculator, who drew a great deal of insight from immense algorithmical work. His method for computing eigenvalues was publsihed in 1846; see [27].

where $t = \tan\theta$. We call this a **Jacobi transformation**. The following stopping criterion should be used:

$$\text{if } |a_{ij}| \leq \text{tol}\,(a_{ii}a_{jj})^{1/2}, \text{set } a_{ij} = 0, \tag{9.5.5}$$

where tol is the relative accuracy desired.

A stable way to perform a Jacobi transformation is to first compute $t = \tan\theta$ as the root of smallest modulus to the quadratic equation $t^2+2\tau t-1 = 0$. This choice ensures that $|\theta| < \pi/4$, and can be shown to minimize the difference $\|A' - A\|_F$. In particular this will prevent the exchange of the two diagonal elements a_{pp} and a_{qq}, when a_{pq} is small, which is critical for the convergence of the Jacobi method. The transformation (9.5.2) is best computed by the following algorithm.

Algorithm 9.5.1
Jacobi transformation matrix ($a_{pq} \neq 0$):

$$\begin{aligned}
&[c, s, l_p, l_q] = \text{jacobi}(a_{pp}, a_{pq}, a_{qq})\\
&\quad \tau = (a_{qq} - a_{pp})/(2a_{pq});\\
&\quad t = \text{sign}\,(\tau)/(|\tau| + \sqrt{1+\tau^2});\\
&\quad c = 1/\sqrt{1+t^2}; \quad s = t \cdot c;\\
&\quad l_p = a_{pp} - ta_{pq};\\
&\quad l_q = a_{qq} + ta_{pq};\\
&\quad \text{end}
\end{aligned}$$

The computed transformation is applied also to the remaining elements in rows and columns p and q of the full matrix A. These are transformed for $j \neq p, q$ according to

$$\begin{aligned}
a'_{jp} = a'_{pj} &= ca_{pj} - sa_{qj} = a_{pj} - s(a_{qj} + ra_{pj}),\\
a'_{jq} = a'_{qj} &= sa_{pj} + ca_{qj} = a_{qj} + s(a_{pj} - ra_{qj}).
\end{aligned}$$

where $r = s/(1 + c) = \tan(\theta/2)$. (The formulas are written in a form, due to Rutishauser [40, 1971], which reduces roundoff errors.)

If symmetry is exploited, then one Jacobi transformation takes about $4n$ flops. Note that an off-diagonal element made zero at one step will in general become nonzero at some later stage. The Jacobi method will also destroy the band structure if A is a banded matrix.

The convergence of the Jacobi method depends on the fact that in each step the quantity

$$S(A) = \sum_{i \neq j} a_{ij}^2 = \|A - D\|_F^2,$$

i.e., the Frobenius norm of the off-diagonal elements is reduced. To see this, we note that the Frobenius norm of a matrix is invariant under multiplication from left

or right with an orthogonal matrix. Therefore, since $a'_{pq} = 0$ we have

$$(a'_{pp})^2 + (a'_{qq})^2 = a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2.$$

We also have that $\|A'\|_F^2 = \|A\|_F^2$, and it follows that

$$S(A') = \|A'\|_F^2 - \sum_{i=1}^{n} (a'_{ii})^2 = S(A) - 2a_{pq}^2.$$

There are various strategies for choosing the order in which the off-diagonal elements are annihilated. Since $S(A')$ is reduced by $2a_{pq}^2$, the optimal choice is to annihilate the off-diagonal element of largest magnitude. This is done in the **classical Jacobi** method. Then since

$$2a_{pq}^2 \geq S(A_k)/N, \qquad N = n(n-1)/2,$$

we have $S(A_{k+1}) \leq (1-1/N)S(A_k)$. This shows that for the classical Jacobi method A_{k+1} converges at least linearly with rate $(1-1/N)$ to a diagonal matrix. In fact it has been shown that ultimately the rate of convergence is quadratic, so that for k large enough, we have $S(A_{k+1}) < cS(A_k)^2$ for some constant c. The iterations are repeated until $S(A_k) < \delta\|A\|_F$, where δ is a tolerance, which can be chosen equal to the unit roundoff u. From the Bauer–Fike Theorem 9.3.4 it then follows that the diagonal elements of A_k then approximate the eigenvalues of A with an error less than $\delta\|A\|_F$.

In the Classical Jacobi method a large amount of effort must be spent on searching for the largest off-diagonal element. Even though it is possible to reduce this time by taking advantage of the fact that only two rows and columns are changed at each step, the Classical Jacobi method is almost never used. In a **cyclic Jacobi method**, the $N = \frac{1}{2}n(n-1)$ off-diagonal elements are instead annihilated in some predetermined order, each element being rotated exactly once in any sequence of N rotations called a **sweep**. Convergence of any cyclic Jacobi method can be guaranteed if any rotation (p, q) is omitted for which $|a_{pq}|$ is smaller than some **threshold**; see Forsythe and Henrici [13, 1960]. To ensure a good rate of convergence this threshold tolerance should be successively decreased after each sweep.

For sequential computers the most popular cyclic ordering is the row-wise scheme, i.e., the rotations are performed in the order

$$\begin{array}{llll} (1,2), & (1,3), & \dots & (1,n) \\ & (2,3), & \dots & (2,n) \\ & & \dots & \dots \\ & & & (n-1,n) \end{array} \tag{9.5.6}$$

which is cyclically repeated. About $2n^3$ flops per sweep is required. In practice, with the cyclic Jacobi method not more than about 5 sweeps are needed to obtain eigenvalues of more than single precision accuracy even when n is large. The number of sweeps grows approximately as $\mathrm{O}(\log n)$, and about $10n^3$ flops are needed to

compute all the eigenvalues of A. This is about 3–5 times more than for the QR algorithm.

An orthogonal system of eigenvectors of A can easily be obtained in the Jacobi method by computing the product of all the transformations

$$X_k = J_1 J_2 \cdots J_k.$$

Then $\lim_{k\to\infty} X_k = X$. If we put $X_0 = I$, then we recursively compute

$$X_k = X_{k-1} J_k, \quad k = 1, 2, \ldots \tag{9.5.7}$$

In each transformation the two columns (p, q) of $X_{k-1}$is rotated, which requires $4n$ flop. Hence in each sweep an additional $2n$ flops is needed, which doubles the operation count for the method.

The Jacobi method is very suitable for parallel computation since several noninteracting rotations, (p_i, q_i) and (p_j, q_j), where p_i, q_i are distinct from p_j, q_j, can be performed simultaneously. If n is even the $n/2$ Jacobi transformations can be performed simultaneously. A sweep needs at least $n-1$ such parallel steps. Several parallel schemes which uses this minimum number of steps have been constructed. These can be illustrated in the $n = 8$ case by

$$(p,q) = \begin{array}{cccc} (1,2), & (3,4), & (5,6), & (7,8) \\ (1,4), & (2,6), & (3,8), & (5,7) \\ (1,6), & (4,8), & (2,7), & (3,5) \\ (1,8), & (6,7), & (4,5), & (2,3) \\ (1,7), & (8,5), & (6,3), & (4,2) \\ (1,5), & (7,3), & (8,2), & (6,4) \\ (1,3), & (5,2), & (7,4), & (8,6) \end{array}.$$

The rotations associated with *each row* of the above can be calculated simultaneously. First the transformations are constructed in parallel; then the transformations from the left are applied in parallel, and finally the transformations from the right.

9.5.2 Jacobi Methods for Computing the SVD.

Several Jacobi-type methods for computing the SVD $A = U\Sigma V^T$ of a matrix were developed in the 1950's. The shortcomings of some of these algorithms have been removed, and as for the real symmetric eigenproblem, there are cases for which Jacobi's method is to be preferred over the QR-algorithm for the SVD. In particular, it computes the smaller singular values more accurately than any algorithm based on a preliminary bidiagonal reduction.

There are two different ways to generalize the Jacobi method for the SVD problem. We assume that $A \in \mathbf{R}^{n\times n}$ is a square nonsymmetric matrix. This is no restriction, sincer we can first compute QR factorization

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

and then apply the Jacobi-SVD method to R. In the **two-sided Jacobi-SVD** algorithm for the SVD of A (Kogbetliantz [29]) the elementary step consists of two-sided Givens transformations

$$A' = J_{pq}(\phi) A J_{pq}^T(\psi), \tag{9.5.8}$$

where $J_{pq}(\phi)$ and $J_{pq}(\psi)$ are determined so that $a'_{pq} = a'_{qp} = 0$. Note that only rows and columns p and q in A are affected by the transformation. The rotations $J_{pq}(\phi)$ and $J_{pq}(\psi)$ are determined by computing the SVD of a 2×2 submatrix

$$A = \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}, \qquad a_{pp} \geq 0, \quad a_{qq} \geq 0.$$

The assumption of nonnegative diagonal elements is no restriction, since we can change the sign of these by premultiplication with an orthogonal matrix diag $(\pm 1, \pm 1)$.

Since the Frobenius norm is invariant under orthogonal transformations it follows that

$$S(A') = S(A) - (a_{pq}^2 + a_{qp}^2), \qquad S(A) = \|A - D\|_F^2.$$

This relation is the basis for a proof that the matrices generated by Kogbetliantz's method converge to a diagonal matrix containing the singular values of A. Orthogonal systems of left and right singular vectors can be obtained by accumulating the product of all the transformations.

The rotation angles can be determined as follows: First a Givens transformation is applied to the left to transform it into an upper triangular 2×2 matrix. If $r_{12} \neq 0$, then we set

$$\begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}^T \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix} \begin{pmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix} \tag{9.5.9}$$

where the rotation angles are determined by the formula

$$\tan 2\psi = \frac{2 r_{11} r_{12}}{r_{22}^2 - r_{11}^2 + r_{12}^2}, \tag{9.5.10}$$

$$\tan\phi = \frac{r_{12} + r_{11}\tan\psi}{r_{22}} = \frac{r_{22}\tan\psi}{r_{11} - r_{12}\tan\psi}. \tag{9.5.11}$$

For stability reasons, in the latter formula, the quotients of absolutely larger numbers are always taken An alternative algorithm for the SVD of 2×2 upper triangular matrix, which always gives high *relative accuracy* in the singular values and vectors, has been developed by Demmel and Kahan; see Problem 5.

At first a drawback of the above algorithm seems to be that it works all the time on a full $m \times n$ unsymmetric matrix. However, if a proper cyclig rotation strategy is used, then at each step the matrix will be essentially triangular. If the column cyclic strategy

$$(1,2), (1,3), (2,3), \ldots, (1,n), \ldots, (n-1,n)$$

is used an upper triangular matrix will be successievly transformed into a lower triangular matrix. The next sweep will transform it back to an upper triangular matrix. During the whole process the matrix can be stored in an upper triangular array. The initial QR factorization also cures some global convergence problems present in the twosided Jacobi-SVD method.

In the **one-sided Jacobi-SVD** algorithm Givens transformations are used to find an orthogonal matrix V such that the matrix AV has orthogonal columns. Then $AV = U\Sigma$ and the SVD of A is readily obtained. The columns can be explicitly interchanged so that the final columns of AV appear in order of decreasing norm. The basic step rotates two columns:

$$(\hat{a}_p, \hat{a}_q) = (a_p, a_q)\begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \qquad p < q. \tag{9.5.12}$$

The parameters c, s are determined so that the rotated columns are orthogonal, or equivalently so that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} \|a_p\|_2^2 & a_p^T a_q \\ a_q^T a_p & \|a_q\|_2^2 \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}^T$$

is diagonal. This 2×2 symmetric eigenproblem can be solved by a Jacobi transformation. To determine the rotation it is better to first compute the QR factorization

$$(a_p, a_q) = (q_1,\ q_2)\begin{pmatrix} r_{pp} & r_{pq} \\ 0 & r_{qq} \end{pmatrix} \equiv QR.$$

If now the 2×2 SVD $R = U\Sigma V^T$ is computed, using one of the algorithm given below, then since $RV = U\Sigma$

$$(a_p, a_q)V = (q_1,\ q_2)U\Sigma$$

will have orthogonal columns. It follows that V is the desired rotation in (9.5.12).

Clearly, the one-sided algorithm is mathematically equivalent to applying Jacobi's method to diagonalize $C = A^T A$, and hence its convergence properties are the same. Convergence of Jacobi's method is related to the fact that in each step the sum of squares of the off-diagonal elements

$$S(C) = \sum_{i \neq j} c_{ij}^2, \qquad C = A^T A$$

is reduced. Hence the rate of convergence is ultimately quadratic, also for multiple singular values. Note that the one-sided Jacobi SVD will by construction have U orthogonal to working accuracy, but loss of orthogonality in V may occur. Therefore the columns of V should be reorthogonalized using a Gram–Schmidt process at the end.

The one-sided method can be applied to a general real (or complex) matrix $A \in \mathbf{R}^{m \times n}$, $m \geq n$, but an intial QR factorization should performed to speed up convergence. If this is performed with *row and column pivoting*, then high

relative accuracy can be achieved for matrices A that are *diagonal scalings of a well-conditioned matrix*, that is which can be decomposed as

$$A = D_1 B D_2,$$

where $D_1,\ D_2$ are diagonal and B well-conditioned. It has been domonstrated that if presorting the rows after decreasing norm $\|a_{i,:}\|_\infty$ and then using column pivoting only gives equally good results. By a careful choice of the rotation sequence the esential triangularity of the matrix can be preserved during the Jacobi iterations.

In a cyclic Jacobi method, the off-diagonal elements are annihilated in some predetermined order, each element being rotated exactly once in any sequence of $N = n(n-1)/2$ rotations called a **sweep**. Parallel implementations can take advantage of the fact that noninteracting rotations, (p_i, q_i) and (p_j, q_j), where p_i, q_i and p_j, q_j are distinct, can be performed simultaneously. If n is even $n/2$ transformations can be performed simultaneously, and a sweep needs at least $n-1$ such parallel steps. In practice, with the cyclic Jacobi method not more than about five sweeps are needed to obtain singular values of more than single precision accuracy even when n is large. The number of sweeps grows approximately as $O(\log n)$.

The alternative algorithm for the SVD of 2×2 upper triangular matrix below always gives high *relative accuracy* in the singular values and vectors, has been developed by Demmel and Kahan, and is based on the relations in Problem 5.

Review Questions

1. What is the asymptotic speed of convergence for the classical Jacobi method? Discuss the advantages and drawbacks of Jacobi methods compared to the QR algorithm.
2. There are two different Jacobi-type methods for computing the SVD were developed. What are they called? What 2×2 subproblems are they based on?

Problems

1. Implement Jacobi's algorithm, using the stopping criterion (9.5.5) with tol $= 10^{-12}$. Use it to compute the eigenvalues of

$$A = \begin{pmatrix} -0.442 & -0.607 & -1.075 \\ -0.607 & 0.806 & 0.455 \\ -1.075 & 0.455 & -1.069 \end{pmatrix},$$

How many Jacobi steps are used?

2. Suppose the matrix

$$\tilde{A} = \begin{pmatrix} 1 & 10^{-2} & 10^{-4} \\ 10^{-2} & 2 & 10^{-2} \\ 10^{-4} & 10^{-2} & 4 \end{pmatrix}.$$

has been obtained at a certain step of the Jacobi algorithm. Estimate the eigenvalues of $\tilde{A}$ as accurately as possible using the Gerschgorin circles with a suitable diagonal transformation, see Problem 9.3.3.

3. Jacobi-type methods can also be constructed for Hermitian matrices using *elementary unitary rotations* of the form

$$U = \begin{pmatrix} \cos\theta & \alpha\sin\theta \\ -\bar{\alpha}\sin\theta & \cos\theta \end{pmatrix}, \qquad |\alpha| = 1.$$

Show that if we take $\alpha = a_{pq}/|a_{pq}|$ then equation (9.5.4) for the angle θ becomes

$$\tau = \cot 2\theta = (a_{pp} - a_{qq})/(2|a_{pq}|), \qquad |a_{pq}| \neq 0.$$

(Note that the diagonal elements a_{pp} and a_{qq} of a Hermitian matrix are real.)

4. Let $A \in \mathbf{C}^{2\times 2}$ be a given matrix, and U a unitary matrix of the form in Problem 3. Determine U so that the matrix $B = U^{-1}AU$ becomes upper triangular, that is, the Schur Canonical Form of A. Use this result to compute the eigenvalues of

$$A = \begin{pmatrix} 9 & 10 \\ -2 & 5 \end{pmatrix}.$$

Outline a Jacobi-type method to compute the Schur Canonical form of a general matrix A.

5. Consider the SVD of an upper triangular 2×2 matrix (9.5.9). where $\sigma_1 \geq \sigma_2$.
(a) Show that the singular values satisfy

$$\sigma_1\sigma_2 = |r_{11}r_{22}|, \quad \sigma_1^2 + \sigma_2^2 = r_{11}^2 + r_{22}^2 + r_{12}^2.$$

Deduce that

$$\sigma_{1,2} = \frac{1}{2}\Big|\sqrt{(r_{11}+r_{22})^2 + r_{12}^2} \pm \sqrt{(r_{11}-r_{22})^2 + r_{12}^2}\Big|, \tag{9.5.13}$$

of which the larger is σ_1 and the smaller $\sigma_2 = |r_{11}r_{22}|/\sigma_1$.
(b) Show that for the right singular vector $(s_v,\ c_v)$ is parallel to $(r_{11}^2 - \sigma_1^2, r_{11}r_{12})$. The left singular vectors then are obtained from

$$(c_u,\ s_u) = (r_{11}c_v - r_{12}s_v, r_{22}s_v)/\sigma_1.$$

SVD of 2×2 upper triangular matrix (9.5.9) with $|r_{11}| \geq |r_{22}|$:

$$
\begin{aligned}
&[c_u, s_u, c_v, s_v, \sigma_1, \sigma_2] = \mathrm{svd}(r_{11}, r_{12}, r_{22}) \\
&\quad l = (|r_{11}| - |r_{22}|)/|r_{11}|; \\
&\quad m = r_{12}/r_{11}; \quad t = 2 - l; \\
&\quad s = \sqrt{t^2 + m^2}; \quad r = \sqrt{l^2 + m^2}; \\
&\quad a = 0.5(s + r); \\
&\quad \sigma_1 = |r_{11}|a; \quad \sigma_2 = |r_{22}|/a; \\
&\quad t = (1 + a)(m/(s + t) + m/(r + l)); \\
&\quad l = \sqrt{t^2 + 4}; \\
&\quad c_v = 2/l; \quad s_v = -t/l; \\
&\quad c_u = (c_v - s_v m)/a; \quad s_u = s_v(r_{22}/r_{11})/a; \\
&\quad \text{end}
\end{aligned}
$$

6. Show that if Kogbetliantz's method is applied to a triangular matrix then after one sweep of the row cyclic algorithm (9.5.6) an upper (lower) triangular matrix becomes lower (upper) triangular.

9.6 Transformation to Condensed Form

9.6.1 Introduction

By Theorem 9.2.1 any matrix can be reduced to upper triangular form, the Schur canonical form, by a unitary similarity transformation. For a normal matrix this triangular form must necessarily be diagonal. In both cases we can read off the eigenvalues from the diagonal. The construction of the similarity transformation depended on the knowledge of successive eigenpairs, and this transformation can therefore in general not be realized by a finite process.

It is, however, possible to reduce a matrix to upper Hessenberg form, which is close to triangular, by a *finite* number of elementary similarity transformations. In the symmetric case, a symmetric tridiagonal form is obtained. In several algorithms for finding the eigenvalues and eigenvectors of a matrix the work is greatly reduced if this transformation is first carried out.

9.6.2 Unitary Elementary Transformations

For transformation of complex matrices to condensed form we need to consider **unitary** Givens and Householder transformations. To generalize Givens rotations to the complex case, we consider matrices of the form

$$
G = \begin{pmatrix} \bar{c} & \bar{s} \\ -s & c \end{pmatrix}, \qquad c = e^{i\gamma}\cos\theta, \qquad s = e^{i\delta}\sin\theta.
$$

It is easily verified that the matrix $G^H = G$, i.e., G is unitary, and that $G^{-1} = G^H$ is itself a plane rotation Given a complex vector $(x_1 \ x_2)^T \in \mathbf{C}^2$ we now want to

determine c and s so that

$$G\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \sigma \\ 0 \end{pmatrix}, \qquad \sigma^2 = |x_1|^2 + |x_2|^2, \tag{9.6.1}$$

Further, (9.6.1) holds provided that

$$c = x_1/\sigma, \qquad s = x_2/\sigma.$$

The following algorithm generalizes Algorithm 7.4.2 to the complex case:

Algorithm 9.6.1

Given $x = (x_1, x_2)^T \neq 0$ construct c, s, and real σ in a complex Givens rotation such that $Gx = \sigma(1, 0)^T$:

$[c, s, \sigma] = \text{givrot}(x_1, x_2)$
  if $|x_1| > |x_2|$
    $t = x_2/x_1$; $u = \sqrt{1 + |t|^2}$;
    $c = (x_1/|x_1|)/u$; $s = tc$; $\sigma = x_1/c$;
  else
    $t = x_1/x_2$; $u = \sqrt{1 + |t|^2}$;
    $s = (x_2/|x_2|)/u$; $c = ts$; $\sigma = x_2/s$;
  end

Householder transformations can also be generalized to the complex case. We consider **unitary** Householder transformations of the form

$$P = I - \frac{1}{\gamma}uu^H, \quad \gamma = \frac{1}{2}u^H u, \quad u \in \mathbf{C}^n. \tag{9.6.2}$$

It is easy to check that P is Hermitian, $P^H = P$, and unitary, $P^{-1} = P$. Given a vector $x \in \mathbf{C}^n$ we want to determine u such that $Px = ke_1$, $|k| = \sigma = \|x\|_2$. It is easily verified that if $x_1 = e^{i\alpha_1}|x_1|$ then u and γ are given by

$$u = x + ke_1, \quad k = \sigma e^{i\alpha_1}, \tag{9.6.3}$$

and

$$\gamma = \frac{1}{2}(\sigma^2 + 2|k||x_1| + |k|^2) = \sigma(\sigma + |x_1|). \tag{9.6.4}$$

Note that u differs from x only in its first component.

9.6.3 Reduction to Hessenberg Form

We now show how to reduce a matrix $A \in \mathbf{R}^{n\times n}$ to **Hessenberg** form by an orthogonal similarity,

$$Q^TAQ = H = \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1,n-1} & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2,n-1} & h_{2n} \\ & h_{32} & \ddots & \vdots & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & h_{n,n-1} & h_{nn} \end{pmatrix}.$$

The orthogonal matrix Q will be constructed as a product of $n-2$ Householder transformations $Q = P_1P_2\cdots P_{n-2}$, where

$$P_k = I - \frac{1}{\gamma_k}u_ku_k^T, \qquad \gamma_k = \frac{1}{2}\|u_k\|_2^2 \tag{9.6.5}$$

(cf. the Householder QR decomposition in Section 8.4.3). Note that P_k is completely specified by u_k and γ_k, and that products of the form PA and AP, can each be computed in $2n^2$ flops by

$$PA = A - u_k(A^Tu_k)^T/\gamma_k, \qquad AP = A - (Au_k)u_k^T/\gamma_k.$$

We compute $A = A^{(1)}, A^{(2)}, \ldots, A^{(n-1)} = H$, where $A^{(k+1)} = P_kA^{(k)}P_k$. In the first step, $k = 1$,

$$A^{(2)} = P_1AP_1 = \begin{pmatrix} h_{11} & h_{12} & \tilde{a}_{13} & \ldots & \tilde{a}_{1n} \\ h_{21} & h_{22} & \tilde{a}_{23} & \ldots & \tilde{a}_{2n} \\ 0 & \tilde{a}_{32} & \tilde{a}_{33} & \ldots & \tilde{a}_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \tilde{a}_{n2} & \tilde{a}_{n3} & \ldots & \tilde{a}_{nn} \end{pmatrix},$$

where P_1 is chosen so that P_1A has zeros in the first column in the positions shown above. These zeros are not destroyed by the post-multiplication $(P_1A)P_1$, which only affects the $n-1$ last columns. All later steps are similar. After $(k-1)$ steps we have computed

$$A^{(k)} = \begin{pmatrix} H_{11} & h_{12} & \tilde{A}_{13} \\ 0 & a_{22} & \tilde{A}_{23} \end{pmatrix}, \tag{9.6.6}$$

where $\begin{pmatrix} H_{11} & h_{12} \end{pmatrix} \in \mathbf{R}^{k\times k}$ is part of the final Hessenberg matrix. P_k is chosen to zero all elements but the first in a_{22}. After $n-2$ steps we have the required form

$$Q^TAQ = A^{(n-1)} = H, \qquad Q = P_1P_2\cdots P_{n-2}. \tag{9.6.7}$$

A simple operation count shows that this reduction requires $5n^3/3$ flops. Note that the transformation matrix Q is not explicitly computed, only the vectors defining the Householder transformations $P_1, P_2, \ldots, P_{n-2}$ are saved. These vectors can conveniently overwrite the corresponding elements in the matrix A using also two extra rows appended to A.

The Hessenberg decomposition $Q^TAQ = H$ is not unique. The following important theorem states that it is uniquely determined once the first column in Q is specified, provided that H has no zero subdiagonal element. A Hessenberg matrix with this property is said to be **unreduced**.

Theorem 9.6.1. *Implicit Q Theorem.*

Given $A, H, Q \in \mathbf{R}^{n\times n}$, where $Q = (q_1, \ldots, q_n)$ is orthogonal and $H = Q^TAQ$ is upper Hessenberg with positive subdiagonal elements. Then H and Q are uniquely determined by the first column q_1 in Q.

Proof. Assume we have already computed $q_1, \ldots, q_k$ and the first $k-1$ columns in H. (Since q_1 is known this assumption is valid for $k = 1$.) Equating the kth columns in $(q_1, q_2, \ldots, q_n)H = A(q_1, q_2, \ldots, q_n)$ we obtain

$$h_{1,k}q_1 + \cdots + h_{k,k}q_k + h_{k+1,k}q_{k+1} = Aq_k.$$

Multiplying this by q_i^T and using the orthogonality of Q, we obtain

$$h_{ik} = q_i^TAq_k, \qquad i = 1, \ldots, k.$$

Since H is unreduced $h_{k+1,k} \neq 0$, and therefore q_{k+1} and $h_{k+1,k}$ are determined (up to a factor of ± 1) by

$$q_{k+1} = h_{k+1,k}^{-1}\Big(Aq_k - \sum_{i=1}^{k} h_{ik}q_i\Big),$$

and the condition that $\|q_{k+1}\|_2 = 1$. ∎

The reduction by Householder transformations is stable in the sense that the computed $\bar{H}$ can be shown to be the *exact result* of an orthogonal similarity transformation of a matrix $A + E$, where

$$\|E\|_F \leq cn^2u\|A\|_F, \tag{9.6.8}$$

and c is a constant of order unity. Moreover if we use the information stored to generate the product $U = P_1P_2\cdots P_{n-2}$ then the computed result is close to the matrix U that reduces $A + E$. This will guarantee that the eigenvalues and transformed eigenvectors of $\bar{H}$ are accurate approximations to those of a matrix close to A. However, it should be noted that *this does not imply that the computed $\bar{H}$ will be close to the matrix H corresponding to the exact reduction of A.* Even the same algorithm run on two computers with different floating point arithmetic may produce very different matrices $\bar{H}$. Behavior of this kind, named **irrelevant instability** by B. N. Parlett, unfortunately continue to cause much unnecessary concern! The backward stability of the reduction ensures that each matrix will be similar to A to working precision and will yield approximate eigenvalues to as much absolute accuracy as is warranted.

The reduction to Hessenberg form can also be achieved by using elementary elimination matrices as introduced in Section 7.3.5. These are lower triangular matrices of the form

$$L_j = I + m_j e_j^T, \qquad m_j = (0, \ldots, 0, m_{j+1,j}, \ldots, m_{n,j})^T.$$

Only the elements *below* the main diagonal in the jth column differ from the unit matrix. If a matrix A is premultiplied by L_j we get

$$L_j A = (I + m_j e_j^T)A = A + m_j(e_j^T A) = A + m_j a_j^T,$$

i.e., multiples of the row a_j^T are *added* to the last $n-j$ rows of A. We complete the similarity transformation $L_j A L_j^{-1} = \tilde{A} L_j^{-1}$ by postmultiplying

$$\tilde{A} L_j^{-1} = \tilde{A}(I - m_j e_j^T) = \tilde{A} - (\tilde{A} m_j) e_j^T.$$

In this operation a linear combination $\tilde{A} m_j$ of the last $n-j$ columns is *subtracted* from the jth column of $\tilde{A}$.

If the pivot element $a_{21} \neq 0$, then we can eliminate the last $n-2$ elements in the first column of A by the transformation $L_2 A$, where

$$m_2 = -(0, 0, a_{31}/a_{21}, \ldots, a_{n1}/a_{21})^T.$$

These zeros are not affected by the postmultiplication $(L_2 A)L_2^{-1}$, which only affects the elements in the last $n-1$ columns. Hence, if all pivot elements are nonzero we can complete the transformation to Hessenberg form. The vectors m_j, $j = 2, \ldots, n-1$ can overwrite the corresponding elements of A. The reduction may be unstable if some pivot elements are small. Therefore, in practice this algorithm has to be modified by the introduction of partial pivoting, in obvious analogy to Gaussian elimination. With this modification the stability of the reduction is usually as good as for the one using Householder reflections. The backward error bound will contain a growth ratio g_n, see Section 7.6.6, but a big growth rarely occurs in practice. The operation count for this reduction can be shown to be $n^3/3 + n^3/2 = 5n^3/6$ flops, or half that for the orthogonal reduction. Because of this reduction by elementary elimination matrices is often the preferred method.

The similarity reduction of a nonsymmetric matrix to tridiagonal form has also been considered. This reduction is of interest also because of its relation to Lanczos bi-orthogonalization and the bi-conjugate gradient method; see Secs. 10.5.2–10.5.3. As shown by Wilkinson [52, pp. 388–405], this reduction can be performed in two steps: first an orthogonal similarity is used to reduce A to lower Hessenberg form; second the appropriate elements in the lower triangular half are zeroed column by column using a sequence of similarity transformations by elementary elimination matrices of the form in (6.3.15).

$$H := (I - m_j e_j^T)H(I + m_j e_j^T), \quad j = 1, \ldots, n-1.$$

In this step *row pivoting can not be used, since this would destroy the lower Hessenberg structure.* As a consequence, the reduction will fail if a zero pivot element is encountered. In this case one must restart the reduction from the beginning.

By (9.6.8) computed eigenvalues will usually have errors at least of order $u\|A\|_F$. Therefore it is desirable to precede the eigenvalue calculation by a diagonal similarity transformation $\tilde{A} = D^{-1}AD$ which reduces the Frobenius norm. (Note that only the off-diagonal elements are effected by such a transformation.) This can be achieved by **balancing** the matrix A. We say that a matrix $\tilde{A}$ is balanced for some norm l_p-norm if $\|\tilde{a}_i\|_p = \|\tilde{a}^i\|_p$, $i = 1, \ldots, n$ where $\tilde{a}_i$ and $\tilde{a}^i$ denote respectively the ith column and ith row of $\tilde{A}$. There are classes of matrices which do not need balancing; for example normal matrices are already balanced for $p = 2$.

An iterative algorithm has been given by Osborne that for any (real or complex) irreducible matrix A and $p = 2$ converges to a balanced matrix $\tilde{A}$. For a discussion and an implementation see Contribution II/11 in [53].

9.6.4 Reduction to Symmetric Tridiagonal Form

If we carry out the orthogonal reduction to Hessenberg form for a real symmetric matrix A, then

$$H^T = (Q^T AQ)^T = Q^T A^T Q = H.$$

It follows that H is a *real symmetric tridiagonal matrix*, which we write

$$Q^T AQ = T = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-1} & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}. \tag{9.6.9}$$

If elementary elimination matrices are used for the reduction *symmetry is not preserved*. Hence in this case the orthogonal reduction is clearly superior. A similar remark applies to the case of the unitary reduction of a Hermitian matrix to Hermitian tridiagonal form.

In the kth step of the orthogonal reduction of a real symmetric matrix we compute $A^{(k+1)} = P_k A^{(k)} P_k$, where P_k is again chosen to zero the last $n - k - 1$ elements in the kth column. By symmetry the corresponding elements in the kth row will be zeroed by the post-multiplication P_k.

It is important to take advantage of symmetry to save storage and operations. Since the intermediate matrix $P_k A^{(k)}$ is not symmetric, this means that we must compute $P_k A^{(k)} P_k$ directly. Dropping the subscripts k we can write

$$\begin{aligned} PAP &= \Big(I - \frac{1}{\gamma}uu^T\Big)A\Big(I - \frac{1}{\gamma}uu^T\Big) && (9.6.10)\\ &= A - up^T - pu^T + u^T puu^T/\gamma && (9.6.11)\\ &= A - uq^T - qu^T, \end{aligned}$$

where

$$p = Au/\gamma, \quad q = p - \beta u, \quad \beta = u^T p/(2\gamma). \tag{9.6.12}$$

If the transformations are carried out in this fashion the operation count for the reduction to tridiagonal form is reduced to about $2n^3/3$ flops,and we only need to store, say, the lower halves of the matrices.

The orthogonal reduction to tridiagonal form has the same stability property as the corresponding algorithm for the unsymmetric case, i.e., the computed tridiagonal matrix is the exact result for a matrix $A+E$, where E satisfies (9.6.8). Hence the eigenvalues of T will differ from the eigenvalues of A by at most $cn^2u\|A\|_F$.

There is a class of symmetric matrices for which small eigenvalues are determined with a very small error compared to $\|A\|_F$. This is the class of **scaled diagonally dominant** matrices, see Barlow and Demmel [3, 1990]. A symmetric scaled diagonally dominant (s.d.d) matrix is a matrix of the form DAD, where A is symmetric and diagonally dominant in the usual sense, and D is an arbitrary diagonal matrix. An example of a s.d.d. matrix is the **graded matrix**

$$A_0 = \begin{pmatrix} 1 & 10^{-4} & \\ 10^{-4} & 10^{-4} & 10^{-8} \\ & 10^{-8} & 10^{-8} \end{pmatrix}$$

whose elements decrease progressively in size as one proceeds diagonally from top to bottom. However, the matrix

$$A_1 = \begin{pmatrix} 10^{-6} & 10^{-2} & \\ 10^{-2} & 1 & 10^{-2} \\ & 10^{-2} & 10^{-6} \end{pmatrix}.$$

is neither diagonally dominant or graded in the usual sense.

The matrix A_0 has an eigenvalue λ of magnitude 10^{-8}, which is quite insensitive to small *relative* perturbations in the elements of the matrix. If the Householder reduction is performed starting from the *top* row of A as described here it is important that the matrix is presented so that the larger elements of A occur in the top left-hand corner. Then the errors in the orthogonal reduction will correspond to small relative errors in the elements of A, and the small eigenvalues of A will not be destroyed.[8]

A similar algorithm can be used to transform a Hermitian matrix into a tridiagonal Hermitian matrix using the complex Householder transformation introduced in Section 9.6.2. With $U = P_1P_2\cdots P_{n-2}$ we obtain $T = U^HAU$, where T is Hermitian and therefore has positive real diagonal elements. By a diagonal similarity DTD^{-1}, $D = \operatorname{diag}(e^{i\phi_1}, e^{i\phi_2}, \ldots, e^{i\phi_n})$ it is possible to further transform T so that the off-diagonal elements are real and nonnegative.

If the orthogonal reduction to tridiagonal form is carried out for a symmetric banded matrix A, then the banded structure will be destroyed. By annihilating pairs of elements using Givens rotations in an ingenious order it is possible to perform the reduction *without* increasing the bandwidth. However, it will then take several rotation to eliminate a single element. This algorithm is described in Parlett [38, Section 10.5.1], see also Contribution II/8 in Wilkinson and Reinsch [53]. An

[8] Note that in the Householder tridiagonalization described in [53], Contribution II/2 the reduction is performed instead from the bottom up.

operation count shows that the standard reduction is slower if the bandwidth is less than $n/6$. Note that the reduction of storage is often equally important!

9.6.5 A Divide and Conquer Algorithm

The basic idea in the divide and conquer algorithm for the symmetric tridiagonal eigenproblem is to divide the tridiagonal matrix (9.7.30) into two smaller symmetric tridiagonal matrices S and T_2 as follows.

$$T = \begin{pmatrix} T_1 & \beta_{k+1}e_k & 0 \\ \beta_{k+1}e_k^T & \alpha_{k+1} & \beta_{k+2}e_1^T \\ 0 & \beta_{k+2}e_1 & T_2 \end{pmatrix} = P \begin{pmatrix} \alpha_{k+1} & \beta_{k+1}e_k^T & \beta_{k+2}e_1^T \\ \beta_{k+1}e_k & T_1 & 0 \\ \beta_{k+2}e_1 & 0 & T_2 \end{pmatrix} P^T. \tag{9.6.13}$$

Here e_j is the jth unit vector of appropriate dimension and P is a permutation matrix permuting block rows and columns 1 and 2. T_1 and T_2 are $k \times k$ and $(n-k-1)\times(n-k-1)$ symmetric tridiagonal matrices and are principle submatrices of T.

Suppose now that the eigendecompositions of $T_i = Q_i D_i Q_i^T$, $i = 1, 2$ are known. Substituting into (9.6.13) we get

$$T = P \begin{pmatrix} \alpha_{k+1} & \beta_{k+1}e_k^T & \beta_{k+2}e_1^T \\ \beta_{k+1}e_k & Q_1 D_1 Q_1^T & 0 \\ \beta_{k+2}e_1 & 0 & Q_2 D_2 Q_2^T \end{pmatrix} P^T = QHQ^T, \tag{9.6.14}$$

where

$$H = \begin{pmatrix} \alpha_{k+1} & \beta_{k+1}l_1^T & \beta_{k+2}f_2^T \\ \beta_{k+1}l_1 & D_1 & 0 \\ \beta_{k+2}f_2 & 0 & D_2 \end{pmatrix}, \qquad Q = P \begin{pmatrix} 1 & 0 & 0 \\ 0 & Q_1 & 0 \\ 0 & 0 & Q_2 \end{pmatrix},$$

and $l_1 = Q_1^T e_k$, $f_2 = Q_2^T e_1$. Hence the matrix T is reduced to H by an orthogonal similarity transformation Q. The matrix H has the form

$$H = \begin{pmatrix} \alpha & z^T \\ z & D \end{pmatrix}, \quad D = \operatorname{diag}(d_2, \ldots, d_n).$$

where $z = (z_2, \ldots, z_n)^T$ is a vector. Such a matrix is called a **symmetric arrowhead matrix**. We assume that $d_2 \geq d_3 \geq \cdots \geq d_n$, which can be achieved by a symmetric permutation.

The eigenvalue problem for symmetric arrowhead matrices has been discussed in detail in Wilkinson [52, pp. 95–96]. In particular, if we assume that the elements d_i are distinct, $d_2 > d_3 > \cdots > d_n$, and that $z_i > 0$, $i = 2, \ldots, n$, then the eigenvalues and eigenvectors of H are characterized by the following lemma (cf. Problem 9.3.8).

Lemma 9.6.2.

The eigenvalues $\{\lambda_i\}_{i=1}^n$ of H satisfy the **secular equation**

$$f(\lambda) = \lambda - \alpha + \sum_{j=2}^{n} \frac{z_j^2}{d_j - \lambda} = 0. \tag{9.6.15}$$

and the interlacing property $\lambda_1 > d_2 > \lambda_2 > \cdots > d_n > \lambda_n$. *For each eigenvalue* λ_i *of* H, *a corresponding (unnormalized) eigenvector is given by*

$$u_i = \left(-1, \frac{z_2}{d_2 - \lambda_i}, \ldots, \frac{z_n}{d_n - \lambda_i}\right)^T. \tag{9.6.16}$$

Hence simple roots of the secular equation are isolated in an interval (d_i, d_{i+1}) where $f(\lambda)$ is monotonic and smooth. A zerofinder based on rational interpolation can be constructed which gets guaranteed quadratic convergence.

We make the following observations:

- If $d_i = d_{i+1}$ for some i, $2 \le i \le n-1$, then it can be shown that one eigenvalue of H equals d_i, and the degree of the secular equation may be reduced by one.
- If $z_i = 0$, then one eigenvalue equals d_i, and again the degree of the secular equation is decreased by one.

The splitting in (9.6.13) can be applied recursively to T_1 and T_2, i.e., we can repeat the splitting on each T_1 and T_2, etc., until the original tridiagonal matrix T has been reduced to a desired number of small subproblems. Then the relations in Lemma 9.6.2 may be applied from the bottom up to glue the eigensystems together.

In practice the formula for the eigenvectors in Lemma 9.6.2 cannot be used directly. The reason for this is that we can only compute an approximation $\hat{\lambda}_i$ to λ_i. Even if $\hat{\lambda}_i$ is very close to λ_i, the approximate ratio $z_j/(d_j - \hat{\lambda}_i)$ can be very different from the corresponding exact ratio. These errors may lead to computed eigenvectors of T which are numerically not orthogonal. Fortunately an ingenious solution to this problem has been found, which involves modifying the vector z rather than increasing the accuracy of the $\hat{\lambda}_i$, see Gu and Eisenstat [22, 1975]. The resulting algorithm seems to outperform the QR algorithm even on single processor computers.

9.6.6 Spectrum Slicing

Sylvester's law of inertia (see Theorem 7.3.8) leads to a simple and important method called **spectrum slicing** for counting the eigenvalues greater than a given real number τ of a Hermitian matrix A. In the following we treat the real symmetric case, but everything goes through also for general Hermitian matrices. The following theorem is a direct consequence of Sylvester's Law of Inertia.

Theorem 9.6.3.

Assume that symmetric Gaussian elimination can be carried through for $A - \tau I$ *yielding the factorization (cf. (6.4.5))*

$$A - \tau I = LDL^T, \quad D = \operatorname{diag}(d_1, \ldots, d_n), \tag{9.6.17}$$

where L *is a unit lower triangular matrix. Then* $A - \tau I$ *is congruent to* D, *and hence the number of eigenvalues of* A *greater than* τ *equals the number of positive elements* $\pi(D)$ *in the sequence* $d_1, \ldots, d_n$.

Example 9.6.1.

The LDL^T factorization

$$A - 1 \cdot I = \begin{pmatrix} 1 & 2 & \\ 2 & 2 & -4 \\ & -4 & -6 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & -2 & \\ & & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & \\ & 1 & 2 \\ & & 1 \end{pmatrix}.$$

shows that the matrix A has two eigenvalues greater than 1.

The LDL^T factorization may fail to exist if $A-\tau I$ is not positive definite. This will happen for example if we choose the shift $\tau = 2$ for the matrix in Example 9.6.1. Then $a_{11} - \tau = 0$, and the first step in the factorization cannot be carried out. A closer analysis shows that the factorization will fail if, and only if, τ equals an eigenvalue to one or more of the $n-1$ leading principal submatrices of A. If τ is chosen in a small interval around each of these values, big growth of elements occurs and the factorization may give the wrong count. In such cases one should perturb τ by a small amount and restart the factorization from the beginning.

For the special case when A is a symmetric tridiagonal matrix the procedure outlined above becomes particularly efficient and reliable. Here the factorization is $T - \tau I = LDL^T$, where L is unit lower bidiagonal and $D = \text{diag}\,(d_1, \ldots, d_n)$. The remarkable fact is that if we only take care to avoid over/underflow then *element growth will not affect the accuracy of the slice*.

Algorithm 9.6.2

Tridiagonal Spectrum Slicing Let T be the tridiagonal matrix (9.6.9). Then the number π of eigenvalues greater than a given number τ is generated by the following algorithm:

$d_1 := \alpha_1 - \tau;$
$\pi :=$ **if** $d_1 > 0$ **then** 1 **else** 0;
for $k = 2 : n$
  $d_k := (\alpha_k - \beta_k(\beta_k/d_{k-1})) - \tau;$
  if $|d_k| < \sqrt{\omega}$ **then** $d_k := \sqrt{\omega};$
  if $d_k > 0$ **then** $\pi := \pi + 1;$
end

Here, to prevent breakdown of the recursion, a small $|d_k|$ is replaced by $\sqrt{\omega}$ where ω is the underflow threshold. The recursion uses only $2n$ flops, and it is not necessary to store the elements d_k. The number of multiplications can be halved by computing initially β_k^2, which however may cause unnecessary over/underflow. Assuming that no over/underflow occurs Algorithm 9.6.6 is backward stable. A round-off error analysis shows that the computed values $\bar{d}_k$ satisfy exactly (let $\beta_1 = 0$)

$$\bar{d}_k = fl\big((\alpha_k - \beta_k(\beta_k/\bar{d}_{k-1})) - \tau\big)$$

$$= \Big(\Big(\alpha_k - \frac{\beta_k^2}{\bar{d}_{k-1}}(1+\epsilon_{1k})(1+\epsilon_{2k})\Big)(1+\epsilon_{3k}) - \tau\Big)(1+\epsilon_{4k}) \qquad (9.6.18)$$
$$\equiv \alpha_k' - \tau - (\beta_k')^2/\bar{d}_{k-1}, \qquad k = 1, \ldots, n,$$

where $|\epsilon_{ik}| \leq u$. Hence, the computed number $\bar{\pi}$ is the exact number of eigenvalues greater than τ of a matrix A', where A' has elements satisfying

$$|\alpha_k' - \alpha_k| \leq u(2|\alpha_k| + |\tau|), \qquad |\beta_k' - \beta_k| \leq 2u|\beta_k|. \qquad (9.6.19)$$

This is a very satisfactory backward error bound. It has been improved even further by Kahan [28, 1966], who shows that the term $2u|\alpha_k|$ in the bound can be dropped, see also Problem 1. Hence it follows that eigenvalues found by bisection differ by a factor at most $(1 \pm u)$ from the exact eigenvalues of a matrix where only the off-diagonal elements are subject to a relative perturbed of at most $2u$. This is obviously a very satisfactory result.

The above technique can be used to locate any individual eigenvalue λ_k of A. Assume we have two values τ_l and τ_u such that for the corresponding diagonal factors we have

$$\pi(D_l) \geq k, \qquad \pi(D_u) < k$$

so that λ_k lies in the interval $[\tau_l, \tau_u)$. We can then using p steps of the bisection (or multisection) method (see Section 6.1.1) locate λ_k in an interval of length $(\tau_u - \tau_l)/2^p$. From Gerschgorin's theorem it follows that all the eigenvalues of a tridiagonal matrix are contained in the union of the intervals $\alpha_i \pm (|\beta_i| + |\beta_{i+1}|)$, $i = 1, \ldots, n$ ($\beta_1 = \beta_{n+1} = 0$).

Using the bound (9.3.20) it follows that the bisection error in each computed eigenvalue is bounded by $|\bar{\lambda}_j - \lambda_j| \leq \|A' - A\|_2$, where from (9.4.11), using the improved bound by Kahan, and the inequalities $|\tau| \leq \|A\|_2$, $|\alpha_k| \leq \|A\|_2$ it follows that

$$|\bar{\lambda}_j - \lambda_j| \leq 5u\|A\|_2. \qquad (9.6.20)$$

This shows that the absolute error in the computed eigenvalues is always small. If some $|\lambda_k|$ is small it may be computed with poor *relative* precision. In some special cases (for example, tridiagonal, graded matrices see Section 9.6.4) even very small eigenvalues are determined to high relative precision by the elements in the matrix.

If many eigenvalues of a general real symmetric matrix A are to be determined by spectrum slicing, then A should initially be reduced to tridiagonal form. However, if A is a banded matrix and only few eigenvalues are to be determined then the Band Cholesky Algorithm 6.4.6 can be used to slice the spectrum. It is then necessary to monitor the element growth in the factorization. We finally mention that the technique of spectrum slicing is also applicable to the computation of selected singular values of a matrix and to the generalized eigenvalue problem

$$Ax = \lambda Bx,$$

where A and B are symmetric and B or A positive definite, see Section 9.9.

Review Questions

1. Describe how an arbitrary square matrix can be reduced to Hessenberg form by a sequence of orthogonal similarity transformations. If this reduction is applied to a real symmetric matrix what condensed form is obtained?
2. Describe the method of spectrum slicing for determining selected eigenvalues of a real symmetric matrix A.

Problems

1. Reduce to tridiagonal form, using an exact orthogonal similarity, the real symmetric matrix
$$A = \begin{pmatrix} 1 & \sqrt{2} & \sqrt{2} & \sqrt{2} \\ \sqrt{2} & -\sqrt{2} & -1 & \sqrt{2} \\ \sqrt{2} & -1 & \sqrt{2} & \sqrt{2} \\ 2 & \sqrt{2} & \sqrt{2} & -3 \end{pmatrix}$$

2. Show that if a real skew symmetric matrix A, $A^T = -A$, is reduced to Hessenberg form H by an orthogonal similarity, then H is a real skew symmetric tridiagonal matrix. Perform the reduction of the circulant matrix A (see Problem 9.1.9) with first row equal to
$$(0, 1, 1, 0, -1, -1).$$

3. To compute the eigenvalues of the following pentadiagonal matrix
$$A = \begin{pmatrix} 4 & 2 & 1 & 0 & 0 & 0 \\ 2 & 4 & 2 & 1 & 0 & 0 \\ 1 & 2 & 4 & 2 & 1 & 0 \\ 0 & 1 & 2 & 4 & 2 & 1 \\ 0 & 0 & 1 & 2 & 4 & 2 \\ 0 & 0 & 0 & 1 & 2 & 4 \end{pmatrix},$$
we first reduce A to tridiagonal form.
(a) Determine a Givens rotation G_{23} which zeros the element in position $(3, 1)$ in $G_{23}A$. Compute the the transformed matrix $A^{(1)} = G_{23}AG_{23}^T$.
(b) In the matrix $A^{(1)}$ a new nonzero element has been introduced. Show how this can be zeroed by a new rotation without introducing any new nonzero elements.
(c) Device a "zero chasing" algorithm to reduce a general real symmetric pentadiagonal matrix $A \in \mathbf{R}^{n\times n}$ to symmetric tridiagonal form. How many rotations are needed? How many flops?

4. (a) Use one Givens rotation to transform to tridiagonal form the matrix
$$A = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix}.$$
(b) Compute the largest eigenvalue of A, using spectrum slicing on the tridiagonal form derived in (a). Then compute the corresponding eigenvector.

5. Show that (9.6.17) can be written

$$\hat{d}_k = \alpha_k - \frac{\beta_k^2}{\hat{d}_{k-1}} \frac{(1+\epsilon_{1k})(1+\epsilon_{2k})}{(1+\epsilon_{3,k-1})(1+\epsilon_{4,k-1})} - \frac{\tau}{(1+\epsilon_{3k})}, \quad k = 1, \ldots, n,$$

where we have put $\bar{d}_k = \hat{d}_k(1+\epsilon_{3k})(1+\epsilon_{4k})$, and $|\epsilon_{ik}| \leq u$. Conclude that since $\text{sign}(\hat{d}_k) = \text{sign}(\bar{d}_k)$ the computed number $\bar{\pi}$ is the exact number of eigenvalues a tridiagonal matrix A' whose elements satisfy

$$|\alpha'_k - \alpha_k| \leq u|\tau|, \quad |\beta'_k - \beta_k| \leq 2u|\beta_k|.$$

9.7 The LR and QR Algorithms

When combined with a preliminary reduction to Hessenberg or symmetric tridiagonal form (see Section 9.6) the QR algorithm yields a very efficient method for finding all eigenvalues and eigenvectors of small to medium size matrices. Then the necessary modifications to make it into a practical method are described. The general nonsymmetric case is treated in Section 9.7.3 and the real symmetric case in Section 9.7.4.

9.7.1 The Basic LR and QR Algorithms

The LR algorithm, developed by Rutishauser in [39, 1958], is an iterative method of reducing a matrix to triangular form by a sequence of similarity transformations. Rutishauser observed that if $A = LR$ then a similarity transformation of A is

$$L^{-1}AL = L^{-1}(LR)L = RL.$$

Hence the matrix obtained by multiplying the factors in reverse order gives a matrix similar to A. The LR algorithm is obtained by repeating this process.

Setting $A_1 = A$ we compute $A_{k+1} = L_k^{-1}A_kL_k$ from

$$A_k = L_kR_k, \qquad A_{k+1} = R_kL_k, \quad k = 1, 2, \ldots \tag{9.7.1}$$

Repeated application of (9.7.1) gives

$$A_k = L_{k-1}^{-1} \cdots L_2^{-1}L_1^{-1}A_1L_1L_2 \cdots L_{k-1}. \tag{9.7.2}$$

or

$$L_1L_2 \cdots L_{k-1}A_k = A_1L_1L_2 \cdots L_{k-1}. \tag{9.7.3}$$

The two matrices defined by

$$T_k = L_1 \cdots L_{k-1}L_k, \qquad U_k = R_kR_{k-1} \cdots R_1, \tag{9.7.4}$$

are lower and upper triangular respectively. Forming the product T_kU_k and using (9.7.3) we have

$$\begin{aligned} T_kU_k &= L_1 \cdots L_{k-1}(L_kR_k)R_{k-1} \cdots R_1 \\ &= L_1 \cdots L_{k-1}A_kR_{k-1} \cdots R_1 \\ &= A_1L_1 \cdots L_{k-1}R_{k-1} \cdots R_1. \end{aligned}$$

Repeating this we obtain the basic relation

$$T_k U_k = A_1^k. \tag{9.7.5}$$

This shows that the close relation between the LR algorithm and the power method.

It is possible to show that under certain restrictions the matrix A_k converges to an upper triangular matrix R_∞. The eigenvalues are then equal to the diagonal elements of R_∞. In establishing the convergence result several assumptions need to be made. for example, that the LR factorization exists at every stage. This is not be true for the simple matrix

$$A = \begin{pmatrix} 0 & 1 \\ -3 & 4 \end{pmatrix},$$

with eigenvalues 1 and 3. Although we could equally well work with the shifted matrix $A + I$, which has a triangular factorization, there are other problems with the LR algorithm, which makes a robust implementation difficult.

In order to avoid the problems with the LR algorithm it seems natural to devise a similar algorithm using orthogonal similarity transformations. This leads to the QR algorithm, developed independently by Francis [14, 1961] and Kublanovskaya [31, 1961].[9] It then represented a significant and genuinely new contribution to eigensystems computation.

In the QR algorithm applied to $A_1 = A$ the matrix $A_{k+1} = Q_k^T A_k Q_k$, is computed from

$$A_k = Q_k R_k, \qquad A_{k+1} = R_k Q_k, \quad k = 1, 2, \ldots, \tag{9.7.6}$$

where Q_k is orthogonal and R_k is upper triangular, i.e., in the kth step we first compute the QR decomposition of the matrix A_k and then multiply the factors in reverse order to get A_{k+1}.

The successive iterates of the QR algorithm satisfy relations similar to those derived for the LR algorithm. We define

$$P_k = Q_1 Q_2 \cdots Q_k, \qquad U_k = R_k \cdots R_2 R_1,$$

where P_k is orthogonal and U_k is upper triangular. Then by repeated applications of (9.7.6) it follows that

$$A_{k+1} = P_k^T A P_k. \tag{9.7.7}$$

Further we have

$$\begin{aligned} P_k U_k &= Q_1 \cdots Q_{k-1}(Q_k R_k) R_{k-1} \cdots R_1 && (9.7.8)\\ &= Q_1 \cdots Q_{k-1} A_k R_{k-1} \cdots R_1 && (9.7.9)\\ &= A_1 Q_1 \cdots Q_{k-1} R_{k-1} \cdots R_1. && (9.7.10) \end{aligned}$$

Repeating this gives

$$P_k U_k = A_1^k. \tag{9.7.11}$$

[9] The QR algorithm was chosen as one of the 10 algorithms with most influence on scientific computing in the 20th century by the editors of the journal Computing in Science and Engineering.

When A is real symmetric and positive definite we can modify the LR algorithm and use the Cholesky factorization $A = LL^T$ instead. The algorithm then takes the form

$$A_k = L_k L_k^T, \qquad A_{k+1} = L_k^T L_k, \quad k = 1, 2, \ldots. \tag{9.7.12}$$

and we have

$$A_{k+1} = L_k^{-1} A_k L_k = L_k^T A_k L_k^{-T}. \tag{9.7.13}$$

Clearly all matrices A_k are symmetric and positive definite and the algorithm is well defined. Repeated application of (9.7.13) gives

$$A_k = T_{k-1}^{-1} A_1 T_{k-1} = T_{k-1}^T A_1 (T_{k-1}^{-1})^T, \tag{9.7.14}$$

where $T_k = L_1 L_2 \cdots L_k$. Further we have

$$A_1^k = (L_1 L_2 \cdots L_k)(L_k^T \cdots L_2^T L_1^T) = T_k T_k^T. \tag{9.7.15}$$

When A is real symmetric and positive definite there is a close relationship between the LR and QR algorithms. For the QR algorithm we have $A_k^T = A_k = R_k^T Q_k^T$ and hence

$$A_k^T A_k = A_k^2 = R_k^T Q_k^T Q_k R_k = R_k^T R_k, \tag{9.7.16}$$

which shows that R_k^T is the lower triangular Cholesky factor of A_k^2.

For the Cholesky LR algorithm we have from (9.7.4) and (9.7.5)

$$A_k^2 = L_k L_{k+1} (L_k L_{k+1})^T. \tag{9.7.17}$$

These two Cholesky factorizations (9.7.16) and (9.7.16) of the matrix A_k^2 must be the same and therefore $R_k^T = L_k L_{k+1}$. Thus

$$A_{k+1} = R_k Q_k = R_k A_k R_k^{-1} = L_{k+1}^T L_k^T A_k (L_{k+1}^T L_k^T)^{-1}.$$

Comparing this with (9.7.14) we deduce that one step of the QR algorithm is equivalent to two steps in the Cholesky LR algorithm. Hence the matrix $A_{(2k+1)}$ obtained by the Cholesky LR algorithm equals the matrix $A_{(k+1)}$ obtained using the QR algorithm.

We now show that in general the QR iteration is related to orthogonal iteration. Given an orthogonal matrix $\tilde{Q}_0 \in \mathbf{R}^{n\times n}$, orthogonal iteration computes a sequence $\tilde{Q}_1$, $\tilde{Q}_2, \ldots$, where

$$Z_k = A\tilde{Q}_k, \qquad Z_k = \tilde{Q}_{k+1} R_k. \qquad k = 0, 1, \ldots \tag{9.7.18}$$

The related sequence of matrices $B_k = \tilde{Q}_k^T A \tilde{Q}_k = \tilde{Q}_k^T Z_k$ similar to A can be computed directly. Using (9.7.18) we have $B_k = (\tilde{Q}_k^T \tilde{Q}_{k+1}) R_k$, which is the QR decomposition of B_k, and

$$B_{k+1} = (\tilde{Q}_{k+1}^T A)\tilde{Q}_{k+1} = (\tilde{Q}_{k+1}^T A \tilde{Q}_k)\tilde{Q}_k^T \tilde{Q}_{k+1} = R_k (\tilde{Q}_k^T \tilde{Q}_{k+1}).$$

Hence, B_{k+1} is obtained by multiplying the QR factors of B_k in reverse order, which is just one step of QR iteration! If, in particular we take $\tilde{Q}_0 = I$ then $B_0 = A_0$, and

it follows that $B_k = A_k$, $k = 0, 1, 2, \ldots$, where A_k is generated by the QR iteration (9.7.6). From the definition of B_k and (9.7.6) we have $\tilde{Q}_k = P_{k-1}$, and (compare (9.4.4))

$$A^k = \tilde{Q}_k \tilde{R}_k, \quad \tilde{R}_k = R_k \cdots R_2 R_1. \tag{9.7.19}$$

From this we can conclude that the first p columns of $\tilde{Q}_k$ form an orthogonal basis for the space spanned by the first p columns of A^k, i.e., $A^k(e_1, \ldots, e_p)$.

In the QR algorithm subspace iteration takes place on the subspaces spanned by the unit vectors $(e_1, \ldots, e_p)$, $p = 1, \ldots, n$. It is important for the understanding of the QR algorithm to recall that therefore, according to Theorem 9.4.1, also inverse iteration by $(A^H)^{-1}$ takes place on the orthogonal complements, i.e., the subspaces spanned by $(e_{p+1}, \ldots, e_n)$, $p = 0, \ldots, n-1$. Note that this means that in the QR algorithm direct iteration is taking place in the top left corner of A, and inverse iteration in the lower right corner. (For the QL algorithm this is reversed, see below.)

9.7.2 Convergence of the Basic QR Algorithm

Assume that the eigenvalues of A satisfy $|\lambda_p| > |\lambda_{p+1}|$, and let (9.4.16) be a corresponding Schur decomposition. Let $P_k = (P_{k1},\ P_{k2})$, $P_{k1} \in \mathbf{R}^{n \times p}$, be defined by (9.7.6). Then by Theorem 9.4.1 with linear rate of convergence equal to $|\lambda_{p+1}/\lambda_p|$

$$\mathcal{R}(P_{k1}) \to \mathcal{R}(U_1).$$

where U_1 spans the dominant invariant subspace of dimension p of A. It follows that A_k will tend to reducible form

$$A_k = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix} + O\left(\left(|\lambda_{p+1}/\lambda_p| \right)^k \right).$$

This result can be used to show that under rather general conditions A_k will tend to an upper triangular matrix R whose diagonal elements then are the eigenvalues of A.

Theorem 9.7.1.

If the eigenvalues of A satisfy $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, then the matrices A_k generated by the QR algorithm will tend to upper triangular form. The lower triangular elements $a_{ij}^{(k)}$, $i > j$, converge to zero with linear rate equal to $|\lambda_i/\lambda_j|$.

Proof. See Watkins [50]. □

If the product P_k, $k = 1, 2, \ldots$ of the transformations are accumulated the eigenvectors may then be found by calculating the eigenvectors of the final triangular matrix and then transforming them back.

To speed up convergence the QR algorithm can be applied to the matrix $\tilde{A} = A - \tau I$, where τ is a **shift**. If τ approximates a simple eigenvalue λ_j of A, then in general $|\lambda_i - \tau| \gg |\lambda_j - \tau|$ for $i \neq j$. By the result above the off-diagonal

elements in the *last* row of $\tilde{A}_k$ will approach zero very fast. Usually a different shift in used in each step. If further the shift is restored at the end of the step the QR iteration can be written

$$A_k - \tau_k I = Q_k R_k, \qquad R_k Q_k + \tau_k I = A_{k+1}, \qquad k = 0, 1, 2, \ldots, \tag{9.7.20}$$

It is easily verified that with this shifted QR iteration we have $A_{k+1} = Q_k^T A_k Q_k$, and the relation to the power method is now expressed by the following result.

Theorem 9.7.2.
Let Q_k and R_k be computed by the QR algorithm (9.7.20). Then

$$\begin{aligned} &(A - \tau_k I) \cdots (A - \tau_1 I)(A - \tau_0 I) = P_k U_k, \\ &P_k = Q_0 Q_1 \cdots Q_k, \qquad U_k = R_k R_{k-1} \cdots R_0. \end{aligned} \tag{9.7.21}$$

Proof. For $k = 0$ the relation (9.7.21) is just the defining equation of Q_0 and R_0. Assume now that the relation is true for $k - 1$. From $A_{k+1} = Q_k^T A_k Q_k$ and using the orthogonality of P_k

$$A_{k+1} - \tau_k I = P_k^T (A - \tau_k I) P_k. \tag{9.7.22}$$

Hence, $R_k = (A_{k+1} - \tau_k I) Q_k^T = P_k^T (A - \tau_k I) P_k Q_k^T = P_k^T (A - \tau_k I) P_{k-1}$. Postmultiplying this equation by U_{k-1} we get

$$R_k U_{k-1} = U_k = P_k^T (A - \tau_k I) P_{k-1} U_{k-1},$$

and thus $P_k U_k = (A - \tau_k I) P_{k-1} U_{k-1}$. Using the inductive hypothesis the theorem follows. □

A variant called the QL algorithm is based on the iteration

$$A_k = Q_k L_k, \qquad L_k Q_k = A_{k+1}, \qquad k = 0, 1, 2, \ldots, \tag{9.7.23}$$

where L_k is *lower* triangular, and is merely a reorganization of the QR algorithm. Let J be a permutation matrix such that JA reverses the rows of A. Then AJ reverses the columns of A and hence JAJ reverses both rows and columns. If R is upper triangular then JRJ is lower triangular. It follows that if $A = QR$ is the QR decomposition then $JAJ = (JQJ)(JRJ)$ is the QL decomposition of JAJ. It follows that the QR algorithm applied to A is the same as the QL algorithm applied to JAJ. The convergence theory is therefore the same for both algorithms. However, in the QL algorithm inverse iteration is taking place in the top left corner of A, and direct iteration in the lower right corner.

An important case where the choice of either the OR or QL algorithm should be preferred is when the matrix A is *graded*, see Section 9.6.4. If the large elements occur in the lower right corner then the QL algorithm is more stable. (Note that then the reduction to tridiagonal form should be done from bottom up; see the

remark in Section 9.6.4.) Of course, the same effect can be achieved by explicitly reversing the ordering of the rows and columns.

For a dense matrix the cost for one QR iteration is $4n^3/3$ flops, which is too much to make it a practical algorithm. However, if the matrix A is initially reduced, as described in Section 9.6, to upper Hessenberg form, or in the real symmetric case to tridiagonal form, this form is preserved by the QR iteration. The cost is then reduced to only $4n^2$ flops per iteration, or about $12n$ flops per iteration in the real symmetric case. The QR algorithm in practice also depends on several other factors to achieve full accuracy and efficiency. Some of these will be discussed in the following sections.

9.7.3 QR Algorithm for Hessenberg Matrices

We first show that Hessenberg form is preserved by the QR iteration. Let H_k be upper Hessenberg and for $k = 0, 1, 2, \ldots$

$$H_k - \tau_k I = Q_k R_k, \qquad R_k Q_k + \tau_k I = H_{k+1}. \tag{9.7.24}$$

First note that the addition or subtraction of $\tau_k I$ does not affect the Hessenberg form. If R_k is nonsingular then $Q_k = (H_k - \tau_k I)R_k^{-1}$ is a product of an upper Hessenberg matrix and an upper triangular matrix, and therefore again a Hessenberg matrix (cf. Problem 6.2.5). Hence $R_k Q_k$ and H_{k+1} are again of upper Hessenberg form.

In the **explicit-shift** QR algorithm we first form the matrix $H_k - \tau_k I$, and then apply a sequence of Givens rotations, $G_{j,j+1}$, $j = 1, \ldots, n-1$ (see (7.4.14)) so that

$$G_{n-1,n} \cdots G_{23} G_{12} (H_k - \tau_k I) = Q_k^T (H_k - \tau_k I) = R_k,$$

becomes upper triangular. At a typical step ($n = 5$, $j = 3$) the partially reduced matrix has the form

$$\begin{pmatrix} \rho_{11} & \times & \times & \times & \times \\ & \rho_{22} & \times & \times & \times \\ & & \nu_{33} & \times & \times \\ & & h_{43} & \times & \times \\ & & & \times & \times \end{pmatrix}.$$

The rotation $G_{3,4}$ is now chosen so that the element h_{43} is annihilated, which carries the reduction one step further. To form H_{k+1} we must now compute

$$R_k Q_k + \tau_k I = R_k G_{12}^T G_{23}^T \cdots G_{n-1,n}^T + \tau_k I.$$

The product $R_k G_{12}^T$ will affect only the first two columns of R_k, which are replaced by linear combinations of one another. This will add a nonzero element in the $(2, 1)$ position. The rotation G_{23}^T will similarly affect the second and third columns in $R_k G_{12}^T$, and adds a nonzero element in the $(3, 2)$ position. The final result is obviously a Hessenberg matrix.

If an upper Hessenberg matrix H has a zero subdiagonal entry, then we can write

$$H = \begin{pmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{pmatrix}.$$

The eigenvalues of H are then the sum of the eigenvalues of the two Hessenberg matrices H_{11} and H_{22}, and the eigenvalue problem **decouples** into two problems of smaller dimensions. In particular, if H_{22} is a scalar, then we have found an eigenvalue and the problem deflates.

If the shift τ is chosen as an exact eigenvalue of H, then $H - \tau I = QR$ has a zero eigenvalue and thus is singular. Since Q is orthogonal R must be singular. Moreover, if H is unreduced then the first $n-1$ columns of $H - \tau I$ are independent and therefore the *last* diagonal element r_{nn} must vanish. Hence the last row in RQ is zero, and the elements in the last row of $H' = RQ + \tau I$ are $h'_{n,n-1} = 0$ and $h'_{nn} = \tau$,

The above result shows that if the shift is equal to an eigenvalue τ then the QR algorithm converges in one step to this eigenvalue. This indicates that τ should be chosen as an approximation to an eigenvalue λ. Then $h_{n,n-1}$ will converge to zero at least with linear rate equal to $|\lambda - \tau| / \min_{\lambda' \neq \lambda} |\lambda' - \tau|$. The choice

$$\tau = h_{nn} = e_n^T H e_n$$

is called the **Rayleigh quotient shift**, since it can be shown to produce the same sequence of shifts as the RQI starting with the vector $q_0 = e_n$. With this shift convergence is therefore *asymptotically quadratic*.

If H is real with complex eigenvalues, then we obviously cannot converge to a complex eigenvalue using only real shifts. We could shift by the eigenvalue of

$$C = \begin{pmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{pmatrix}, \tag{9.7.25}$$

closest to $h_{n,n}$, although this has the disadvantage of introducing complex arithmetic even when A is real. A way to avoid this will be described later.

A important question is when to stop the iterations and accept an eigenvalue approximation. We set $h_{n,n-1} = 0$ and accept h_{nn} as an eigenvalue if

$$|h_{n,n-1}| \leq \epsilon(|h_{n-1,n-1}| + |h_{n,n}|),$$

where ϵ is a small constant times the unit roundoff. This criterion can be justified since it corresponds to a small backward error. In practice the size of *all* subdiagonal elements should be monitored. Whenever

$$|h_{i,i-1}| \leq \epsilon(|h_{i-1,i-1}| + |h_{i,i}|),$$

for some $i < n$, we set $|h_{i,i-1}|$ and continue to work on smaller subproblems. This is important for the efficiency of the algorithm, since the work is proportional to the square of the dimension of the Hessenberg matrix. An empirical observation is that on the average less than two QR iterations per eigenvalue are required.

When the shift is explicitly subtracted from the diagonal elements this may introduce large relative errors in any eigenvalue much smaller than the shift. We now describe an **implicit-shift** QR-algorithm, which avoids this type of error. This is based on Theorem9.6.1, which says that the matrix H_{k+1} in a QR iteration (9.7.24) *is essentially uniquely defined by the first column in Q_k, provided it is unreduced.*

In the following, for simplicity, we drop the iteration index and write (9.7.24) as

$$H - \tau I = QR, \qquad H' = RQ + \tau I. \tag{9.7.26}$$

To apply Theorem 9.6.1 to the QR algorithm we must find the first column q_1 in Q. From $H - \tau I = QR$ with R upper triangular it follows that $r_{11}q_1$ equals the first column in $H - \tau I$, which is

$$h_1 = (h_{11} - \tau, h_{21}, 0, \cdots, 0)^T.$$

If we choose a Givens rotation G_{12} so that $G_{12}^T h_1 = \pm\|h_1\|_2 e_1$, then $G_{12}e_1$ is proportional to h_1, and (take $n = 6$)

$$G_{12}^T H = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times, \end{pmatrix} \quad G_{12}^T H G_{12} = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

To preserve the Hessenberg form a rotation G_{23} is chosen to zero the element $+$,

$$G_{23}^T G_{12}^T H G_{12} G_{23} = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & + & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

We continue to chase the element $+$ down the diagonal, with rotations $G_{34}, \ldots, G_{n-1,n}$ until it disappears. We have then obtained a Hessenberg matrix $Q^T H Q$, where the first column in Q is $G_{12}G_{23}\cdots G_{n-1,n}e_1 = G_{12}e_1$. From Theorem 9.6.1 it follows that the computed Hessenberg matrix is indeed H'. Note that the information of the shift τ is contained in G_{12}, and the shift is not explicitly subtracted from the other diagonal elements. The cost of one QR iteration is $4n^2$ flops.

To avoid complex arithmetic when H is real one can *adopt the implicit-shift QR algorithm to compute the real Schur form* in Theorem 9.2.2, where R is quasi-triangular with 1×1 and 2×2 diagonal blocks. For real matrices this will save a factor of 2–4 over using complex arithmetic. Let τ_1 and τ_2 be the eigenvalues of the matrix C in (9.7.25), and consider two QR iterations with these shifts,

$$\begin{aligned} H - \tau_1 I &= Q_1 R_1, \quad H' = R_1 Q_1 + \tau_1 I, \\ H' - \tau_2 I &= Q_2 R_2, \quad H'' = R_2 Q_2 + \tau_2 I. \end{aligned}$$

We now show how to compute H'' directly from H using real arithmetic. We have $H'' = (Q_1 Q_2)^T H Q_1 Q_2$ and from Theorem 9.7.2

$$\begin{aligned} (Q_1 Q_2)(R_2 R_1) &= (H - \tau_1 I)(H - \tau_2 I) \\ &= H^2 - (\tau_1 + \tau_2)H + \tau_1\tau_2 I \equiv G, \end{aligned}$$

where $(\tau_1 + \tau_2)$ and $\tau_1\tau_2$ are real. By the uniqueness theorem (Q_1Q_2) is determined from its first column, which is proportional to the first column $g_1 = Ge_1 = (u, v, w, 0, \ldots, 0)^T$ of G. Taking out a factor $h_{21} \neq 0$ this can be written $g_1 = h_{21}(p, q, r, 0, \ldots, 0)^T$, where

$$\begin{aligned} p &= (h_{11}^2 - (\tau_1 + \tau_2)h_{11} + \tau_1\tau_2)/h_{21} + h_{12}, \\ q &= h_{11} + h_{22} - (\tau_1 + \tau_2), \qquad r = h_{32}. \end{aligned} \tag{9.7.27}$$

Note that we do not even have to compute τ_1 and τ_2, since we have $\tau_1 + \tau_2 = h_{n-1,n-1} + h_{n,n}$, and $\tau_1\tau_2 = \det(C)$. Substituting this into (9.7.27), and grouping terms to reduce roundoff errors, we get

$$\begin{aligned} p &= [(h_{nn} - h_{11})(h_{n-1,n-1} - h_{11}) - h_{n,n-1}h_{n-1,n}]/h_{21} + h_{12} \\ q &= (h_{22} - h_{11}) - (h_{nn} - h_{11}) - (h_{n-1,n-1} - h_{11}), \qquad r = h_{32}. \end{aligned}$$

The double QR step iteration can now be implemented by a chasing algorithm. We first choose rotations G_{23} and G_{12} so that $G_1^T g_1 = G_{12}^T G_{23}^T g_1 = \pm\|g_1\|_2 e_1$, and carry out a similarity transformation

$$G_1^T H = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}, \quad G_1^T H G_1 = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times \\ + & + & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

To preserve the Hessenberg form we then choose the transformation $G_2 = G_{34}G_{23}$ to zero out the two elements + in the first column. Then

$$G_2^T G_1^T H G_1 G_2 = \begin{pmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & + & \times & \times & \times & \times \\ & + & + & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

Note that this step is similar to the first step. The "bulge" of + elements has now shifted one step down along the diagonal, and we continue to chase these elements until they disappear below the last row. We have then completed one double step of the implicit QR algorithm.

Suppose the QR algorithm has converged to the final upper triangular matrix T. Then we have

$$P^T H P = T, \qquad P = Q_0 Q_1 Q_2 \cdots,$$

where Q_k is a product of Givens rotations, and P is the product of all the transformations used. The eigenvectors z_i, $i = 1, 2, \ldots, n$ of T satisfy $Tz_i = \lambda_i z_i$, $z_1 = e_1$, and z_i is a linear combination of $e_1, \ldots, e_i$. The nonzero components of z_i can then

be computed by back-substitution

$$z_{ii} = 1, \quad z_{ji} = -\Big(\sum_{k=j+1}^{i} t_{jk} z_{ki}\Big) \Big/ (\lambda_j - \lambda_i), \quad j = i-1, \ldots, 1. \tag{9.7.28}$$

The eigenvectors of H are then given by Pz_i, $i = 1, 2, \ldots, n$. Finally if H has been obtained by reducing a matrix A to Hessenberg form as described in Section 9.6.3, then the eigenvectors of A can be computed from

$$x_i = UPz_i, \quad i = 1, 2, \ldots, n, \qquad U^H A U = H. \tag{9.7.29}$$

When only a few selected eigenvectors are wanted, then a more efficient way is to compute these by using inverse iteration. However, if more than a quarter of the eigenvectors are required, it is better to use the procedure outlined above.

It must be remembered that the matrix A may be defective, in which case there is no complete set of eigenvectors. In practice it is very difficult to take this into account, since with any procedure that involves rounding errors one cannot demonstrate that a matrix is defective. Usually one therefore should attempt to find a complete set of eigenvectors. If the matrix is nearly defective this will often be evident, in that corresponding computed eigenvectors will be almost parallel.

If we do not want the eigenvectors, then it is not necessary to save the sequence of orthogonal transformations. It is even possible to avoid storing the rotations by performing the postmultiplications simultaneously with the premultiplications. For example, once we have formed $G_{23}G_{12}H_k$ the first two columns do not enter in the remaining steps and we can perform the postmultiplication with G_{12}^T. Hence we can alternately pre- and postmultiply; in the next step we compute $(G_{34}((G_{23}G_{12}H_k)G_{12}^T))G_{23}^T$, and so on.

From the real Schur form $Q^T A Q = T$ computed by the QR algorithm, we get information about some of the invariant subspaces of A. If

$$T = \begin{pmatrix} T_{11} & T_{12} \\ & T_{22} \end{pmatrix}, \quad Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix},$$

and $\lambda(T_{11}) \cap \lambda(T_{22}) = 0$, then Q_1 is an orthogonal basis for the unique invariant subspace associated with $\lambda(T_{11})$. However, this observation is useful only if we want the invariant subspace corresponding to a set of eigenvalues appearing at the top of the diagonal in T. Fortunately, it is easy to modify the real Schur decomposition so that an arbitrary set of eigenvalues are permuted to the top position. Clearly we can achieve this by performing a sequence of transformations, where in each step we interchange two nearby eigenvalues in the Schur form. Thus we only need to consider the 2×2 case,

$$Q^T A Q = T = \begin{pmatrix} \lambda_1 & h_{12} \\ 0 & \lambda_2 \end{pmatrix}, \quad \lambda_1 \neq \lambda_2.$$

To reverse the order of the eigenvalues we note that $Tx = \lambda_2 x$ where

$$x = \begin{pmatrix} h_{12} \\ \lambda_2 - \lambda_1 \end{pmatrix}.$$

Let G^T be a Givens rotation such that $G^T x = \gamma e_1$. Then $G^T T G(G^T x) = \lambda_2 G^T x$, i.e. $G^T x$ is an eigenvector of $\hat{T} = GTG^T$. It follows that $\hat{T}e_1 = \lambda_2 e_1$ and $\hat{T}$ must have the form

$$\hat{Q}^T A \hat{Q} = \hat{T} = \begin{pmatrix} \lambda_2 & \pm h_{12} \\ 0 & \lambda_1 \end{pmatrix},$$

where $\hat{Q} = QG$.

9.7.4 QR Algorithm for Symmetric Tridiagonal Matrices

By the methods described in Section 9.6 any Hermitian (real symmetric) matrix can by a unitary (orthogonal) similarity transformation be reduced into real, symmetric tridiagonal form

$$T = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \ddots & \ddots & \\ & & \ddots & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}. \tag{9.7.30}$$

A tridiagonal matrix T is called **unreduced** if all off-diagonal elements are nonzero, $\beta_i \neq 0$, $i = 2, \ldots, n$. Let T be unreduced and λ an eigenvalue of T. Then $\operatorname{rank}(T - \lambda I) = n - 1$ (the submatrix obtained by crossing out the first row and last column of $T - \lambda I$ has nonzero determinant, $\beta_2 \cdots \beta_n \neq 0$). Hence there is only one eigenvector corresponding to λ and since T is diagonalizable λ must have multiplicity one. *Thus all eigenvalues of an unreduced symmetric tridiagonal matrix are distinct.* In the following we can assume that T is unreduced, since otherwise it can be split up in smaller unreduced tridiagonal matrices.

The QR algorithm also preserves symmetry. Hence it follows that if T is symmetric tridiagonal, and

$$T - \tau I = QR, \qquad T' = RQ + \tau I, \tag{9.7.31}$$

then also $T' = Q^T T Q$ is symmetric tridiagonal.

From the Implicit Q Theorem (Theorem 9.6.1) we have the following result, which can be used to develop an implicit QR algorithm.

Theorem 9.7.3.

Let A be real symmetric, $Q = (q_1, \ldots, q_n)$ orthogonal, and $T = Q^T A Q$ an unreduced symmetric tridiagonal matrix. Then Q and T are essentially uniquely determined by the first column q_1 of Q.

Suppose we can find an orthogonal matrix Q with the same first column q_1 as in (9.7.31) such that $Q^T A Q$ is an unreduced tridiagonal matrix. Then by Theorem 9.7.3 it must be the result of one step of the QR algorithm with shift τ. Equating the first columns in $T - \tau I = QR$ it follows that $r_{11} q_1$ equals the first column t_1 in $T - \tau I$. In the implicit shift algorithm a Givens rotation G_{12} is chosen

so that

$$G_{12}^T t_1 = \pm\|t_1\|_2 e_1, \qquad t_1 = (\alpha_1 - \tau, \beta_2, 0, \cdots, 0)^T.$$

We now perform the similarity transformation $G_{12}^T T G_{12}$, which results in fill-in in positions (1,3) and (3,1), pictured below for $n = 5$:

$$G_{12}^T T = \begin{pmatrix} \times & \times & + & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}, \quad G_{12}^T T G_{12} = \begin{pmatrix} \times & \times & + & & & \\ \times & \times & \times & & & \\ + & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

To preserve the tridiagonal form a rotation G_{23} can be used to zero out the fill-in elements.

$$G_{23}^T G_{12}^T \; T \; G_{12} G_{23} = \begin{pmatrix} \times & \times & & & & \\ \times & \times & \times & + & & \\ & \times & \times & \times & & \\ & + & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{pmatrix}.$$

We continue to "chase the bulge" of + elements down the diagonal, with transformations $G_{34}, \ldots, G_{n-1,n}$ after which it disappears. We have then obtained a symmetric tridiagonal matrix $Q^T T Q$, where the first column in Q is $G_{12} G_{23} \cdots G_{n-1,n} e_1 = G_{12} e_1$. By Theorem 9.6.1 it follows that the result must be the matrix T' in (9.7.31).

There are several possible ways to choose the shift. Suppose that we are working with the submatrix ending with row r, and that the current elements of the two by two trailing matrix is

$$\begin{pmatrix} \alpha_{r-1} & \beta_r \\ \beta_r & \alpha_r \end{pmatrix}, \tag{9.7.32}$$

The Rayleigh quotient shift $\tau = \alpha_r$, gives the same result as Rayleigh Quotient Iteration starting with e_r. This leads to generic cubic convergence, but not guaranteed. In practice the **Wilkinson shift** has proved more efficient. This shift equals the eigenvalue of the submatrix (9.7.32), which is closest to α_r. A suitable formula for computing this shift is

$$\tau = \alpha_r - \beta_r^2 \Big/ \left(|d| + \operatorname{sign}(d) \sqrt{d^2 + \beta_r^2} \right), \quad d = (\alpha_{r-1} - \alpha_r)/2 \tag{9.7.33}$$

(cf. Algorithm (9.5.1)). A great advantage of the Wilkinson shift is that it gives guaranteed *global* convergence.[10] It can also be shown to give almost always *local cubic convergence*, although quadratic convergence might be possible.

[10] A proof is given in Parlett [38, Chapter 8].

Example 9.7.1. Consider an unreduced tridiagonal matrix of the form

$$T = \begin{pmatrix} \times & \times & 0 \\ \times & \times & \epsilon \\ 0 & \epsilon & t_{33} \end{pmatrix}.$$

Show, that with the shift $\tau = t_{33}$, the first step in the reduction to upper triangular form gives a matrix of the form

$$G_{12}(T - sI) = \begin{pmatrix} \times & \times & s_1\epsilon \\ 0 & a & c_1\epsilon \\ 0 & \epsilon & 0 \end{pmatrix}.$$

If we complete this step of the QR algorithm, $QR = T - \tau I$, the matrix $\hat{T} = RQ + \tau I$, has elements $\hat{t}_{32} = \hat{t}_{23} = -c_1\epsilon^3/(\epsilon^2 + a^2)$. This shows that if $\epsilon \ll$ the QR method tends to converge cubically.

As for the QR algorithm for unsymmetric matrices it is important to check for negligible subdiagonal elements using the criterion

$$|\beta_i| \leq \epsilon\,(|\alpha_{i-1}| + |\alpha_i|).$$

When this criterion is satisfied for some $i < n$, we set β_i equal to zero and the problem decouples. At any step we can partition the current matrix so that

$$T = \begin{pmatrix} T_{11} & & \\ & T_{22} & \\ & & D_3 \end{pmatrix},$$

where D_3 is diagonal and T_{22} is unreduced. The QR algorithm is then applied to T_{22}.

We will not give more details of the algorithm here. If full account of symmetry is taken then one QR iteration can be implemented in only $9n$ multiplications, $2n$ divisions, $n - 1$ square roots and $6n$ additions. By reorganizing the inner loop of the QR algorithm, it is possible to eliminate square roots and lower the operation count to about 4n multiplications, 3n divisions and 5n additions. This **rational QR algorithm** is the fastest way to get the eigenvalues alone, but does not directly yield the eigenvectors.

The Wilkinson shift may not give the eigenvalues in monotonic order. If some of the smallest or largest eigenvalues are wanted, then it is usually recommended to use Wilkinson shifts anyway and risk finding a few extra eigenvalues. To check if all wanted eigenvalues have been found one can use spectrum slicing, see Section 9.6.5. For a detailed discussion of variants of the symmetric tridiagonal QR algorithm, see Parlett [38].

If T has been obtained by reducing a Hermitian matrix to real symmetric tridiagonal form, $U^H AU = T$, then the eigenvectors are given by

$$x_i = UPe_i, \quad i = 1, 2, \ldots, n, \tag{9.7.34}$$

where $P = Q_0 Q_1 Q_2 \cdots$ is the product of all transformations in the QR algorithm. Note that the eigenvector matrix $X = UP$ will by definition be orthogonal.

If eigenvectors are to be computed, the cost of a QR iteration goes up to $4n^2$ flops and the overall cost to $O(n^3)$. To reduce the number of QR iterations where we accumulate transformations, we can first compute the eigenvalues *without* accumulating the product of the transformations. We then perform the QR algorithm again, now shifting with the computed eigenvalues, the **perfect shifts**, convergence occurs in one iteration. This may reduce the cost of computing eigenvectors by about 40%. As in the unsymmetric case, if fewer than a quarter of the eigenvectors are wanted, then inverse iteration should be used instead. The drawback of this approach, however, is the difficulty of getting orthogonal eigenvectors to clustered eigenvalues.

For symmetric tridiagonal matrices one often uses the QL algorithm instead of the QR algorithm. We showed in Section 9.7.1 that the QL algorithm is just the QR algorithm on JAJ, where J is the permutation matrix that reverses the elements in a vector. If A is tridiagonal then JAJ is tridiagonal with the diagonal elements in reverse order.

In the implicit QL algorithm one chooses the shift from the top of A and chases the bulge from bottom to top. The reason for preferring the QL algorithm is simply that in practice it is often the case that the tridiagonal matrix is graded with the large elements at the bottom. Since for reasons of stability the small eigenvalues should be determined first the QL algorithm is preferable in this case. For matrices graded in the other direction the QR algorithm should be used, or rows and columns reversed before the QL algorithm is applied.

9.7.5 QR-SVD algorithms for Bidiagonal Matrices

For the computation of the SVD of a matrix $A \in \mathbf{R}^{m\times n}$ it is usually advisable to first perform a QR decomposition with column pivoting of A

$$A\Pi = Q \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{9.7.35}$$

(We assume in the following that $m \geq n$. This is no restriction since otherwise we can consider A^T.) Let let $R = U_R \Sigma V^T$ be the SVD of R. Then it follows that

$$A = U\Sigma V^T, \quad U = Q \begin{pmatrix} U_R \\ 0 \end{pmatrix}. \tag{9.7.36}$$

Clearly the singular values and the right singular vectors of $A\Pi$ and R are the same and the first n left singular vectors of A are easily obtained from those of R.

Starting with $R_1 = R$, a sequence of upper triangular matrices R_k, $k = 1, 2, \ldots$. In step k the QR factorization of a the *lower* triangular matrix is computed

$$R_k^T = Q_{k+1} R_{k+1}, \tag{9.7.37}$$

In the next step R_{k+1} is transposed and the process repeated. As we now show This iteration is related to the basic unshifted QR algorithm for $R^T R$ *and* $R^T R$.

Using (9.7.37) we observe that

$$R_k^T R_k = Q_{k+1}(R_{k+1}R_k)$$

is the QR factorization of $R_k^T R_k$. Forming the product in reverse order gives

$$\begin{aligned}(R_{k+1}R_k)Q_{k+1} &= R_{k+1}R_{k+1}^T Q_{k+1}^T Q_{k+1} = R_{k+1}R_{k+1}^T \\ &= R_{k+2}^T Q_{k+2}^T Q_{k+2} R_{k+2} = R_{k+2}^T R_{k+2}.\end{aligned}$$

Hence two successive iterations of (9.7.37) are equivalent to one iteration of the basic QR algorithm for $R^T R$. Moreover this is achieved without forming $R^T R$, which is essential to avoid loss of accuracy.

Using the orthogonality of Q_{k+1} it follows from (9.7.37) that $R_{k+1} = Q_{k+1}^T R_k^T$, and hence

$$R_{k+1}^T R_{k+1} = R_k(Q_{k+1}Q_{k+1}^T)R_k^T = R_k R_k^T.$$

Further we have

$$R_{k+2}R_{k+2}^T = R_{k+2}R_{k+1}Q_{k+2} = Q_{k+2}^T(R_k R_k^T)Q_{k+2}. \tag{9.7.38}$$

which shows that we are simultaneously performing an iteration on $R_k R_k^T$, again without explicitly forming this matrix.

One iteration of (9.7.37) is equivalent to one iteration of the Cholesky LR algorithm applied to $B_k = R_k R_k^T$. This follows since B_k has the Cholesky factorization $B_k = R_{k+1}^T R_{k+1}$ and multiplication of these factors in reverse order gives $B_{k+1} = R_{k+1}R_{k+1}^T$. (Recall that for a symmetric, positive definite matrix two steps of the LR algorithm is equivalent to one step of the QR algorithm.)

The convergence of this algorithm is enhanced provided the QR factorization of A in the first step is performed using column pivoting. It has been shown that then already the diagonal elements of R_1 often are surprisingly good approximations to the singular values of A.

For the QR-SVD algorithm to be efficient it is necessary to initially reduce A to a compact form that is preserved during the QR iterations and to introduce shifts. The proper compact form here is a bidiagonal form B. It was described in Section 8.6.6 how any matrix $A \in \mathbf{R}^{m\times n}$ can be reduced to upper bidiagonal form. Performing this reduction on R we have

$$Q_B^T R P_B = B = \begin{pmatrix} q_1 & e_2 & & & \\ & q_2 & e_3 & & \\ & & \ddots & \ddots & \\ & & & q_{n-1} & e_n \\ & & & & q_n \end{pmatrix}. \tag{9.7.39}$$

with orthogonal transformations from left and right. Using a sequence of Householder transformations

$$Q_B = Q_1 \cdots Q_n \in \mathbf{R}^{n\times n}, \qquad P_B = P_1 \cdots P_{n-2} \in \mathbf{R}^{n\times n}.$$

the reduction can be carried out in $\frac{4}{3}n^3$ flops. Note that also a complex matrix A can be reduced to *real* bidiagonal form using complex Householder transformations, see Section 9.1.2. The singular values of B equal those of A and the left and right singular vectors can be constructed from those of B.

We first notice that if in (9.7.39) $e_i = 0$, then the matrix B breaks into two upper bidiagonal matrices, for which the singular values can be computed independently. If $q_i = 0$, then B has a singular value equal to zero. Applying a sequence of Givens rotations from the left, $G_{i,i+1}, G_{i,i+2}, \ldots, G_{i,n}$ the ith row be zeroed out, and again the matrix breaks up into two parts. Hence we may without loss of generality assume that none of the elements q_1, q_i, e_i, $i = 2, \ldots, n$ are zero. This assumption implies that the matrix B^TB has nondiagonal elements $\alpha_{i+1} = q_i e_{i+1} \neq 0$, and hence is unreduced. It follows that all eigenvalues of B^TB are positive and distinct, and we have $\sigma_1 > \cdots > \sigma_n > 0$.

Since shifts are essential for achieving rapid convergence and deflation we now look into alternative ways of implementing the QR-SVD algorithm.

We first proceed by forming the symmetric matrix

$$C = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix} \in \mathbf{R}^{2n\times 2n}. \tag{9.7.40}$$

whose eigenvalues are $\pm\sigma_i$, $i = 1, \ldots, n$. After reordering rows and columns by an odd/even permutation C becomes symmetric tridiagonal matrix with zeros on the main diagonal. Hence

$$T = P^TCP = \begin{pmatrix} 0 & q_1 & & & & \\ q_1 & 0 & e_2 & & & \\ & e_2 & 0 & q_2 & & \\ & & q_2 & 0 & \ddots & \\ & & & \ddots & \ddots & q_n \\ & & & & q_n & 0 \end{pmatrix} \tag{9.7.41}$$

where P is the permutation matrix whose columns are those of the identity in the order $(n+1, 1, n+2, 2, \ldots, 2n, n)$. Hence the QR algorithm, the divide and conquer algorithm, and spectrum slicing (see Problem 6) are all applicable to this special tridiagonal matrix to compute the singular values of B. A disadvantage of this approach is that the dimension is essentially doubled.

A closer inspection of the QR algorithm applied to T reveals it to be equivalent to an algorithm where the iterations are carried out directly on B. It is also equivalent to an *implicit* version of the QR algorithm applied to the symmetric tridiagonal matrix $T = B^TB$.

We now consider the application of the implicit shift QR algorithm to B^TB. Since forming B^TB would lead to a severe loss of accuracy in the small singular values it is essential to work directly with the matrix B. The Wilkinson shift τ can be determined as the smallest eigenvalue of the lower right 2×2 submatrix in BB^T, or equivalently as the square of the smallest singular value of the 2×2 upper

triangular submatrix in (9.7.39)

$$\begin{pmatrix} q_{n-1} & e_n \\ 0 & q_n \end{pmatrix}.$$

In the implicit shift QR algorithm for B^TB we first determine a Givens rotation $T_1 = G_{12}$ so that

$$G_{12}^T t_1 = \pm\|t_1\|_2 e_1, \qquad t_1 = (q_1^2 - \tau, q_1 e_2, 0, \ldots, 0)^T, \tag{9.7.42}$$

where t_1 is the first column in $B^TB - \tau I$ and τ is the shift. Suppose we next apply a sequence of Givens transformations such that

$$T_{n-1}^T \cdots T_2^T T_1^T B^T B T_1 T_2 \cdots T_{n-1}$$

is tridiagonal, but we wish to avoid doing this explicitly. Let us start by applying the transformation T_1 to B. Then we get (take $n = 5$),

$$BT_1 = \begin{array}{c} \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \end{array} \begin{pmatrix} \times & \times & & & \\ + & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix}.$$

If we now premultiply by a Givens rotation $S_1^T = R_{12}$ to zero out the $+$ element, this creates a new nonzero element in the $(1,3)$ position; To preserve the bidiagonal form we then choose the transformation $T_2 = R_{23}$ to zero out the element $+$:

$$S_1^T BT_1 = \begin{array}{c} \\ \rightarrow \\ \rightarrow \\ \\ \\ \\ \end{array} \begin{pmatrix} \times & \times & + & & \\ \oplus & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix}, \quad S_1^T BT_1T_2 = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array} \begin{pmatrix} & \downarrow & \downarrow & & \\ \times & \times & \oplus & & \\ & \times & \times & & \\ & + & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix}.$$

We can now continue to chase the element $+$ down, with transformations alternately from the right and left until we get a new bidiagonal matrix

$$\hat{B} = (S_{n-1}^T \cdots S_1^T) B (T_1 \cdots T_{n-1}) = U^T B P.$$

But then the matrix

$$\hat{T} = \hat{B}^T \hat{B} = P^T B^T U U^T B P = P^T T P$$

is tridiagonal, where the first column of P equals the first column of T_1. Hence if $\hat{T}$ is unreduced it must be the result of one QR iteration on $T = B^TB$ with shift equal to τ.

The subdiagonal entries of T equal $q_i e_{i+1}$, $i = 1, \ldots, n-1$. If some element e_{i+1} is zero, then the bidiagonal matrix splits into two smaller bidiagonal matrices

$$B = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}.$$

If $q_i = 0$, then we can zero the ith row by premultiplication by a sequence Givens transformations $R_{i,i+1}, \ldots, R_{i,n}$, and the matrix then splits as above. In practice two convergence criteria are used. After each QR step if

$$|e_{i+1}| \leq 0.5u(|q_i| + |q_{i+1}|),$$

where u is the unit roundoff, we set $e_{i+1} = 0$. We then find the smallest p and the largest q such that B splits into quadratic subblocks

$$\begin{pmatrix} B_1 & 0 & 0 \\ 0 & B_2 & 0 \\ 0 & 0 & B_3 \end{pmatrix},$$

of dimensions $p, n-p-q$ and, q where B_3 is diagonal and B_2 has a nonzero subdiagonal. Second, if diagonal elements in B_2 satisfy

$$|q_i| \leq 0.5u(|e_i| + |e_{i+1}|),$$

set $q_i = 0$, zero the superdiagonal element in the same row, and repartition B. Otherwise continue the QR algorithm on B_2.

A justification for these tests is that roundoff in a rotation could make the matrix indistinguishable from one with a q_i or e_{i+1} equal to zero. Also, the error introduced by the tests is not larger than some constant times $u\|B\|_2$.

The implicit QR-SVD algorithm can be shown to be backward stable. This essentially follows from the fact that we have only applied a sequence of orthogonal transformations to A. Hence the computed singular values $\bar{\Sigma} = \operatorname{diag}(\bar{\sigma}_k)$ are the exact singular values of a nearby matrix $A + E$, where $\|E\|_2 \leq c(m,n) \cdot u\sigma_1$. Here $c(m,n)$ is a constant depending on m and n and u the unit roundoff. From Theorem 7.3.4

$$|\bar{\sigma}_k - \sigma_k| \leq c(m,n) \cdot u\sigma_1.$$

Thus, if A is nearly rank deficient, this will always be revealed by the computed singular values. Note, however, that the smaller singular values may not be computed with high relative accuracy.

When all the superdiagonal elements in B have converged to zero we have $Q_S^T B T_S = \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$. Hence

$$U^T A V = \begin{pmatrix} \Sigma \\ 0 \end{pmatrix}, \quad U = Q_B \operatorname{diag}(Q_S, I_{m-n}), \quad V = T_B T_S \tag{9.7.43}$$

is the singular value decomposition of A. Usually less than $2n$ iterations are needed in the second phase. One QR iteration requires $14n$ multiplications and $2n$ calls to givrot. Accumulating the rotations into U requires $6mn$ flops. Accumulating the

Table 9.7.1. *Comparison of multiplications for SVD algorithms.*

Required	Golub–Reinsch SVD	Chan SVD
Σ, U_1, V	$(3+C)mn^2 + \frac{11}{3}n^3$	$3mn^2 + 2(C+1)n^3$
Σ, U_1	$(3+C)mn^2 - n^3$	$3mn^2 + (C+4/3)n^3$
Σ, V	$2mn^2 + Cn^3$	$mn^2 + (C+5/3)n^3$
Σ	$2mn^2 - 2n^3/3$	$mn^2 + n^3$

rotations into V requires $6n^2$ flops. If singular vectors are desired, the cost of a QR iteration goes up to $4n^2$ flops and the overall cost to $O(n^3)$. See Table 9.7.5 for a comparison of flop counts for different variants.

To reduce the number of QR iterations where we accumulate transformations we can first compute the singular values without accumulating vectors. If we then choose shifts based on the computed singular values, the *perfect shifts*, convergence occurs in one iteration. This may reduce the cost about 40%. If fewer than 25% of the singular vectors are wanted, then inverse iteration should be used instead. The drawback of this approach is the difficulty of getting orthogonal singular vectors to clustered singular values.

An important implementation issue is that the bidiagonal matrix is often graded, i.e., the elements may be large at one end and small at the other. For example, if in the Chan-SVD column pivoting is used in the initial QR decomposition, then the matrix is usually graded from large at upper left to small at lower right as illustrated below

$$\begin{pmatrix} 1 & 10^{-1} & & \\ & 10^{-2} & 10^{-3} & \\ & & 10^{-4} & 10^{-5} \\ & & & 10^{-6} \end{pmatrix}. \tag{9.7.44}$$

From the following perturbation result it follows that it should be possible to compute all singular values of a bidiagonal matrix to *full relative precision independent of their magnitudes.*

Theorem 9.7.4. (Demmel and Kahan [9, 1990])

Let $B \in \mathbf{R}^{n\times n}$ be a bidiagonal matrix with singular values $\sigma_1 \geq \cdots \geq \sigma_n$. Let $|\delta B| \leq \omega |B|$, and let $\bar{\sigma}_1 \geq \cdots \geq \bar{\sigma}_n$ be the singular values of $\bar{B} = B + \delta B$. Then if $\eta = (2n-1)\omega < 1$,

$$|\bar{\sigma}_i - \sigma_i| \leq \frac{\eta}{1-\eta}|\sigma_i|, \tag{9.7.45}$$

$$\max\{\sin\theta(u_i, \tilde{u}_i), \sin\theta(v_i, \tilde{v}_i)\} \leq \frac{\sqrt{2}\eta(1+\eta)}{\mathrm{relgap}_i - \eta}, \tag{9.7.46}$$

$i = 1, \ldots, n$, where the **relative gap** *between singular values is*

$$\mathrm{relgap}_i = \min_{j\neq i} \frac{|\sigma_i - \sigma_j|}{\sigma_i + \sigma_j}. \tag{9.7.47}$$

The QR algorithm as described above tries to converge to the singular values from smallest to largest, and "chases the bulge" from top to bottom. Convergence will then be fast. However, if B is graded the opposite way then the QR algorithm may require many more steps. To avoid this the rows and columns of B could in this case be reversed before the QR algorithm is applied. Alternatively many algorithms check for the direction of grading. Note that the matrix may break up into diagonal blocks which are graded in different ways.

To compute small singular values of a bidiagonal matrix accurately one can use the unshifted QR-SVD algorithm given by (9.7.37). which uses the iteration

$$B_k^T = Q_{k+1} B_{k+1}, \quad k = 0, 1, 2, \ldots. \tag{9.7.48}$$

In each step the lower bidiagonal matrix B_k^T is transformed into an upper bidiagonal matrix B_{k+1}.

$$Q_1^T B = \begin{matrix} \rightarrow \\ \rightarrow \\ \\ \\ \end{matrix} \begin{pmatrix} \times & + & & & \\ \otimes & \times & & & \\ & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \end{pmatrix}, \quad Q_2 Q_1^T B = \begin{matrix} \\ \rightarrow \\ \rightarrow \\ \\ \end{matrix} \begin{pmatrix} \times & \times & & & \\ & \times & + & & \\ & \otimes & \times & & \\ & & \times & \times & \\ & & & \times & \times \end{pmatrix},$$

etc. Each iteration in (9.7.48) can be performed with a sequence of $n-1$ Givens rotations at a cost of only $2n$ multplications and $n-1$ calls to givrot. Two steps of the iteration is equivalent to one step of the zero shift QR algorithm. (Recall that one step of the QR algorithm with nonzero shifts, requires $12n$ multiplications and $4n$ additions.) The zero shift algorithm is very simple and uses no subtractions, Hence each entry of the transformed matrix is computed to high *relative* accuracy.

Algorithm 9.7.1 THE ZERO SHIFT QR ALGORITHM
The algorithm performs p steps of the zero shift QR algorithm on the bidiagonal matrix B in (9.7.39):

```
for k = 1 : 2p
  for i = 1 : n − 1
    [c, s, r] = givrot(q_i, e_{i+1});
    q_i = r;   q_{i+1} = q_{i+1} * c;
    e_{i+1} = q_{i+1} * s;
  end
end
```

If two successive steps of (9.7.48) are interleaved we get the **zero shift QR algorithm**, the implementation of which has been studied in depth by Demmel and Kahan [9]. To give full accuracy for the smaller sigingular values the convergence tests used for standard shifted QR-SVD algorithm must be modified. This is a non-trivial task, for which we refer to the original paper.

9.7.6 Singular Values by Spectrum Slicing

An algorithm for computing singular values can be developed by applying Algorithm 9.6.6 for spectrum slicing to the special symmetric tridiagonal matrix T in (9.7.41). Taking advantage of the zero diagonal this algorithm simplifies and one slice requires only of the order $2n$ flops. Given the elements $q_1, \ldots, q_n$ and $e_2, \ldots, e_n$ of T in (9.7.41), the following algorithm generates the number π of singular values of T greater than a given value $\sigma > 0$.

Algorithm 9.7.2
Singular Values by Spectrum Slicing Let T be the tridiagonal matrix (9.6.9). Then the number π of eigenvalues greater than a given number σ is generated by the following algorithm:

$d_1 := -\sigma;$
$flip := -1;$
$\pi :=$ **if** $d_1 > 0$ **then** 1 **else** 0;
for $k = 2 : 2n$
$flip := -flip;$
if $flip = 1$ **then** $\beta = q_{k/2}$
  else $\beta = e_{(k+1)/2};$
end
  $d_k := -\beta(\beta/d_{k-1}) - \tau;$
  if $|d_k| < \sqrt{\omega}$ **then** $d_k := \sqrt{\omega};$
  if $d_k > 0$ **then** $\pi := \pi + 1;$
end

Spectrum slicing algorithm for computing singular values has been analyzed by Fernando [11]. and shown to provide high relative accuracy also for tiny singular values.

Review Questions

1. What is meant by a graded matrix, and what precautions need to be taken when transforming such a matrix to condensed form?
2. For a certain class of symmetric matrices small eigenvalues are determined with a very small error compared to $\|A\|_F$. Which?
3. If one step of the QR algorithm is performed on A with a shift τ equal to an eigenvalue of A, what can you say about the result? Describe how the shift usually is chosen in the QR algorithm applied to a real symmetric tridiagonal matrix.
4. What are the advantages of the implicit shift version of the QR algorithm for a real Hessenberg matrix H?

5. Suppose the eigenvalues to a Hessenberg matrix have been computed using the QR algorithm. How are the eigenvectors best computed (a) if all eigenvectors are needed; (b) if only a few eigenvectors are needed.

6. (a) Show that the symmetry of a matrix is preserved during the QR algorithm. What about normality?

(b) Show that the Hessenberg form is preserved during the QR algorithm.

7. What condensed form is usually chosen for the singular value decomposition? What kind of transformations are used for bringing the matrix to condensed form? How are the singular values computed for the condensed form?

Problems

1. Perform a QR step without shift on the matrix

$$A = \begin{pmatrix} \cos\theta & \sin\theta \\ \sin\theta & 0 \end{pmatrix}$$

and show that the nondiagonal elements are reduced to $-\sin^3\theta$.

2. Let T be the tridiagonal matrix in (9.7.30), and suppose a QR step using the shift $\tau = \alpha_n$ is carried out,

$$T - \alpha_n I = QR, \qquad \tilde{T} = RQ + \alpha_n I.$$

Generalize the result from Problem 2, and show that if $\gamma = \min_i |\lambda_i(T_{n-1}) - \alpha_n| > 0$, then $|\tilde{\beta}_n| \le |\beta_n|^3/\gamma^2$.

3. Show that a complex matrix A can be reduced to *real* bidiagonal form using a sequence of unitary Householder transformations, see (9.6.2)–(9.6.3)

4. Let C be the matrix in (9.7.40) and P the permutation matrix whose columns are those of the identity matrix in the order $(n+1, 1, n+2, 2, \ldots, 2n, n)$. Show that the matrix $P^T C P$ becomes a tridiagonal matrix T of the form in (9.7.41).

5. To compute the SVD of a matrix $A \in \mathbf{R}^{m\times 2}$ we can first reduce A to upper triangular form by a QR decomposition

$$A = (a_1, a_2) = (q_1,\ q_2)\begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad R = \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix}.$$

Then, as outlined in Golub and Van Loan [21, Problem 8.5.1], a Givens rotation G can be determined such that $B = GRG^T$ is symmetric. Finally, B can be diagonalized by a Jacobi transformation. Derive the details of this algorithm!

6. (a) Let σ_i be the singular values of the matrix

$$M = \begin{pmatrix} z_1 & & & \\ z_2 & d_2 & & \\ \vdots & & \ddots & \\ z_n & & & d_n \end{pmatrix} \in \mathbf{R}^{n\times n},$$

where the elements d_i are distinct. Show the interlacing property

$$0 < \sigma_1 < d_2 < \cdots < d_n < \sigma_n < d_n + \|z\|_2.$$

(b) Show that σ_i satisfies the secular equation

$$f(\sigma) = 1 + \sum_{k=1}^{n} \frac{z_k^2}{d_k^2 - \sigma^2} = 0.$$

Give expressions for the right and left singular vectors of M.

Hint: See Lemma 9.6.2.

7. Modify Algorithm 9.7.1 for the zero shift QR-SVD algorithm so that the two loops are merged into one.

9.8 Subspace Methods for Large Eigenvalue Problems

In many applications eigenvalue problems arise involving matrices so large that they cannot be conveniently treated by the methods described so far. For such problems, it is not reasonable to ask for a complete set of eigenvalues and eigenvectors, and usually only some extreme eigenvalues (often at one end of the spectrum) are required. In the 1980's typical values could be to compute 10 eigenpairs of a matrix of order 10,000. In the late 1990's problems are solved where 1,000 eigenpairs are computed for matrices of order 1,000,000!

We concentrate on the symmetric eigenvalue problem since fortunately many of the very large eigenvalue problems that arise are symmetric. We first consider the general problem of obtaining approximations from a subspace of $\mathbf{R}^n$. We then survey the two main classes of methods developed for large or very large eigenvalue problems.

9.8.1 The Rayleigh–Ritz Procedure

Let $\mathcal{S}$ be the subspace of $\mathbf{R}^n$ spanned by the columns of a given matrix $S = (s_1, \ldots, s_m) \in \mathbf{R}^{n\times m}$ (usually $m \ll n$). We consider here the problem of finding the best set of approximate eigenvectors in $\mathcal{S}$ to eigenvectors of a Hermitian matrix A. The following generalization of the Rayleigh quotient is the essential tool needed.

Theorem 9.8.1.

Let A be Hermitian and $Q \in \mathbf{R}^{n\times p}$ be orthonormal, $Q^HQ = I_p$. Then the residual norm $\|AQ - QC\|_2$ is minimized for $C = M$ where

$$M = \rho(Q) = Q^HAQ \tag{9.8.1}$$

is the corresponding Rayleigh quotient matrix. Further, if $\theta_1, \ldots, \theta_p$ are the eigenvalues of M, there are p eigenvalues $\lambda_{i1}, \ldots, \lambda_{ip}$ of A, such that

$$|\lambda_{ij} - \theta_j| \le \|AQ - QM\|_2, \quad j = 1, \ldots, p. \tag{9.8.2}$$

Proof. See Parlett [38, Section 11-5]. □

We can now outline the complete procedure:

Algorithm 9.8.1
The Rayleigh–Ritz procedure

1. Determine an orthonormal matrix $Q = (q_1, \ldots, q_m)$ such that $\mathcal{R}(Q) = \mathcal{S}$.

2. Form the matrix $B = AQ = (Aq_1, \ldots, Aq_m)$ and the generalized Rayleigh quotient matrix
$$M = Q^H(AQ) \in \mathbf{R}^{m\times m}. \tag{9.8.3}$$

3. Compute the $p \leq m$ eigenpairs of the Hermitian matrix M which are of interest
$$Mz_i = \theta_1 z_i, \qquad i = 1, \ldots, p. \tag{9.8.4}$$
The eigenvectors can be chosen such that $Z = (z_1, \ldots, z_m)$ is a unitary matrix. The eigenvalues θ_i are the **Ritz values**, and the vectors $y_i = Qz_i$ the **Ritz vectors**.

4. Compute the residual matrix $R = (r_1, \ldots, r_p)$, where
$$r_i = Ay_i - y_i\theta_i = (AQ)z_i - y_i\theta_i. \tag{9.8.5}$$
Then each interval
$$\big[\theta_i - \|r_i\|_2, \theta_i + \|r_i\|_2\big], \qquad i = 1, \ldots, p, \tag{9.8.6}$$
contains an eigenvalue λ_i of A.

The pairs (θ_i, y_i), $i = 1, \ldots, p$ are the best approximate eigenpairs of A which can be derived from the space $\mathcal{S}$. If some of the intervals in (9.8.6) overlap, we cannot be sure to have approximations to p eigenvalues of A. However, there are always p eigenvalues in the intervals defined by (9.8.2).

We can get error bounds for the approximate eigenspaces from an elegant generalization of Theorem 9.3.15. We first need to define the **gap** of the spectrum of A with respect to a given set of approximate eigenvalues.

Definition 9.8.2.
Let $\lambda(A) = \{\lambda_1, \ldots, \lambda_n\}$ be eigenvalues of a Hermitian matrix A. For the set $\rho = \{\theta_1, \ldots, \theta_p\}$, let $s_\rho = \{\lambda_{i_1}, \ldots, \lambda_{i_p}\}$ be a subset of $\lambda(A)$ minimizing $\max_j |\theta_j - \lambda_{i_j}|$. Then we define

$$\operatorname{gap}(\rho) = \min_{\lambda\in\lambda(A)} |\lambda - \theta_i|, \quad \lambda \notin s_\rho, \quad \theta_i \in \rho. \tag{9.8.7}$$

Theorem 9.8.3.

Let $Q \in \mathbf{R}^{n\times p}$ be orthonormal and A a Hermitian matrix. Let $\{\theta_1, \ldots, \theta_p\}$ be the eigenvalues of $H = \rho(Q) = Q^H AQ$, and let $s_r = \{\lambda_{i_1}, \ldots, \lambda_{i_p}\}$ be a subset of eigenvalues of A such that $\max_j |\theta_j - \lambda_{i_j}|$ is minimized. If $\mathcal{Z}$ is the invariant subspace of A corresponding to s_r, then

$$\theta(\mathcal{Q}, \mathcal{Z}) \leq \|AQ - QH\|_2/\text{gap}\,(\rho). \tag{9.8.8}$$

where $\sin\theta(\mathcal{Q}, \mathcal{Z})$ is the largest angle between the subspaces $\mathcal{Q}$ and $\mathcal{Z}$.

9.8.2 Subspace Iteration for Hermitian Matrices

In Section 9.4.4 subspace iteration, or orthogonal iteration, was introduced as a block version of the power method. Subspace iteration has long been one of the most important methods for solving large sparse eigenvalue problems. In particular it has been used much in structural engineering, and developed to a high standard of refinement.

In simple subspace iteration we start with an initial matrix $Q_0 \in \mathbf{R}^{n\times p}$ ($1 < p \ll n$) with orthogonal columns. From this a sequence of matrices $\{Q_k\}$ are computed from

$$Z_k = AQ_{k-1}, \qquad Q_k R_k = Z_k, \qquad k = 1, 2, \ldots, \tag{9.8.9}$$

where $Q_k R_k$ is the QR decomposition of the matrix Z_k. There is no need for the matrix A to be known explicitly; only an algorithm (subroutine) for computing the matrix-vector product Aq for an arbitrary vector q is required. This iteration (9.8.9) generates a sequence of subspaces $\mathcal{S}_k = \mathcal{R}(A^k Q_0) = \mathcal{R}(Q_k)$, and we seek approximate eigenvectors of A in these subspaces. It can be shown (see Section 9.4.4) that if A has p dominant eigenvalues $\lambda_1, \cdots, \lambda_p$, i.e.,

$$|\lambda_1| \geq \cdots \geq |\lambda_p| > |\lambda_{p+1}| \geq \cdots \geq |\lambda_n|$$

then the subspaces $\mathcal{S}_k$, $k = 0, 1, 2, \ldots$ converge almost always to the corresponding dominating invariant subspace.The convergence is linear with rate $|\lambda_{p+1}/\lambda_p|$.

For the individual eigenvalues $\lambda_i > \lambda_{i+1}$, $i \leq p$, it holds that

$$|r_{ii}^{(k)} - \lambda_i| = O(|\lambda_{i+1}/\lambda_i|^k), \qquad i = 1, \ldots, p.$$

where $r_{ii}^{(k)}$ are the diagonal elements in R_k. This rate of convergence is often unacceptably slow. We can improve this by including the Rayleigh–Ritz procedure in orthogonal iteration. For the real symmetric (Hermitian) case this leads to the improved algorithm below.

Algorithm 9.8.2

Orthogonal Iteration, Hermitian Case.

With $Q_0 \in \mathbf{R}^{n\times p}$ compute for $k = 1, 2, \ldots$ a sequence of matrices Q_k as follows:

1. Compute $Z_k = AQ_{k-1}$;
2. Compute the QR decomposition $Z_k = \bar{Q}_k R_k$;
3. Form the (matrix) Rayleigh quotient $B_k = \bar{Q}_k^T (A\bar{Q}_k)$;
4. Compute eigenvalue decomposition $B_k = U_k \Theta_k U_k^T$;
5. Compute the matrix of Ritz vectors $Q_k = \bar{Q}_k U_k$.

It can be shown that

$$|\theta_i^{(k)} - \lambda_i| = \mathrm{O}(|\lambda_{p+1}/\lambda_i|^k), \qquad \Theta_k = \mathrm{diag}\,(\theta_1^{(k)}, \ldots, \theta_p^{(k)}),$$

which is a much more favorable rate of convergence than without the Rayleigh–Ritz procedure. The columns of Q_k are the Ritz vectors, and they will converge to the corresponding eigenvectors of A.

Example 9.8.1.

Let A have the eigenvalues $\lambda_1 = 100$, $\lambda_2 = 99$, $\lambda_3 = 98$ $\lambda_4 = 10$, and $\lambda_5 = 5$. With $p = 3$ the asymptotic convergence ratios for the jth eigenvalue with and without Rayleigh–Ritz acceleration are:

j	without R-R	with R-R
1	0.99	0.1
2	0.99	0.101
3	0.102	0.102

The work in step 1 of Algorithm 9.8.2 consists of p matrix times vector operations with the matrix A. If the modified Gram-Schmidt method is used step 2 requires $p(p+1)n$ flops. To form the Rayleigh quotient matrix requires a further p matrix times vector multiplications and $p(p+1)n/2$ flops, taking the symmetry of B_k into account. Finally steps 4 and 5 take about $5p^3$ and $p^2 n$ flops, respectively.

Note that the same subspace $\mathcal{S}_k$ is generated by k consecutive steps of 1, as with the complete Algorithm 9.8.2. Therefore the rather costly orthogonalization and Rayleigh–Ritz acceleration need not be carried out at every step. However, to be able to check convergence to the individual eigenvalues we need the Rayleigh–Ritz approximations. If we then form the residual vectors

$$r_i = Aq_i^{(k)} - q_i^{(k)}\theta_i = (AQ_k)u_i^{(k)} - q_i^{(k)}\theta_i. \tag{9.8.10}$$

and compute $\|r_i\|_2$ each interval $[\theta_i - \|r_i\|_2, \theta_i + \|r_i\|_2]$ will contain an eigenvalue of A. Sophisticated versions of subspace iteration have been developed. A highlight is the Contribution II/9 by Rutishauser in [40].

Algorithm 9.8.2 can be generalized to nonsymmetric matrices, by substituting in step 4 the Schur decomposition

$$B_k = U_k S_k U_k^T,$$

where S_k is upper triangular. The vectors q_i then converge to the Schur vector u_i of A.

If interior eigenvalues are wanted then we can consider the **spectral transformation** (see Section 9.4.2)

$$\hat{A} = (A - \mu I)^{-1}.$$

The eigenvalues of $\hat{A}$ and A are related through $\hat{\lambda}_i = 1/(\lambda_i - \mu)$. Hence, the eigenvalues λ in a neighborhood of μ will correspond to outer eigenvalues of $\hat{A}$, and can be determined by applying subspace iteration to $\hat{A}$. To perform the multiplication $\hat{A}q$ we need to be able to solve systems of equations of the form

$$(A - \mu I)p = q. \tag{9.8.11}$$

This can be done, e.g., by first computing an LU factorization of $A - \mu I$ or by an iterative method.

9.8.3 Krylov Subspaces

Of great importance for iterative methods are the subspaces of the form

$$\mathcal{K}_m(v, A) = \operatorname{span}\big(v, Av, \ldots, A^{m-1}v\big), \tag{9.8.12}$$

generated by a matrix A and a single vector v. These are called **Krylov subspaces**[11] and the corresponding matrix

$$K_m = \big(v, Av, \ldots, A^{m-1}v\big)$$

is called a Krylov matrix. If $m \leq n$ the dimension of $\mathcal{K}_m$ usually equals m unless v is specially related to A.

Many methods for the solving the eigenvalue problem developed by Krylov and others in the 1930's and 40's aimed at bringing the characteristic equation into polynomial form. Although this in general is a bad idea, we will consider one approach, which is of interest because of its connection with Krylov subspace methods and the **Lanczos process**.

Throughout this section we assume that $A \in \mathbf{R}^{n\times n}$ is a real symmetric matrix. Associated with A is the characteristic polynomial (9.1.5)

$$p(\lambda) = (-1)^n(\lambda^n - \xi_{n-1}\lambda^{n-1} - \cdots \xi_0) = 0.$$

The Cayley–Hamilton theorem states that $p(A) = 0$, that is

$$A^n = \xi_{n-1}A^{n-1} + \cdots \xi_1 A + \xi_0. \tag{9.8.13}$$

In particular we have

$$\begin{aligned} A^n v &= \xi_{n-1}A^{n-1}v + \cdots \xi_1 Av + \xi_0 v \\ &= [v, Av, \ldots, A^{n-1}v]x, \end{aligned}$$

[11] Named after Aleksei Nikolaevich Krylov (1877–1945) Russian mathematician. Krylov worked at the Naval Academy in Saint-Petersburg and in 1931 published a paper [30] on what is now called "Krylov subspaces".

where $x = (\xi_0, \xi_1, \ldots, \xi_{n-1})^T$.

Consider the **Krylov sequence** of vectors, $v_0 = v$,

$$v_{j+1} = Av_j, \quad j = 0 : n-1. \tag{9.8.14}$$

We assume in the following that v is chosen so that $v_i \neq 0$, $i = 0 : n-1$, Then we may write (9.8.14) as

$$xBx = v_n, \quad B = [v_0, v_1, \ldots, v_{n+1}] \tag{9.8.15}$$

which is a linear equations in n unknowns.

Multiplying (9.8.15) on the left with B^T we obtain a symmetric linear system, the normal equations

$$Mx = z, \quad M = B^T B, \quad z = B^T v_n.$$

The elements m_{ij} of the matrix M are

$$m_{i+1,j+1} = v_i^T v_j = (A^i v)^T A^j v = v^T A^{i+j} v.$$

They only depend on the sum of the indices and we write

$$m_{i+1,j+1} = \mu_{i+j}, \quad i + j = 0; 2n-1.$$

Unfortunately this system tends to be very ill-conditioned. For larger values of n the Krylov vectors soon become parallel to the eigenvector associated with the dominant eigenvalue.

The Krylov subspace $\mathcal{K}_m(v, A)$ depends on both A and v. However, it is important to note the following simply verified invariance properties:

- Scaling: $\mathcal{K}_m(\alpha v, \beta A) = \mathcal{K}_m(v, A)$, $\alpha \neq 0$, $\beta \neq 0$.
- Translation: $\mathcal{K}_m(v, A - \mu I) = \mathcal{K}_m(v, A)$.
- Similarity: $\mathcal{K}_m(Q^T v, Q^T A Q) = Q^T \mathcal{K}_m(v, A)$, $Q^T Q = I$.

These invariance can be used to deduce some important properties of methods using Krylov subspaces. Since A and $-A$ generate the same subspaces the left and right part of the spectrum of A are equally approximated. The invariance with respect to shifting shows, e.g, that it does not matter if A is positive definite or not.

We note that the Krylov subspace $\mathcal{K}(v, A)$ is spanned by the vectors generated by performing $k-1$ steps of the power method starting with v. However, in the power method we throw away previous vectors and just use the last vector $A^k v$ to get an approximate eigenvector. It turns out that this is wasteful and that much more powerful methods can be developed which work with the complete Krylov subspace.

Any vector $x \in \mathcal{K}_m(v)$ can be written in the form

$$x = \sum_{i=0}^{m-1} c_i A^i v = P_{m-1}(A)v,$$

where P_{m-1} is a polynomial of degree less than m. This provides a link between polynomial approximation and Krylov type methods, the importance of which will become clear in the following.

A fundamental question is: How well can an eigenvector of A be approximated by a vector in $\mathcal{K}(v, A)$? Let Π_k denote the orthogonal projector onto the Krylov subspace $\mathcal{K}(v, A)$. The following lemma bounds the distance $\|u_i - \Pi_k u_i\|_2$, where u_i is a particular eigenvector of A.

Theorem 9.8.4.

Assume that A is diagonalizable and let the initial vector v have the expansion

$$v = \sum_{k=1}^{n} \alpha_k u_k \tag{9.8.16}$$

in terms of the normalized eigenvectors $u_1, \ldots, u_n$. Let P_{k-1} be the set of polynomials of degree at most $k-1$ such that $p(\lambda_i) = 1$. Then, if $\alpha_i \neq 0$ the following inequality holds:

$$\|u_i - \Pi_k u_i\|_2 \leq \xi_i \epsilon_i^{(k)}, \quad \xi_i = \sum_{j \neq i} |\alpha_j|/|\alpha_i|, \tag{9.8.17}$$

where

$$\epsilon_i^{(k)} = \min_{p \in P_{k-1}} \max_{\lambda \in \lambda(A) - \lambda_i} |p(\lambda)|. \tag{9.8.18}$$

Proof. We note that any vector in $\mathcal{K}_k$ can be written $q(A)v$, where q is a polynomial $q \in P_{k-1}$. Since Π_k is the orthogonal projector onto $\mathcal{K}_k$ we have

$$\|(I - \Pi_k)u_i\|_2 \leq \|u_i - q(A)v\|_2.$$

Using the expansion (9.8.16) of v it follows that for any polynomial $p \in P_{k-1}$ with $p(\lambda_i) = 1$ we have

$$\|(I - \Pi_k)\alpha_i u_i\|_2 \leq \left\|\alpha_i u_i - \sum_{j=1}^{n} \alpha_j p(\lambda_j) u_j\right\|_2 \leq \max_{j \neq i} |p(\lambda_j)| \left| \sum_{j \neq i} |\alpha_j| \right|.$$

The last inequality follows noticing that the component in the eigenvector u_i is zero and using the triangle inequality. Finally dividing by $|\alpha_i|$ establishes the result. □

To obtain error bounds we use the properties of the Chebyshev polynomials. We now consider the Hermitian case and assume that the eigenvalues of A are simple and ordered so that $\lambda_1 > \lambda_2 > \cdots > \lambda_n$. Let $T_k(x)$ be the Chebyshev polynomial of the first kind of degree k. Then $|T_k(x)| \leq 1$ for $|x| \leq 1$, and for $|x| \geq 1$ we have

$$T_k(x) = \frac{1}{2}\left[(x + \sqrt{x^2 - 1})^k + (x - \sqrt{x^2 - 1})^k\right]. \tag{9.8.19}$$

Now if we take

$$x = l_i(\lambda) = 1 + 2\frac{\lambda - \lambda_{i+1}}{\lambda_{i+1} - \lambda_n}, \quad \gamma_i = l_i(\lambda_i) = 1 + 2\frac{\lambda_i - \lambda_{i+1}}{\lambda_i - \lambda_n}. \tag{9.8.20}$$

the interval $\lambda = [\lambda_{i+1}, \lambda_n]$ is mapped onto $x = [-1, 1]$, and $\gamma_1 > 1$. In particular, for $i = 1$, we take

$$p(\lambda) = \frac{T_{k-1}(l_1(\lambda))}{T_{k-1}(\gamma_1)}.$$

Then $p(\lambda_1) = 1$ as required by Theorem 9.8.4. When k is large we have

$$\epsilon_1^{(k)} \le \max_{\lambda \in \lambda(A) - \lambda_i} |p(\lambda)| \le \frac{1}{T_{k-1}(\gamma_1)} \approx 2 \Big/ \left(\gamma_1 + \sqrt{\gamma_1^2 - 1}\right)^{k-1}. \tag{9.8.21}$$

The steep climb of the Chebyshev polynomials outside the interval $[-1, 1]$ explains the powerful approximation properties of the Krylov subspaces. The approximation error tends to zero with a rate depending on the gap $\lambda_1 - \lambda_2$ normalized by the spread of the rest of the eigenvalues $\lambda_2 - \lambda_n$. Note that this has the correct form with respect to the invariance properties of the Krylov subspaces.

By considering the matrix $-A$ we get analogous convergence results for the rightmost eigenvalue λ_n of A. In general, for $i > 1$, similar but weaker results can be proved using polynomials of the form

$$p(\lambda) = q_{i-1}(\lambda)\frac{T_{k-i}(l_i(\lambda))}{T_{k-i}(\gamma_i)}, \quad q_{i-1}(\lambda) = \prod_{j=1}^{i-1} \frac{\lambda_j - \lambda}{\lambda_j - \lambda_i}.$$

Notice that $q_{i-1}(\lambda)$ is a polynomial of degree $i-1$ with $q_{i-1}(\lambda_j) = 0$, $j = 1, \ldots, i-1$, and $q_{i-1}(\lambda_i) = 1$. Further

$$\max_{\lambda \in \lambda(A) - \lambda_i} |q_{i-1}(\lambda)| \le |q_{i-1}(\lambda_n)| = C_i. \tag{9.8.22}$$

Thus when k is large we have

$$\epsilon_i^{(k)} \le C_i / T_{k-i}(\gamma_i). \tag{9.8.23}$$

This indicates that we can expect interior eigenvalues and eigenvectors to be less well approximated by Krylov-type methods.

9.8.4 The Lanczos Process

We will now show that the Rayleigh–Ritz procedure can be applied to the sequence of Krylov subspaces $\mathcal{K}_m(v)$, $m = 1, 2, 3, \ldots$, in a very efficient way using the **Lanczos process**. The Lanczos process, developed by Lanczos [33, 1950], can be viewed as a way for reducing a symmetric matrix A to tridiagonal form $T = Q^T A Q$. Here

$Q = (q_1, q_2, \ldots, q_n)$ is orthogonal, where q_1 can be chosen arbitrarily, and

$$T = T_n = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \ddots & \ddots & \\ & & \ddots & \alpha_{n-1} & \beta_n \\ & & & \beta_n & \alpha_n \end{pmatrix}. \tag{9.8.24}$$

is symmetric tridiagonal.

Equating the first $n-1$ columns in $A(q_1, q_2, \ldots, q_n) = (q_1, q_2, \ldots, q_n)T$ gives

$$Aq_j = \beta_j q_{j-1} + \alpha_j q_j + \beta_{j+1} q_{j+1}, \quad j = 1, \ldots, n-1.$$

where we have put $\beta_1 q_0 \equiv 0$. The requirement that $q_{j+1} \perp q_j$ gives

$$\alpha_j = q_j^T (Aq_j - \beta_j q_{j-1}),$$

(Note that since $q_j \perp q_{j-1}$ the last term could in theory be dropped; however, since a loss of orthogonality occurs in practice it should be kept. This corresponds to using the modified rather than the classical Gram-Schmidt orthogonalization process.)

Further solving for q_{j+1},

$$\beta_{j+1} q_{j+1} = r_{j+1}, \qquad r_{j+1} = Aq_j - \alpha_j q_j - \beta_j q_{j-1},$$

so if $r_{j+1} \neq 0$, then β_{j+1} and q_{j+1} is obtained by normalizing r_{j+1}. Given q_1 these equations can be used recursively to compute the elements in the tridiagonal matrix T and the orthogonal matrix Q.

Algorithm 9.8.3
The Lanczos Process.
Let A be a symmetric matrix and $q_1 \neq 0$ a given vector. The following algorithm computes in exact arithmetic after k steps a symmetric tridiagonal matrix $T_k = \text{trid}\,(\beta_j, \alpha_j, \beta_{j+1})$ and a matrix $Q_k = (q_1, \ldots, q_k)$ with orthogonal columns spanning the Krylov subspace $\mathcal{K}_k(q_1, A)$:

$r_0 = q_1;\ q_0 = 0;$
$\beta_1 = \|r_0\|_2 = 1;$
for $j = 1, 2, 3 \ldots$
  $q_j = r_{j-1}/\beta_j;$
  $r_j = Aq_j - \beta_j q_{j-1};$
  $\alpha_j = q_j^T r_j;$
  $r_j = r_j - \alpha_j q_j;$
  $\beta_{j+1} = \|r_j\|_2;$
  if $\beta_{j+1} = 0$ **then** *exit*;
end

Note that A only occurs in the matrix-vector operation Aq_j. Hence, the matrix A need not be explicitly available, and can be represented by a subroutine. Only three n-vectors are needed in storage.

It is easy to see that if the Lanczos algorithm can be carried out for k steps then it holds

$$AQ_k = Q_k T_k + \beta_{k+1} q_{k+1} e_k^T. \tag{9.8.25}$$

The Lanczos process stops if $\beta_{k+1} = 0$ since then q_{k+1} is not defined. However, then by (9.8.25) it holds that $AQ_k = Q_k T_k$, and thus Q_k spans an invariant subspace of A. This means that the eigenvalues of T_k also are eigenvalues of A. (For example, if q_1 happens to be an eigenvector of A, the process stops after one step.) Further eigenvalues of A can the be determined by restarting the Lanczos process with a vector orthogonal to $q_1, \ldots, q_k$.

By construction it follows that $\text{span}(Q_k) = \mathcal{K}_k(A, b)$. Multiplying (9.8.25) by Q_k^T and using $Q_k^T q_{k+1} = 0$ it follows that $T_k = Q_k^T A Q_k$, and hence T_k is the generalized Rayleigh quotient matrix corresponding to $\mathcal{K}_k(A, b)$. The Ritz values are the eigenvalues θ_i of T_k, and the Ritz vectors are $y_i = Q_k z_i$, where z_i are the eigenvectors of T_k corresponding to θ_i.

In principle we could at each step compute the Ritz values θ_i and Ritz vectors y_i, $i = 1, \ldots, k$. Then the accuracy of the eigenvalue approximations could be assessed from the residual norms $\|Ay_i - \theta_i y_i\|_2$, and used to decide if the process should be stopped. However, this is not necessary since using (9.8.25) we have

$$Ay_i - y_i\theta_i = AQ_k z_i - Q_k z_i \theta_i = (AQ_k - Q_k T_k) z_i = \beta_{k+1} q_{k+1} e_k^T z_i.$$

Taking norms we get

$$\|Ay_i - y_i\theta_i\|_2 = \beta_{k+1} |e_k^T z_i|. \tag{9.8.26}$$

i.e., we can compute the residual norm just from the bottom element of the normalized eigenvectors of T_k. This is fortunate since then we need to access the Q matrix only after the process has converged. The vectors can be stored on secondary storage, or often better, regenerated at the end. The result (9.8.26) also explains why some Ritz values can be very accurate approximations even when β_{k+1} is not small.

So far we have discussed the Lanczos process in exact arithmetic. In practice, roundoff will cause the generated vectors to lose orthogonality. A possible remedy is to reorthogonalize each generated vector q_{k+1} to all previous vectors $q_k, \ldots, q_1$. This is however very costly both in terms of storage and operations.

A satisfactory analysis of the numerical properties of the Lanczos process was first given by C. C. Paige [36, 1971]. He showed that it could be very effective in computing accurate approximations to a few of the extreme eigenvalues of A even in the face of total loss of orthogonality! The key to the behaviour is, that at the same time as orthogonality is lost, a Ritz pair converges to an eigenpair of A. As the algorithm proceeds it will soon start to converge to a second copy of the already converged eigenvalue, and so on. The effect of finite precision is to slow down convergence, but does not prevent accurate approximations to be found!

The Lanczos process is also the basis for several methods for solving large scale symmetric linear systems, and least squares problems, see Section 10.4.

9.8.5 Golub–Kahan Bidiagonalization.

A Lanczos process can also be developed for computing singular values and singular vectors to a rectangular matrix A. For this purpose we consider here the Golub–Kahan bidiagonalization (GKBD) of a matrix $A \in \mathbf{R}^{m\times n}$, $m \geq n$. This has important applications for computing approximations to the large singular values and corresponding singular vectors, as well as for solving large scale least squares problems.

In Section 8.4.8 we gave an algorithm for computing the decomposition

$$A = U \begin{pmatrix} B \\ 0 \end{pmatrix} V^T, \quad U^T U = I_m, \quad V^T V = I_n, \tag{9.8.27}$$

where $U = (u_1, \ldots, u_m)$ and $V = (v_1, \ldots, v_n)$ are chosen as products of Householder transformations and B is upper bidiagonal. If we set $U_1 = (u_1, \ldots, u_n)$ then from (9.8.27) we have

$$AV = U_1 B, \qquad A^T U_1 = V B^T. \tag{9.8.28}$$

In an alternative approach, given by Golub and Kahan [19, 1965], the columns of U and V are generated sequentially, as in the Lanczos process.

A more useful variant of this bidiagonalization algorithm is obtained by instead taking transforming A into **lower** bidiagonal form

$$B_n = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_n \\ & & & \beta_{n+1} \end{pmatrix} \in \mathbf{R}^{(n+1)\times n}. \tag{9.8.29}$$

(Note that B_n is not square.) Equating columns in (9.8.28) we obtain, setting $\beta_1 v_0 \equiv 0$, $\alpha_{n+1} v_{n+1} \equiv 0$, the recurrence relations

$$\begin{aligned} A^T u_j &= \beta_j v_{j-1} + \alpha_j v_j, \\ A v_j &= \alpha_j u_j + \beta_{j+1} u_{j+1}, \quad j = 1, \ldots, n. \end{aligned} \tag{9.8.30}$$

Starting with a given vector $u_1 \in \mathbf{R}^m$, $\|u_1\|_2 = 1$, we can now recursively generate the vectors $v_1, u_2, v_2, \ldots, u_{m+1}$ and corresponding elements in B_n using, for $j = 1, 2, \ldots$, the formulas

$$r_j = A^T u_j - \beta_j v_{j-1}, \quad \alpha_j = \|r_j\|_2, \quad v_j = r_j/\alpha_j, \tag{9.8.31}$$

$$p_j = A v_j - \alpha_j u_j, \qquad \beta_{j+1} = \|p_j\|_2, \quad u_{j+1} = p_j/\beta_{j+1}. \tag{9.8.32}$$

For this bidiagonalization scheme we have

$$u_j \in \mathcal{K}_j(AA^T, u_1), \qquad v_j \in \mathcal{K}_j(A^T A, A^T u_1).$$

There is a close relationship between the above bidiagonalization process and the Lanczos process applied to the two matrices AA^T and A^TA. Note that these matrices have the same nonzero eigenvalues σ_i^2, $i = 1, \ldots, n$, and that the corresponding eigenvectors equal the left and right singular vectors of A, respectively.

The GKBD process (9.8.31)–(9.8.32) generates in exact arithmetic the same sequences of vectors $u_1, u_2, \ldots$ and $v_1, v_2, \ldots$ as are obtained by simultaneously applying the Lanczos process to AA^T with starting vector $u_1 = b/\|b\|_2$, and to A^TA with starting vector $v_1 = A^Tb/\|A^Tb\|_2$.

In floating point arithmetic the computed Lanczos vectors will lose orthogonality. In spite of this the extreme (largest and smallest) singular values of the truncated bidiagonal matrix $B_k \in \mathbf{R}^{(k+1)\times k}$ tend to be quite good approximations to the corresponding singular values of A, even for $k \ll n$. Let the singular value decomposition of B_k be $B_k = P_{k+1}\Omega_k Q_k^T$. Then approximations to the singular vectors of A are

$$\hat{U}_k = U_k P_{k+1}, \qquad \hat{V}_k = V_k Q_k.$$

This is a simple way of realizing the Ritz–Galerkin projection process on the subspaces $\mathcal{K}_j(A^TA, v_1)$ and $\mathcal{K}_j(AA^T, Av_1)$. The corresponding approximations are called Ritz values and Ritz vectors.

Lanczos algorithms for computing selected singular values and vectors have been developed, which have been used, e.g., in information retrieval problems and in seismic tomography. In these applications typically, the 100–200 largest singular values and vectors for matrices having up to 30,000 rows and 20,000 columns are required.

9.8.6 Arnoldi's Method.

Arnoldi's method is an orthogonal projection method onto Krylov subspace $\mathcal{K}_m$ for general non Hermitian matrices. The procedure starts by building an orthogonal basis for $\mathcal{K}_m$

Algorithm 9.8.4
The Arnoldi process.
Let A be a matrix and v_1, $\|v_1\|_2 = 1$, a given vector. The following algorithm computes in exact arithmetic after k steps a Hessenberg matrix $H_k = (h_{ij})$ and a matrix $V_k = (v_1, \ldots, v_k)$ with orthogonal columns spanning the Krylov subspace $\mathcal{K}_k(v_1, A)$:

for $j = 1 : k$
  for $i = 1 : j$
    $h_{ij} = v_i^H(Av_j)$;
  end
  $r_j = Av_j - \sum_{i=1}^{j} h_{ij}v_i$;
  $h_{j+1,j} = \|r_j\|_2$;

if $h_{j+1,j} = 0$ **then** $exit$;
$v_{j+1} = r_j / h_{j+1,j}$;
end

The Hessenberg matrix $H_k \in \mathbf{C}^{k\times k}$ and the unitary matrix V_k computed in the Arnoldi process satisfy the relations

$$AV_k = V_k H_k + h_{k+1,k} v_{k+1} e_k^H, \tag{9.8.33}$$
$$V_k^H A V_k = H_k. \tag{9.8.34}$$

The process will break down at step j if and only if the vector r_j vanishes. When this happens we have $AV_k = V_k H_k$, and so $\mathcal{R}(V_k)$ is an invariant subspace of A. By (9.8.33) $H_k = V_k^H A V_k$ and thus the Ritz values and Ritz vectors are obtained from the eigenvalues and eigenvectors of H_k. The residual norms can be inexpensively obtained as follows (cf. (9.8.26))

$$\|(A - \theta_i I) y_i\|_2 = h_{m+1,m} |e_k^T z_i|. \tag{9.8.35}$$

The proof of this relation is left as an exercise.

Review Questions

1. Tell the names of two algorithms for (sparse) symmetric eigenvalue problems, where the matrix A need not to be explicitly available but only as a subroutine for the calculation of Aq for an arbitrary vector q. Describe one of the algorithms.
2. Tell the names of two algorithms for (sparse) symmetric eigenvalue problems, where the matrix A need not to be explicitly available but only as a subroutine for the calculation of Aq for an arbitrary vector q. Describe one of the algorithms.

Problems

1. (To be added.)

9.9 Generalized Eigenvalue Problems

9.9.1 Introduction

In this section we consider the **generalized eigenvalue problem** of computing nontrivial solutions (λ, x) of

$$Ax = \lambda Bx, \tag{9.9.1}$$

where A and B are square matrices of order n. The family of matrices $A - \lambda B$ is called a **matrix pencil**.[12] It is called a **regular** pencil if $\det(A - \lambda B) \not\equiv 0$, else it is a **singular** pencil. A simple example of a singular pencil is

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix},$$

where A and B have a null vector e_2 in common.

If $A - \lambda B$ is a regular pencil, then the eigenvalues λ are the zeros of the characteristic equation

$$\det(A - \lambda B) = 0. \tag{9.9.2}$$

If the degree of the characteristic polynomial is $n - p$, then we say that $A - \lambda B$ has p eigenvalues at ∞.

Example 9.9.1.

The characteristic equation of the pencil

$$A - \lambda B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \lambda \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

is $\det(A - \lambda B) = 1 - \lambda$ and has degree one. There is one eigenvalue $\lambda = \infty$ corresponding to the eigenvector e_1.

Note that infinite eigenvalues of $A - \lambda B$ simply correspond to the zero eigenvalues of the pencil $B - \lambda A$.

If S and T are nonsingular matrices then (9.9.2) is equivalent to

$$\det S(A - \lambda B)T = \det(SAT - \lambda SBT) = 0.$$

The two pencils $A - \lambda B$ and $SAT - \lambda SBT$ are said to be **equivalent**. They have the same eigenvalues and the eigenvectors are simply related.

If A and B are symmetric, then symmetry is preserved under congruence transformations in which $T = S^T$. The two pencils are then said to be **congruent**. Of particular interest are orthogonal congruence transformations, $S = Q^T$ and $T = Q$, where Q is orthogonal. Such transformations are stable since they preserve the 2-norm,

$$\|Q^T A Q\|_2 = \|A\|_2, \qquad \|Q^T B Q\|_2 = \|B\|_2.$$

9.9.2 Canonical Forms

The algebraic and analytic theory of the generalized eigenvalue problem is much more complicated than for the standard problem, and a complete treatment is outside the scoop of this book. There is a canonical form for regular matrix pencils corresponding to the Jordan canonical form, Theorem 9.2.7, which we state without proof.

[12] The word "pencil" comes from optics and geometry, and is used for any one parameter family of curves, matrices, etc.

Theorem 9.9.1. Kronecker's Canonical Form.

Let $A - \lambda B \in \mathbf{C}^{n\times n}$ be a regular matrix pencil. Then there are nonsingular matrices $X, Z \in \mathbf{C}^{n\times n}$, such that $X^{-1}(A - \lambda B)Z = \hat{A} - \lambda\hat{B}$, where

$$\begin{aligned}\hat{A} &= \operatorname{diag}(J_{m_1}(\lambda_1), \ldots, J_{m_s}(\lambda_s), I_{m_{s+1}}, \ldots, I_{m_t}),\\ \hat{B} &= \operatorname{diag}(I_{m_1}, \ldots, I_{m_s}, J_{m_{s+1}}(0), \ldots, J_{m_t}(0)),\end{aligned} \tag{9.9.3}$$

and where $J_{m_i}(\lambda_i)$ are Jordan blocks and the blocks $s+1, \ldots, t$ correspond to infinite eigenvalues. The numbers $m_1, \ldots, m_t$ are unique and $\sum_{i=1}^{t} m_i = n$.

The disadvantage with the Kronecker Canonical Form is that it depends discontinuously on A and B and is unstable. There is also a generalization of the Schur Canonical Form (Theorem 9.2.1), which can be computed stably and more efficiently.

Theorem 9.9.2. Generalized Schur Canonical Form.

Let $A - \lambda B \in \mathbf{C}^{n\times n}$ be a regular matrix pencil. Then there exist unitary matrices U and V so that

$$UAV = T_A, \qquad UBV = T_B,$$

where both T_A and T_B are upper triangular. The eigenvalues of the pencil are the ratios of the diagonal elements of T_A and T_B.

Proof. See Stewart [1973, Ch. 7.6]. □

As for the standard case, when A and B are real, then U and V can be chosen real and orthogonal if T_A and T_B are allowed to have 2×2 diagonal blocks corresponding to complex conjugate eigenvalues.

9.9.3 Reduction to Standard Form

When B is nonsingular the eigenvalue problem (9.9.1) is formally equivalent to the standard eigenvalue problem $B^{-1}Ax = \lambda x$. However, when B is singular such a reduction is not possible. Also, if B is close to a singular matrix, then we can expect to lose accuracy in forming $B^{-1}A$.

Of particular interest is the case when the problem can be reduced to a symmetric eigenvalue problem of standard form. A surprising fact is that any real square matrix F can be written as $F = AB^{-1}$ or $F = B^{-1}A$ where A and B are suitable symmetric matrices. For a proof see Parlett [38, Section 15-2] (cf. also Problem 1). Hence, even if A and B are symmetric the generalized eigenvalue problems embodies all the difficulties of the unsymmetric standard eigenvalue problem. However, if B is also positive definite, then the problem (9.9.1) can be reduced to a standard symmetric eigenvalue problem. This reduction is equivalent to the simultaneous transformation of the two quadratic forms x^TAx and x^TBx to diagonal form.

Theorem 9.9.3.

Let A and B be real symmetric square matrices and B also positive definite. Then there exists a nonsingular matrix X such that

$$X^TAX = D_A, \qquad X^TBX = D_B \tag{9.9.4}$$

are real and diagonal. The eigenvalues of $A - \lambda B$ are given by

$$\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) = D_A D_B^{-1}.$$

Proof. Let $B = LL^T$ be the Cholesky factorization of B. Then

$$L^{-1}(A - \lambda B)L^{-T} = \tilde{A} - \lambda I, \qquad \tilde{A} = \tilde{A} = L^{-1}AL^{-T}, \tag{9.9.5}$$

where $\tilde{A}$ is real and symmetric. Let $\tilde{A} = Q^T D_A Q$ be the eigendecomposition of $\tilde{A}$. Then we have

$$X^T(A - \lambda B)X = D_A - \lambda D_B, \qquad X = (QL^{-1})^T,$$

and the theorem follows. □

Given the pencil $A - \lambda B$ the pencil $\hat{A} - \lambda\hat{B} = \gamma A + \sigma B - \lambda(-\sigma A + \gamma B)$, where $\gamma^2 + \sigma^2 = 1$ has the same eigenvectors and the eigenvalues are related through

$$\lambda = (\gamma\hat{\lambda} + \sigma)/(-\sigma\hat{\lambda} + \gamma). \tag{9.9.6}$$

Hence, for the above reduction to be applicable, it suffices that some linear combination $-\sigma A + \gamma B$ is positive definite. It can be shown that if

$$\inf_{x \neq 0} \left((x^TAx)^2 + (x^TBx)^2 \right)^{1/2} > 0$$

then there exist such γ and σ.

Under the assumptions in Theorem 9.9.3 the symmetric pencil $A - \lambda B$ has n real roots. Moreover, the eigenvectors can be chosen to be B-orthogonal, i.e.,

$$x_i^TBx_j = 0, \qquad i \neq j.$$

This generalizes the standard symmetric case for which $B = I$.

Numerical methods can be based on the *explicit reduction to standard form* in (9.9.5). $Ax = \lambda Bx$ is then equivalent to $Cy = \lambda y$, where

$$C = L^{-1}AL^{-T}, \qquad y = L^Tx. \tag{9.9.7}$$

Computing the Cholesky decomposition $B = LL^T$ and forming $C = (L^{-1}A)L^{-T}$ takes about $5n^3/12$ flops if symmetry is used, see Wilkinson and Reinsch, Contribution II/10, [53]. If eigenvectors are not wanted, then the transform matrix L need not be saved.

If A and B are symmetric band matrices and $B = LL^T$ positive definite, then although L inherits the bandwidth of A the matrix $C = (L^{-1}A)L^{-T}$ will in general be a full matrix. Hence in this case it may not be practical to form C. Crawford [7] has devised an algorithm for reduction to standard form which interleaves orthogonal transformations in such way that the matrix C retains the bandwidth of A, see Problem 2.

The round-off errors made in the reduction to standard form are in general such that they could be produced by small perturbations in A and B. When B is ill-conditioned then the eigenvalues λ may vary widely in magnitude, and a small perturbation in B can correspond to large perturbations in the eigenvalues. Surprisingly, well-conditioned eigenvalues are often given accurately in spite of the ill-conditioning of B. Typically L will have elements in its lower part. This will produce a matrix $(L^{-1}A)L^{-T}$ which is graded so that the large elements appear in the lower right corner. Hence, a reduction to tridiagonal form should work from bottom to top and the QL-algorithm should be used.

Example 9.9.2. *Wilkinson and Reinsch [53, p. 310].*

The matrix pencil $A - \lambda B$, where

$$A = \begin{pmatrix} 2 & 2 \\ 2 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 2 \\ 2 & 4.0001 \end{pmatrix},$$

has one eigenvalue ≈ -2 and one $\mathrm{O}(10^4)$. The true matrix

$$(L^{-1}A)L^{-T} = \begin{pmatrix} 2 & -200 \\ -200 & 10000 \end{pmatrix}$$

is graded, and the small eigenvalue is insensitive to relative perturbation in its elements.

9.9.4 Methods for Generalized Eigenvalue Problems

We first note that the power method and inverse iteration can both be extended to the generalized eigenvalue problems. Starting with some q_0 with $\|q_0\|_2 = 1$, these iterations now become

$$\begin{aligned} B\hat{q}_k &= Aq_{k-1}, \quad q_k = \hat{q}_k/\|\hat{q}_k\|, \\ (A - \sigma B)\hat{q}_k &= Bq_{k-1}, \qquad q_k = \hat{q}_k/\|\hat{q}_k\|, \qquad k = 1, 2, \ldots \end{aligned}$$

respectively. Note that $B = I$ gives the iterations in equations (9.5.4) and (9.5.7). The Rayleigh Quotient Iteration also extends to the generalized eigenvalue problem: For $k = 0, 1, 2, \ldots$ compute

$$(A - \rho(q_{k-1})B)\hat{q}_k = Bq_{k-1}, \qquad q_k = \hat{q}_k/\|q_k\|_2, \tag{9.9.8}$$

where the (generalized) Rayleigh quotient of x is

$$\rho(x) = \frac{x^H Ax}{x^H Bx}.$$

In the symmetric definite case the Rayleigh Quotient Iteration has asymptotically cubic convergence and the residuals $\|(A-\mu_k B)x_k\|_{B^{-1}}$ decrease monotonically.

The Rayleigh Quotient method is advantageous to use when A and B have band structure, since it does not require an explicit reduction to standard form. The method of spectrum slicing can be used to count eigenvalues of $A-\lambda B$ in an interval.

Theorem 9.9.4.
Let $A-\sigma B$ have the Cholesky factorization

$$A-\sigma B = LDL^T, \qquad D = \operatorname{diag}(d_1,\ldots,d_n),$$

where L is unit lower triangular. If B is positive definite then the number of eigenvalues of A greater than σ equals the number of positive elements $\pi(D)$ in the sequence $d_1,\ldots,d_n$.

Proof. The proof follows from Sylvester's Law of Inertia (Theorem 7.3.8) and the fact that by Theorem 9.9.1 A and B are congruent to D_A and D_B with $\Lambda = D_A D_B^{-1}$. □

For a nearly singular pencil (A,B) it may be preferable to use the **QZ algorithm** of Moler and Stewart which is a generalization of the implicit QR algorithm. Here the matrix A is first reduced to upper Hessenberg form H_A and simultaneously B to upper triangular form R_B using standard Householder transformations and Givens rotations. Infinite eigenvalues, which correspond to zero diagonal elements of R_B are then eliminated. Finally the implicit shift QR algorithm is applied to $H_A R_B^{-1}$, without explicitly forming this product. This is achieved by computing unitary matrices Q and Z such that QAZ is upper Hessenberg, QBZ upper triangular and choosing the first column of Q proportional to the first column of $H_A R_B^{-1} - \sigma I$. A double shift technique can also be used if A and B are real. The matrix H_A will converge to upper triangular form and the eigenvalues of $A-\lambda B$ will be obtained as ratios of diagonal elements of the transformed H_A and R_B. For a more detailed description of the algorithm see Stewart [43, Chapter 7.6].

The total work in the QZ algorithm is about $15n^3$ flops for eigenvalues alone, $8n^3$ more for Q and $10n^3$ for Z (assuming 2 QZ iterations per eigenvalue). It avoids the loss of accuracy related to explicitly inverting B. Although the algorithm is applicable to the case when A is symmetric and B positive definite, the transformations do not preserve symmetry and the method is just as expensive as for the general problem.

9.9.5 The Generalized SVD.

We now introduce the **generalized singular value decomposition** (GSVD) of two matrices $A \in \mathbf{R}^{m\times n}$ and $B \in \mathbf{R}^{p\times n}$ with the same number of columns. The GSVD has applications to, e.g., constrained least squares problems. The GSVD is related to the generalized eigenvalue problem $A^TAx = \lambda B^TBx$, but as in the case

of the SVD the formation of A^TA and B^TB should be avoided. In the theorems below we assume for notational convenience that $m \geq n$.

Theorem 9.9.5. The Generalized Singular Value Decomposition (GSVD). *Let $A \in \mathbf{R}^{m\times n}$, $m \geq n$, and $B \in \mathbf{R}^{p\times n}$ be given matrices. Assume that*

$$\operatorname{rank}(M) = k \leq n, \qquad M = \begin{pmatrix} A \\ B \end{pmatrix}.$$

Then there exist orthogonal matrices $U_A \in \mathbf{R}^{m\times m}$ and $U_B \in \mathbf{R}^{p\times p}$ and a matrix $Z \in \mathbf{R}^{k\times n}$ of rank k such that

$$U_A^TA = \begin{pmatrix} D_A \\ 0 \end{pmatrix} Z, \qquad U_B^TB = \begin{pmatrix} D_B & 0 \\ 0 & 0 \end{pmatrix} Z, \tag{9.9.9}$$

where

$$D_A = \operatorname{diag}(\alpha_1, \ldots, \alpha_k), \qquad D_B = \operatorname{diag}(\beta_1, \ldots, \beta_q), \qquad q = \min(p, k).$$

Further, we have

$$\begin{aligned} &0 \leq \alpha_1 \leq \cdots \leq \alpha_k \leq 1, \qquad 1 \geq \beta_1 \geq \cdots \geq \beta_q \geq 0, \\ &\alpha_i^2 + \beta_i^2 = 1, \quad i = 1, \ldots, q, \quad \alpha_i = 1, \quad i = q+1, \ldots, k, \end{aligned}$$

and the singular values of Z equal the nonzero singular values of M.

Proof. We now give a constructive proof of Theorem 9.9.5 using the CS decomposition, Let the SVD of M be

$$M = \begin{pmatrix} A \\ B \end{pmatrix} = Q \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix} P^T,$$

where Q and P are orthogonal matrices of order $(m+p)$ and n, respectively, and

$$\Sigma_1 = \operatorname{diag}(\sigma_1, \ldots, \sigma_k), \qquad \sigma_1 \geq \cdots \geq \sigma_k > 0.$$

Set $t = m + p - k$ and partition Q and P as follows:

$$Q = \begin{pmatrix} Q_{11} & Q_{12} \\ \underbrace{Q_{21}}_{k} & \underbrace{Q_{22}}_{t} \end{pmatrix} \begin{matrix} \}m \\ \}p \end{matrix}, \qquad P = (\underbrace{P_1}_{k}, \underbrace{P_2}_{n-k}).$$

Then the SVD of M can be written

$$\begin{pmatrix} A \\ B \end{pmatrix} P = \begin{pmatrix} AP_1 & 0 \\ BP_1 & 0 \end{pmatrix} = \begin{pmatrix} Q_{11} \\ Q_{21} \end{pmatrix} (\,\Sigma_1 \quad 0\,). \tag{9.9.10}$$

Now let

$$Q_{11} = U_A \begin{pmatrix} C \\ 0 \end{pmatrix} V^T, \qquad Q_{21} = U_B \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T$$

be the CS decomposition of Q_{11} and Q_{21}. Substituting this into (9.9.10) we obtain

$$AP = U_A \begin{pmatrix} C \\ 0 \end{pmatrix} V^T \begin{pmatrix} \Sigma_1 & 0 \end{pmatrix},$$
$$BP = U_B \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix} V^T \begin{pmatrix} \Sigma_1 & 0 \end{pmatrix},$$

and (9.9.9) follows with

$$D_A = C, \quad D_B = S, \quad Z = V^T \begin{pmatrix} \Sigma_1 & 0 \end{pmatrix} P^T.$$

Here $\sigma_1 \geq \cdots \geq \sigma_k > 0$ are the singular values of Z. □

When $B \in \mathbf{R}^{n\times n}$ is square and nonsingular the GSVD of A and B corresponds to the SVD of AB^{-1}. However, when A or B is ill-conditioned, then computing AB^{-1} would usually lead to unnecessarily large errors, so this approach is to be avoided. It is important to note that when B is not square, or is singular, then the SVD of $AB^{\dagger}$ does not in general correspond to the GSVD.

9.9.6 The CS Decomposition.

The CS decomposition is a special case of the generalized SVD (GSVD) which is of interest in its own right.

Theorem 9.9.6. CS Decomposition. *Let $Q \in \mathbf{R}^{(m+p)\times n}$ have orthonormal columns, and be partitioned as*

$$Q = \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} \begin{matrix} \}m \\ \}p \end{matrix} \in \mathbf{R}^{(m+p)\times n}, \qquad m \geq n, \tag{9.9.11}$$

i.e., $Q^TQ = Q_1^TQ_1 + Q_2^TQ_2 = I_n$. Then there are orthogonal matrices $U_1 \in \mathbf{R}^{m\times m}$, $U_2 \in \mathbf{R}^{p\times p}$, and $V \in \mathbf{R}^{n\times n}$, and square nonnegative diagonal matrices

$$C = \operatorname{diag}(c_1, \ldots, c_q), \quad S = \operatorname{diag}(s_1, \ldots, s_q), \qquad q = \min(n, p), \tag{9.9.12}$$

satisfying $C^2 + S^2 = I_q$ such that

$$\begin{pmatrix} U_1^T & 0 \\ 0 & U_2^T \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} V = \begin{pmatrix} U_1^TQ_1V \\ U_2^TQ_2V \end{pmatrix} = \begin{pmatrix} \Sigma_1 \\ \Sigma_2 \end{pmatrix} \begin{matrix} \}m \\ \}p \end{matrix} \tag{9.9.13}$$

has one of the following forms:

$$p \geq n: \quad \begin{pmatrix} C \\ 0 \\ S \\ 0 \end{pmatrix} \begin{matrix} \}n \\ \}m-n \\ \}n \\ \}p-n \end{matrix}, \qquad p < n: \quad \begin{pmatrix} C & 0 \\ 0 & I \\ 0 & 0 \\ \underbrace{S}_{p} & \underbrace{0}_{n-p} \end{pmatrix} \begin{matrix} \}p \\ \}n-p \\ \}m-n \\ \}p \end{matrix}.$$

The diagonal elements c_i and s_i are

$$c_i = \cos(\theta_i), \quad s_i = \sin(\theta_i), \quad i = 1, \ldots, q,$$

where without loss of generality, we may assume that

$$0 \le \theta_1 \le \theta_2 \le \cdots \le \theta_q \le \pi/2.$$

Proof. To construct U_1, V, and C, note that since U_1 and V are orthogonal and C is a nonnegative diagonal matrix, (9.9.13) is the SVD of Q_1. Hence the elements c_i are the singular values of Q_1. If we put $\tilde{Q}_2 = Q_2 V$, then the matrix

$$\begin{pmatrix} C \\ 0 \\ \tilde{Q}_2 \end{pmatrix} = \begin{pmatrix} U_1^T & 0 \\ 0 & I_p \end{pmatrix} \begin{pmatrix} Q_1 \\ Q_2 \end{pmatrix} V$$

has orthonormal columns. Thus $C^2 + \tilde{Q}_2^T \tilde{Q}_2 = I_n$, which implies that $\tilde{Q}_2^T \tilde{Q}_2 = I_n - C^2$ is diagonal and hence the matrix $\tilde{Q}_2 = (\tilde{q}_1^{(2)}, \ldots, \tilde{q}_n^{(2)})$ has orthogonal columns.

We assume that the singular values $c_i = \cos(\theta_i)$ of Q_1 have been ordered according to (9.9.6) and that $c_r < c_{r+1} = 1$. Then the matrix $U_2 = (u_1^{(2)}, \ldots, u_p^{(2)})$ is constructed as follows. Since $\|\tilde{q}_j^{(2)}\|_2^2 = 1 - c_j^2 \neq 0, \quad j \le r$ we take

$$u_j^{(2)} = \tilde{q}_j^{(2)} / \|\tilde{q}_j^{(2)}\|_2, \qquad j = 1, \ldots, r,$$

and fill the possibly remaining columns of U_2 with orthonormal vectors in the orthogonal complement of $\mathcal{R}(\tilde{Q}_2)$. From the construction it follows that $U_2 \in \mathbf{R}^{p \times p}$ is orthogonal and that

$$U_2^T \tilde{Q}_2 = U_2 Q_2 V = \begin{pmatrix} S & 0 \\ 0 & 0 \end{pmatrix}, \qquad S = \operatorname{diag}(s_1, \ldots, s_q)$$

with $s_j = (1 - c_j^2)^{1/2} > 0$, if $j = 1, \ldots, r$, and $s_j = 0$, if $j = r+1, \ldots, q$. ∎

In the theorem above we assumed that $m \ge n$. The general case gives rise to four different forms corresponding to cases where Q_1 and/or Q_2 have too few rows to accommodate a full diagonal matrix of order n.

The proof of the CS decomposition is constructive. In particular U_1, V, and C can be computed by a standard SVD algorithm. However, the above algorithm for computing U_2 is unstable when some singular values c_i are close to 1.

Review Questions

1. What is meant by a regular matrix pencil? Give examples of a singular pencil, and a regular pencil that has an infinite eigenvalue.
2. Formulate a generalized Schur Canonical Form. Show that the eigenvalues of the pencil are easily obtained from the canonical form.

3. Let A and B be real symmetric matrices, and B also positive definite. Show that there is a congruence transformation that diagonalizes the two matrices simultaneously. How is the Rayleigh Quotient iteration generalized to this type of eigenvalue problems, and what is its order of convergence?

Problems

1. Show that the matrix pencil $A - \lambda B$ where

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

has complex eigenvalues, even though A and B are both real and symmetric.

2. Let A and B be symmetric tridiagonal matrices. Assume that B is positive definite and let $B = LL^T$, where the Cholesky factor L is lower bidiagonal.

(a) Show that L can be factored as $L = L_1 L_2 \cdots L_n$, where L_k differs from the unit matrix only in the kth column.

(b) Consider the recursion

$$A_1 = A, \quad A_{k+1} = Q_k L_k^{-1} A_k L_k^{-T} Q_k^T, \quad k = 1, \ldots, n.$$

Show that if Q_k are orthogonal, then the eigenvalues of A_{n+1} are the same as those for the generalized eigenvalue problem $Ax = \lambda Bx$.

(c) Show how to construct Q_k as a sequence of Givens rotations so that the matrices A_k are all tridiagonal. (The general case, when A and B have symmetric bandwidth $m > 1$, can be treated by considering A and B as block-tridiagonal.)

Notes

Complex Givens rotations and complex Householder transformations are treated in detail by Wilkinson [52, pp. 47–50]. For implementation details of complex Householder transformations, see the survey by R. B. Lehoucq [34, 1996].

For a more complete treatment of matrix functions see Chapter V in Gantmacher [15, 1959] and Lancaster [32, 1985]. Stewart and Sun [45] is a lucid treatise of matrix perturbation theory, with many historical comments and a very useful bibliography. Ward 1977 analyzed the method based on scaling and squaring for computing the exponential of a matrix and gave an a posteriori error bound. Moler and Van Loan 1978 gave a backward error analysis covering truncation error in the Padé approximation.

An analysis and a survey of inverse iteration for a single eigenvector is given by Ipsen [26]. The relation between simultaneous iteration and the QR algorithm and is explained in Watkins [50].

A still unsurpassed text on computational methods for the eigenvalue problem is Wilkinson [52, 1965]. Also the Algol subroutines and discussions in Wilkinson and Reinsch [53, 1971] are very instructive. An excellent discussion of the symmetric eigenvalue problem is given in Parlett [38, 1980]. Methods for solving large scale eigenvalue problems are treated by Saad [41, 1992].

The monograph by Bhatia [5] on perturbation theory for eigenspaces of Hermitian matrices is a valuable source of reference.

A stable algorithm for computing the SVD based on an initial reduction to bidiagonal form was first sketched by Golub and Kahan in [19]. The adaption of the QR algorithm, using a simplified process due to Wilkinson, for computing the SVD of the bidiagonal matrix was described by Golub [18]. The "final" form of the QR algorithm for computing the SVD was given by Golub and Reinsch [20]. The GSVD was first studied by Van Loan [21, 1996]. Paige and Saunders [37, 1981] extended the GSVD to handle all possible cases, and gave a computationally more amenable form.

For a survey of cases when it is possible to compute singular values and singular vectors with high relative accuracy; see [8].

Many important practical details on implementation of eigenvalue algorithms can be found in the documentation of the EISPACK and LAPACK software; see Smith et al. [42, 1976], B. S. Garbow et al. [16, 1977], and E. Anderson et al. [1, 1999].

Bibliography

[1] Edward Anderson, Zhaojun Bai, Christian Bischof, S. Blackford, J. Demmel, J. Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, A. McKenney, and Danny Sorensen, editors. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

[2] Zhaojun Bai, James W. Demmel, Jack J. Dongarra, Axel Ruhe, and Henk A. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, PA, 2000.

[3] Jesse Barlow and James W. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal*, 27:762–791, 1990.

[4] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences*. SIAM, Philadelphia, PA, 1994.

[5] Rajendra Bhatia. *Matrix Analysis*. Springer, New York, 1997.

[6] Shivkumar Chandrasekaran and Ilse C. F. Ipsen. Analysis of a QR algorithm for computing singular values. *SIAM J. Matrix Anal. Appl.*, 16:2:520–535, 1995.

[7] C. R. Crawford. Reduction of a band-symmetric generalized eigenvalue problem. *Comm. ACM*, 16:41–44, 1973.

[8] James W. Demmel, Ming Gu, Stanley Eisenstat, Ivan Slapničar, Kresimir Veselić, and Zlatko Drmač. Computing the singular value decomposition with high relative accuracy. *Linear Algebra Appl.*, 299:21–80, 1999.

[9] James W. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM J. Sci. Statist. Comput.*, 11:873–912, 1990.

[10] James W. Demmel and K. Veselić. Jacobi's method is more accurate than QR. *SIAM J. Matrix Anal. Appl.*, 13:4:1204–1245, 1992.

[11] K. V. Fernando. Accurately counting singular values of bidiagonal matrices and eigenvalues of skew-symmetric tridiagonal matrices. *SIAM J. Matrix Anal. Appl.*, 20:2:373–399, 1998.

[12] Roger Fletcher and Danny C. Sorensen. An algorithmic derivation of the Jordan canonical form. *American Mathematical Monthly*, 90, 1983.

[13] George E. Forsythe and Peter Henrici. The cyclic Jacobi method for computing the principal values of a complex matrix. *Trans. Amer. Math. Soc.*, 94:1–23, 1960.

[14] J. G. F. Francis. The QR transformation. Part I and II. *Computer J.*, 4:265–271, 332–345, 1961–1962.

[15] F. R. Gantmacher. *The Theory of Matrices. Vols. I and II.* Chelsea Publishing Co, New York, 1959.

[16] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and G. W. Stewart. *Matrix Eigensystems Routines: EISPACK Guide Extension.* Springer-Verlag, New York, 1977.

[17] S. A. Gerschgorin. Über die Abgrenzung der Eigenwerte einer Matrix. *Akademia Nauk SSSR, Mathematics and Natural Sciences*, 6:749–754, 1931.

[18] Gene H. Golub. Least squares, singular values and matrix approximations. *Aplikace Matematiky*, 13:44–51, 1968.

[19] Gene H. Golub and W. Kahan. Calculating the singular values and pseudoinverse of a matrix. *SIAM J. Numer. Anal. Ser. B*, 2:205–224, 1965.

[20] Gene H. Golub and Christian Reinsch. Singular value decomposition and least squares solution. *Numer. Math*, 14:403–420, 1970.

[21] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[22] Ming Gu and Stanley C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal. *SIAM J. Matrix. Anal. Appl.*, 16:172–191, 1995.

[23] M. R. Hestenes. Inversion of matrices by biorthogonalization and related results. *J. Soc. Indust. Appl. Math.*, 6:51–90, 1958.

[24] Nicholas J. Higham. QR factorization with complete pivoting and accurate computation of the svd. *Linear Algebra Appl.*, 309:153–174, 2000.

[25] Nicholas J. Higham. The scaling and squaring method for the matrix exponential function. *SIAM J. Matrix Anal. Appl.*, 2004.

[26] Ilse Ipsen. Computing an eigenvector with inverse iteration. *SIAM Review*, 39:254–291, 1997.

[27] C. G. J. Jacobi. Über ein leichtes Verfahren die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. *J. reine angew. Math..*, 30:51–94, 1846.

[28] W. M. Kahan. Accurate eigenvalues of a symmetric tri-diagonal matrix. Tech. Report No. CS-41, Revised June 1968, Computer Science Department, Stanford University, CA, 1966.

[29] E. G. Kogbetliantz. Solution of linear equations by diagonalization of coefficients matrix. *Quart. Appl. Math.*, 13:123–132, 1955.

[30] A. N. Krylov. On the numerical solution of the equation by which, in technical matters, frequencies of small oscillations of material systems are determined. *Izv. Akad. Nauk. S.S.S.R. Otdel.Mat. Estest. Nauk*, VII:4:491–539, 1931. in Russian.

[31] Vera N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Comput. Math. Phys.*, 3:637–657, 1961.

[32] Peter Lancaster and M. Tismenetsky. *The Theory of Matrices*. Academic Press, New York, 1985.

[33] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards, Sect. B*, 45:255–282, 1950.

[34] Richard B. Lehoucq. The computations of elementary unitary matrices. *ACM Trans. Math. Software*, 22:393–400, 1996.

[35] Cleve Moler and Charles F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45:3–49, 2003.

[36] Christopher C. Paige. *The Computation of Eigenvalues and Eigenvectors of Very Large Sparse Matrices*. PhD thesis, University of London, 1971.

[37] Christopher C. Paige and Michael A. Saunders. Toward a generalized singular value decomposition. *SIAM J. Numer. Anal.*, 18:398–405, 1981.

[38] Beresford N. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics 20. SIAM, Philadelphia, PA, 1998.

[39] Heinz Rutishauser. Solution of eigenvalue problems with the lr-transformation. *Nat. Bureau of Standards, Appl. Math. Ser.*, 49:47–81, 1958.

[40] Heinz Rutishauser. The Jacobi method for real symmetric matrices. *Numer. Math.*, 9:1–10, 1966.

[41] Yosef Saad. *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

[42] B. T. Smith, J. M. Boyle, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystems Routines—EISPACK Guide*. Springer-Verlag, New York, second edition, 1976.

[43] G. W. Stewart. *Introduction to Matrix Computations*. Academic Press, New York, 1973.

[44] G. W. Stewart. *Matrix Algorithms Volume II: Eigensystems*. SIAM, Philadelphia, PA, 2001.

[45] George W. Stewart and Ji guang. Sun. *Matrix Perturbation Theory*. Academic Press, Boston, MA, 1990.

[46] Gilbert Strang. *Linear Algebra and Its Applications*. Academic Press, New York, third edition, 1988.

[47] Lloyd N. Trefethen. Pseudospectra of linear operators. *SIAM Review*, 39:383–406, 1997.

[48] Charles F. Van Loan. Generalizing the singular value decomposition. *SIAM J. Numer. Anal.*, 13:76–83, 1976.

[49] Richard S. Varga. *Gerschgorin and his Circles*. Springer, Berlin, Heidelberg, New York, 2004.

[50] David S. Watkins. Understanding the QR algorithm. *SIAM Review*, 24:427–440, 1982.

[51] David S. Watkins. *Fundamentals of Matrix Computation*. Wiley-InterScience, New York, 2002.

[52] James H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.

[53] James H. Wilkinson and C. Reinsch, editors. *Handbook for Automatic Computation. Vol. II, Linear Algebra*. Springer-Verlag, New York, 1971.

Index

adjoint matrix, 2
Aitken extrapolation, 50
algorithm
  Givens rotations, 68
  Lanczos, 109
  orthogonal iteration, 103
  Rayleigh–Ritz procedure, 102
  singular values by spectrum slicing, 99
  svd, 60, 65
  The Arnoldi process, 112
  tridiagonal spectrum slicing, 76
analytic function
  of matrix, 22
Arnoldi's method, 112–113
arrowhead matrix, 74

Bauer–Fike's theorem, 38
bidiagonal decomposition
  Lanczos process, 111

canonical form
  Kronecker, 115
  Schur, 11–14
Cayley–Hamilton theorem, 18, 105
characteristic equation, 3
characteristic polynomial, 3
CS decomposition, 120–121

decomposition
  block diagonal, 16
  CS, 120–121
  GSVD, 118
deflation, 51–52
deflation of matrix, 7, 52
departure from normality, 14
divide and conquer
  tridiagonal eigenproblem, 74–75
dominant
  invariant subspace, 57

eigenvalue
  algebraic multiplicity, 7
  by spectrum slicing, 75–77
  defective, 8
  dominant, 49
  error bound, 43–46
  geometric multiplicity, 8
  Jacobi's method, 59–62
  of Kronecker product, 9
  of Kronecker sum, 9
  perturbation, 38–46
  power method, 49–57
  subspace iteration, 56–57
eigenvalue of matrix, 3
eigenvalue problem
  large, 101–113
eigenvector
  of matrix, 3
  perturbation, 38–46
elementary rotations
  unitary, 66
exponential of matrix, 21

field of values, 43
Fischer's theorem, 41
flop count
  QR algorithm for SVD, 97
  QR step, 84
  reduction to Hessenberg form, 69, 72
functions
  matrix-valued, 21–29

gap

of spectrum, 102
generalized eigenvalue problem, 113–118
generalized SVD, 118–120
Gerschgorin disks, 36
Gerschgorin's theorem, 36, 37
Givens rotation
unitary, 67
GKBD, *see* Golub–Kahan bidiagonalization
Golub–Kahan bidiagonalization, 111–112
in finite precision, 112
grade
of vector, 9
graded matrix, 73
graph
connected, 6
directed, 6
growth ratio, 71

Hermitian matrix, 2
Hessenberg form
reduction to, 69–72
Hessenberg matrix
unreduced, 10, 70
Hotelling, 51
Householder reflection
unitary, 67, 68

instability
irrelevant, 70
invariant subspace, 5
inverse iteration, 52–55
shift, 52

Jacobi transformation, 60
Jacobi's method
classical, 61
cyclic, 61
for SVD, 62
sweep, 61
threshold, 61
Jordan block, 9
Jordan canonical form, 15–18

Kronecker
product, 9
sum, 9
Kronecker's canonical form, 115
Krylov
subspaces, 105–108

Lanczos bidiagonalization, *see* Golub–Kahan bidiagonalization, 112
Lanczos process, 108–111
Lyapunov's equation, 15

Markov chain, 29
matrix
adjoint, 2
defective, 8
derogatory, 17
diagonalizable, 5
eigenvalue of, 3
eigenvector of, 3
elementary divisors, 18
exponential, 21
functions, 21–29
graded, 73
Hermitian, 2
irreducible, 6
non-negative irreducible, 28
normal, 13
quasi-triangular, 12
reducible, 6
row stochastic, 29
scaled diagonally dominant, 73
square root, 27
trace, 3
unitary, 2
matrix exponential
hump, 24
matrix pencil, 114
congruent, 114
equivalent, 114
regular, 114
singular, 114
minimal polynomial, 17
of vector, 9
minimax characterization
of eigenvalues, 41

Newton's interpolation formula

for matrix functions, 35
Non-negative matrices, 28–29

one-sided Jacobi SVD, 64–65
orthogonal iteration, 56, 103

Padé approximant, 25
Perron–Frobenius theorem, 28
perturbation
of eigenvalue, 38–46
of eigenvector, 38–46
power method, 49–57
principal vector, 17

QR algorithm, 79–92, 98
explicit-shift, 84
for SVD, 92–96
Hessenberg matrix, 84–88
implicit shift, 85
perfect shifts, 92
rational, 91
Rayleigh quotient shift, 85
symmetric tridiagonal matrix, 89–92
Wilkinson shift, 90
QZ algorithm, 118

radius of convergence, 19
Rayleigh quotient, 43
iteration, 55, 117
matrix, 102, 110
Rayleigh–Ritz procedure, 101–103
reduction
to standard form, 115–117
reduction to
Hessenberg form, 69–72
symmetric tridiagonal form, 72–74
residual vector, 44
Ritz values, 102
Ritz vectors, 102
row stochastic matrix, 29
RQI, *see* Rayleigh quotient iteration

scaled diagonally dominant, 73
Schur
canonical form, 11–14
generalized, 115
vectors, 12
Schur decomposition, 27
secular equation, 48, 74
similarity transformation, 4
singular values
by spectrum slicing, 99
relative gap, 97
spectral abscissa, 4, 22
spectral radius, 4, 19
spectral transformation, 52, 105
spectrum of matrix, 3
spectrum slicing, 75–77
square root of matrix, 27
subspace
invariant, 5
subspace iteration, 103–105
SVD
generalized, 118–120
Sylvester's
equation, 14
law of inertia, 118
symmetric tridiagonal form
reduction to, 72–74

theorem
Cayley-Hamilton, 18
implicit Q, 70, 89
transformation
similarity, 4
tridiagonal matrix, 48
unreduced, 89
two-side Jacobi-SVD, 63
two-sided Jacobi-SVD, 65

unreduced
Hessenberg matrix, 10

vector
principal, 17

Wielandt–Hoffman theorem, 42

zero shift QR algorithm, 98