STUDIES IN COMPUTATIONAL MATHEMATICS 2
editors: C.BREZINSKI and L.WUYTACK

EXTRAPOLATION METHODS

theory and practice

Claude **BREZINSKI**
Michela **REDIVO ZAGLIA**

NORTH-HOLLAND

EXTRAPOLATION METHODS
THEORY AND PRACTICE

STUDIES IN COMPUTATIONAL MATHEMATICS 2

Editors:

C. BREZINSKI
University of Lille
Villeneuve d'Ascq, France

L. WUYTACK
University of Antwerp
Wilrijk, Belgium

ELSEVIER
Amsterdam – Boston – London – New York – Oxford – Paris – San Diego
San Francisco – Singapore – Sydney – Tokyo

EXTRAPOLATION METHODS
THEORY AND PRACTICE

Claude BREZINSKI
Université des Sciences et Technologies de Lille
Villeneuve d'Ascq, France

Michela REDIVO ZAGLIA
Università degli Studi di Padova
Padova, Italy

ELSEVIER
Amsterdam – Boston – London – New York – Oxford – Paris – San Diego
San Francisco – Singapore – Sydney – Tokyo

ELSEVIER SCIENCE PUBLISHERS B.V.
Sara Burgerhartstraat 25
P.O. Box 211, 1000 AE Amsterdam, The Netherlands

First edition 1991
Second impression 2002

Library of Congress Cataloging-in-Publication Data

Brezinski, Claude, 1941-
Extrapolation methods : theory and practice / Claude Brezinski, Michela Redivo Zaglia.
p. cm. -- (Studies in computational mathematics ; 2)
Includes bibliographical references and index.
ISBN 0-444-88814-4
1. Extrapolation. 2. Extrapolation--Data processing. I. Redivo Zaglia, Michela. II. Title. III. Series.
QA281.B74 1991
511'.42--dc20 91-33014
CIP

This book is sold in conjunction with a diskette.

ISBN SET: 0 444 88814 4

♾ The paper used in this publication meets the requirements of ANSI/NISO Z39.48-1992 (Permanence of Paper).

Printed in The Netherlands.

PREFACE

In numerical analysis, in applied mathematics and in engineering one has often to deal with sequences and series. They are produced by iterative methods, perturbation techniques, approximation procedures depending on a parameter, and so on. Very often in practice those sequences or series converge so slowly that it is a serious drawback to their effective use. This is the reason why convergence acceleration methods have been studied for many years and applied to various situations. They are based on the very natural idea of extrapolation and, in many cases, they lead to the solution of problems which were unsolvable otherwise. Extrapolation methods now constitute a particular domain of numerical analysis having connections with many other important topics as Padé approximation, continued fractions, formal orthogonal polynomials, projection methods to name a few. They also form the basis of new methods for solving various problems and have many applications as well. Analytical methods seem to become more and more in favour in numerical analysis and applied mathematics and thus one can think (and we do hope) that extrapolation procedures will become more widely used in the future.

The aim of this book is twofold. First it is a self-contained and, as much as possible, exhaustive exposition of the theory of extrapolation methods and of the various algorithms and procedures for accelerating the convergence of scalar and vector sequences. Our second aim is to convince people working with sequences to use extrapolation methods and to help them in this respect. This is the reason why we provide many subroutines (written in FORTRAN 77) with their directions for use. We also include many numerical examples showing the effectiveness of the procedures and a quite consequent chapter on applications. In order to reduce the size of the book the proofs of the theoretical results have been omitted and replaced by references to the existing literature. However, on the other hand, some results and applications are given here for the first time. We have also included suggestions for further research.

The first chapter is a general presentation of extrapolation methods and algorithms. It does not require any special knowledge and gives the necessary prerequisites. The second chapter is devoted to the algorithms for accelerating scalar sequences. Special devices for a better use of extrapolation procedures are given in the third chapter. Chapter four deals with acceleration of vector sequences while chapter five presents the so-called continuous prediction algorithms for functional extrapolation. The sixth chapter is quite a big one. It is devoted to applications of extrapolation methods which range from the solution of systems of equations, differential equations, quadratures to problems in statistics. The last chapter presents the subroutines. They have been written in order to be as portable as possible and can be found on the floppy disk included in this book with the main programs and the numerical results. They are all new.

We intend to write a book which can be of interest for researchers in the field and to those needing to use extrapolation methods for solving a particular problem. We also hope that it can be used for graduate courses on the subject.

It is our pleasure to thank our colleagues and students who participate, directly or not, to the preparation of this monograph. In particular some of them read the manuscript or parts of it and made several important comments. We would like to specially express our gratitude to M. Calvani, G. F. Cariolaro, F. Cordellier, A. Draux, B. Germain-Bonne, A. M. Litovsky, A. C. Matos, S. Paszkowski, M. Pichat, M. Piñar, M. Prévost, V. Ramirez, H. Sadok, A. Sidi and J. van Iseghem.

We would like to thank M. Morandi Cecchi for inviting C. Brezinski to the University of Padova for a one month stay during which the book was completed.

A special thank is also due to F. J. van Drunen, J. Butterfield and A. Carter from North-Holland Publishing Company for their very efficient assistance in the preparation of the book and to M. Agnello, A. Calore and R. Lazzari from the University of Padova who typed the textual part of the manuscript.

Claude Brezinski
Université des Sciences et Technologies de Lille

Michela Redivo Zaglia
Università degli Studi di Padova

CONTENTS

This page intentionally left blank

Chapter 1

INTRODUCTION TO THE THEORY

1.1 First steps

The aim of this chapter is to be an introduction to convergence acceleration methods which are usually obtained by an extrapolation procedure.

Let (S_n) be a sequence of (real or complex) numbers which converges to S. We shall transform the sequence (S_n) into another sequence (T_n) and denote by T such a transformation.

For example we can have

$$T_n = \frac{S_n + S_{n+1}}{2}, \qquad n = 0, 1, \ldots$$

or

$$T_n = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}, \qquad n = 0, 1, \ldots$$

which is the well-known Δ^2 process due to Aitken [6].

In order to present some practical interest the new sequence (T_n) must exhibit, at least for some particular classes of convergent sequences (S_n), the following properties

1. (T_n) must converge
2. (T_n) must converge to the same limit as (S_n)
3. (T_n) must converge to S faster than (S_n), that is $\lim_{n\to\infty} (T_n - S)/(S_n - S) = 0$.

- In the case 2, we say that the transformation T is *regular* for the sequence (S_n).

- In the case 3, we say that
  the transformation T *accelerates the convergence* of the sequence (S_n) or that
  the sequence (T_n) *converges faster* than (S_n).

Usually these properties do not hold for all converging sequences (S_n) and, in particular, the last one since, as proved by Delahaye and Germain-Bonne [139], a universal transformation T accelerating all the converging sequences cannot exist (this question will be developed in section 1.10). This negative result also holds for some classes of sequences such as the set of monotone sequences or that of logarithmic sequences (that is such that $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=1$). Thus, this negative result means that it will be always interesting to find and to study new sequence transformations since, in fact, each of them is only able to accelerate the convergence of certain classes of sequences.

What is now the answer to the first two above mentioned properties? The first example was a linear transformation for which it is easy to see that, for all converging sequence (S_n), the sequence (T_n) converges and has the same limit as (S_n). Such linear transformations, called summation processes, have been widely studied and the transformations named after Euler, Cesaro, Hausdorff, Abel and others, are well known. The positive answer to properties 1 and 2 above for all convergent sequences is a consequence of the so-called Toeplitz theorem which can be found in the literature and whose conditions are very easily checked in practice. Some summation processes are very powerful for some sequences as is the case with Romberg's method for accelerating the trapezoidal rule which is explained in any textbook of numerical analysis. However let us look again at our first transformation and try to find the class of sequences which it accelerates. We have

$$\frac{T_n-S}{S_n-S}=\frac{1}{2}\left(1+\frac{S_{n+1}-S}{S_n-S}\right)$$

and thus

$$\lim_{n\to\infty}(T_n-S)/(S_n-S)=0$$

if and only if

$$\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=-1$$

which shows that this transformation is only able to accelerate the convergence of a very restricted class of sequences. This is mainly the case for all summation processes.

On the other hand, let us now look at our second sequence transformation which is Aitken's Δ^2 process. It can be easily proved that it accelerates the convergence of all the sequences for which it exists $\lambda \in [-1, +1[$ such that

$$\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = \lambda$$

which is a much wider class than the class of sequences accelerated by our first linear transformation. But, since in mathematics as in life nothing can be obtained without pain, the drawback is that the answer to properties 1 and 2 is no more positive for all convergent sequences. Examples of convergent sequences (S_n) for which the sequence (T_n) obtained by Aitken's process has two accumulation points, are known, see section 2.3. But it can also be proved that if such a (T_n) converges, then its limit is the same as the limit of the sequence (S_n), see Tucker [440].

In conclusion, nonlinear sequence transformations usually have better acceleration properties than linear summation processes (that is, they accelerate wider classes of sequences) but, on the other hand, they do not always transform a convergent sequence into another converging sequence and, even if so, both limits can be different.

In this book we shall be mostly interested by nonlinear sequence transformations. Surveys on linear summation processes were given by Joyce [253], Powell and Shah [361] and Wimp [465]. One can also consult Wynn [481], Wimp [463, 464], Niethammer [334], Gabutti [168], Gabutti and Lyness [170] and Walz [451] among others where interesting developments and applications of linear sequence transformations can be found.

There is another problem which must be mentioned at this stage. When using Aitken's process, the computation of T_n uses S_n, S_{n+1} and S_{n+2}. For some sequences it is possible that $\lim_{n\to\infty} (T_n - S)/(S_n - S) = 0$ and that $\lim_{n\to\infty} (T_n - S)/(S_{n+1} - S)$ or $\lim_{n\to\infty} (T_n - S)/(S_{n+2} - S)$ be different from zero. In particular if $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = 0$ then (T_n) obtained by Aitken's process converges faster than (S_n) and (S_{n+1}) but not always faster than (S_{n+2}). Thus, in the study of a sequence transformation, it would be better to look at the ratio $(T_n - S)/(S_{n+k} - S)$

where S_{n+k} is the term with the greatest index used in the computation of T_n. However it must be remarked that

$$\frac{T_n - S}{S_{n+k} - S} = \frac{T_n - S}{S_n - S} \cdot \frac{S_n - S}{S_{n+1} - S} \cdot \ldots \cdot \frac{S_{n+k-1} - S}{S_{n+k} - S}$$

which shows that if $(T_n - S)/(S_n - S)$ tends to zero and if $(S_{n+1} - S)/(S_n - S)$ is always away from zero and do not tend to it, then the ratio $(T_n - S)/(S_{n+k} - S)$ also tends to zero. In practice, avoiding a null limit for $(S_{n+1} - S)/(S_n - S)$ is not a severe restriction since, in such a case, (S_n) converges fast enough and does not need to be accelerated.

We shall now exemplify some interesting properties of sequence transformations on our two preceding examples. In the study of a sequence transformation the first question to be asked and solved (before those of convergence and acceleration) is an algebraic one: it concerns the so-called kernel of the transformation that is the set of sequences for which $\exists S$ such that $\forall n, T_n = S$ (in the sequel $\forall n$ would eventually mean $\forall n \geq N$).

For our linear summation process it is easy to check that its kernel is the set of sequences of the form

$$S_n = S + a(-1)^n$$

where a is a scalar.

For Aitken's process the kernel is the set of sequences of the form

$$S_n = S + a\lambda^n$$

where a and λ are scalars with $a \neq 0$ and $\lambda \neq 1$.

Thus, obviously, the kernel of Aitken's process contains the kernel of the first linear summation process.

As we can see, in both cases, the kernel depends on some (almost) arbitrary parameters, S and a in the first case, S, a and $\lambda(\neq 1)$ in the second.

If the sequence (S_n) to be accelerated belongs to the kernel of the transformation used then, by construction, we shall have $\forall n, T_n = S$.

Of course, usually, S is the limit of the sequence (S_n) but this is not always the case and the question needs to be studied. For example, in Aitken's process, S is the limit of (S_n) if $|\lambda| < 1$. If $|\lambda| > 1, (S_n)$ diverges and S is often called its anti-limit. If $|\lambda| = 1, (S_n)$ has no limit at all

or it only takes a finite number of distinct values and S is, in this case, their arithmetical mean.

The two above expressions give the explicit form of the sequences belonging to the respective kernels of our transformations. For that reason we shall call them the explicit forms of the kernel.
However the kernel can also be given in an implicit form that is by means of a relation which holds among consecutive terms of the sequence. Thus, for the first transformation, it is equivalent to write that, $\forall n$

$$S_{n+1} - S = -(S_n - S)$$

while, for Aitken's process, we have $\forall n$

$$S_{n+1} - S = \lambda(S_n - S).$$

Such a difference equation (see Lakshmikantham and Trigiante [270]) is called the implicit form of the kernel because it does not give directly (that is explicitly) the form of the sequences belonging to the kernel but only implicitly as the solution of this difference equation. Solving this difference equation, which is obvious in our examples, leads to the explicit form of the kernel. Of course, both forms are equivalent and depend on parameters.

We are now ready to enter into the details and to explain what an extrapolation method is.

1.2 What is an extrapolation method?

As we saw in the previous section, the implicit and explicit forms of the kernel of a sequence transformation depend on several parameters $S, a_1, \ldots, a_p$. The explicit form of the kernel explicitly gives the expression of the sequences of the kernel in terms of the unknown parameters which can take (almost) arbitrary values. The implicit form of the kernel consists in a relation among consecutive terms of the sequence, involving the unknown parameters $a_1, \ldots, a_p$ and S, that is a relation of the form

$$R(S_n, \ldots, S_{n+q}, S) = 0$$

which must be satisfied $\forall n$, if and only if (S_n) belongs to the kernel $\mathcal{K}_T$ of the transformation T.

A sequence transformation $T : (S_n) \longmapsto (T_n)$ is said to be an extrapolation method if it is such that $\forall n, T_n = S$ if and only if $(S_n) \in \mathcal{K}_T$.

Thus any sequence transformation can be viewed as an extrapolation method.

What is the reason for this name? Of course, it comes from interpolation and we shall now explain how a transformation T is built from its kernel $\mathcal{K}_T$ that is from the given relation R.

$S_n, S_{n+1}, \ldots, S_{n+p+q}$ being given we are looking for the sequence $(u_n) \in \mathcal{K}_T$ satisfying the interpolation conditions

$$u_i = S_i, \quad i = n, \ldots, n+p+q.$$

Then, since (u_n) belongs to $\mathcal{K}_T$, it satisfies the relation R that is, $\forall i$

$$R(u_i, \ldots, u_{i+q}, S) = 0.$$

But, thanks to the interpolation conditions we also have

$$R(S_i, \ldots, S_{i+q}, S) = 0$$

for $i = n, \ldots, n+p$. This is a system of $(p+1)$ equations with $(p+1)$ unknowns $S, a_1, \ldots, a_p$ whose solution depends on n, the index of the first interpolation condition. We shall solve this system to obtain the value of the unknown S which, since it depends on n, will be denoted by T_n (sometimes to recall that it also depends on $k = p+q$ it will be called $T_k^{(n)}$).

In order for the preceding system to have a solution we assume that the derivative of R with respect to the last variable is different from zero which guarantees, by the implicit function theorem, the existence of a function G (depending on the unknown parameters $a_1, \ldots, a_p$) such that

$$S = G(S_i, \ldots, S_{i+q})$$

for $i = n, \ldots, n+p$. The solution $T_n = S$ of this system depends on $S_n, \ldots, S_{n+k}$ and thus we have

$$T_n = F(S_n, \ldots, S_{n+k}).$$

Let us give an example to illustrate our purpose. We assume that R has the following form

$$R(u_i, u_{i+1}, S) = a_1(u_i - S) + a_2(u_{i+1} - S) = 0, \quad a_1 \cdot a_2 \neq 0,$$

and thus we have to solve the system

$$\begin{cases} a_1(S_n - S) + a_2(S_{n+1} - S) = 0 \\ a_1(S_{n+1} - S) + a_2(S_{n+2} - S) = 0. \end{cases}$$

Since this system does not change if each equation is multiplied by a non zero constant then a_1 and a_2 are not independent and the system corresponds to $p = q = 1$.
The derivative of R with respect to its last variable is equal to $-(a_1 + a_2)$ which is assumed to be different from zero.
Then G is given by

$$S = (a_1 u_i + a_2 u_{i+1})/(a_1 + a_2)$$

and the system to be solved becomes

$$\begin{cases} S = (a_1 S_n + a_2 S_{n+1})/(a_1 + a_2) \\ S = (a_1 S_{n+1} + a_2 S_{n+2})/(a_1 + a_2). \end{cases}$$

Thus we do not restrict the generality if we assume that $a_1 + a_2 = 1$ and the system writes

$$\begin{cases} S = a_1 S_n + (1 - a_1) S_{n+1} \\ S = a_1 S_{n+1} + (1 - a_1) S_{n+2} \end{cases}$$

or

$$0 = a_1 \Delta S_n + (1 - a_1) \Delta S_{n+1}$$

where Δ is the difference operator defined by $\Delta v_n = v_{n+1} - v_n$ and $\Delta^{k+1} v_n = \Delta^k v_{n+1} - \Delta^k v_n$.

The last relation gives

$$a_1 = \Delta S_{n+1} / \Delta^2 S_n$$

($\Delta^2 S_n \neq 0$ since $a_1 + a_2 \neq 0$) and thus we finally obtain

$$S = T_n = \frac{\Delta S_{n+1}}{\Delta^2 S_n} \cdot S_n + \left(1 - \frac{\Delta S_{n+1}}{\Delta^2 S_n}\right) \cdot S_{n+1}$$

that is

$$T_n = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}$$

which is Aitken's Δ^2 process (whose name becomes from the Δ^2 in the denominator).

Thus we saw how to construct a sequence transformation T from a given kernel $\mathcal{K}_T$. By construction $\forall n, T_n = S$ if and only if $(S_n) \in \mathcal{K}_T$. Sometimes it can happen that the starting point of a study is the sequence transformation T and one has to look for its kernel. This is, in particular, the case when a new sequence transformation is obtained by modifying another one. Usually this approach is much more difficult than the direct approach explained above (see section 1.4).

1.3 What is an extrapolation algorithm?

Let us come back to the example of Aitken's Δ^2 process given in the preceding section. We saw that the system to be solved for constructing T is

$$\begin{cases} T_n = S = a_1 S_n + (1 - a_1) S_{n+1} \\ T_n = S = a_1 S_{n+1} + (1 - a_1) S_{n+2}. \end{cases}$$

Adding and subtracting S_n to the first equation and S_{n+1} to the second one, leads to the equivalent system

$$\begin{cases} S_n &= T_n + b\Delta S_n \\ S_{n+1} &= T_n + b\Delta S_{n+1} \end{cases}$$

where $b = a_1 - 1$.

We have to solve this system for the unknown T_n. Using the classical determinantal formulæ giving the solution of a system of linear equations we know that T_n can be written as a ratio of determinants

$$T_n = \frac{\begin{vmatrix} S_n & S_{n+1} \\ \Delta S_n & \Delta S_{n+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta S_n & \Delta S_{n+1} \end{vmatrix}}.$$

Of course the computation of a determinant of dimension 2 is well known and easy to perform and in the preceding case we obtain

$$T_n = \frac{S_n \Delta S_{n+1} - S_{n+1} \Delta S_n}{\Delta S_{n+1} - \Delta S_n} = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}$$

which is again the formula of Aitken's process.

Let us now take a more complicated example to illustrate the problems encountered in our approach. We assume now that R has the form

$$a_1(u_i - S) + a_2(u_{i+1} - S) + \cdots + a_{k+1}(u_{i+k} - S) = 0$$

with $a_1 \cdot a_{k+1} \neq 0$. We now set $p = q = k$. For $k = 1$, the kernel of Aitken's process is recovered. Performing the same procedure as above (assuming that $a_1 + \cdots + a_{k+1} = 1$ since this sum has to be different from zero) leads to the system

$$\left\{\begin{array}{lcl} S_n & = & T_n + b_1\Delta S_n + \cdots + b_k\Delta S_{n+k-1} \\ S_{n+1} & = & T_n + b_1\Delta S_{n+1} + \cdots + b_k\Delta S_{n+k} \\ \vdots & & \vdots \\ S_{n+k} & = & T_n + b_1\Delta S_{n+k} + \cdots + b_k\Delta S_{n+2k-1}. \end{array}\right.$$

Solving this system by the classical determinantal formulæ gives for T_n

$$T_n = \frac{\begin{vmatrix} S_n & S_{n+1} & \cdots & S_{n+k} \\ \Delta S_n & \Delta S_{n+1} & \cdots & \Delta S_{n+k} \\ \vdots & \vdots & & \vdots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ \Delta S_n & \Delta S_{n+1} & \cdots & \Delta S_{n+k} \\ \vdots & \vdots & & \vdots \\ \Delta S_{n+k-1} & \Delta S_{n+k} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}.$$

In that case T_n will be denoted as $e_k(S_n)$. It is a well known sequence transformation due to Shanks [392].

The computation of $e_k(S_n)$ needs the computation of two determinants of dimension $(k+1)$ that is about $2(k+1)(k+1)!$ multiplications. For $k = 9$ this is more than $7 \cdot 10^7$ multiplications. Thus even if these determinants can be calculated in a reasonable time, the result obtained will be, in most cases, vitiated by an important error due to the computer's arithmetic. This is a well known objection to the computation of determinants which, together with the prohibitive time, leads numerical analysts to say that they don't know how to compute determinants.

If the determinants involved in the definition of T_n have some special structures, as is the case for Shanks' transformation, then it is possible to

obtain some rules (that is an algorithm) for computing recursively these ratios of determinants. Such an algorithm will be called an extrapolation algorithm. The implementation of some sequence transformations needs the knowledge of a corresponding extrapolation algorithm because their definition involves determinants; this is the case for Shanks' transformation. Some other transformations as Aitken's process, do not need such an algorithm because their direct implementation is easier and even obvious.

For implementing Shanks' transformation, it is possible to use the ε-algorithm of Wynn [470] whose rules are the following

$$\varepsilon_{-1}^{(n)} = 0\,, \quad \varepsilon_0^{(n)} = S_n\,, \quad n = 0, 1, \ldots$$

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}\,, \quad k, n = 0, 1, \ldots$$

and we have

$$\varepsilon_{2k}^{(n)} = e_k(S_n)$$

the ε's with an odd lower index being intermediate quantities without any interesting meaning.

The ε-algorithm is one of the most important extrapolation algorithms and it will be studied in section 2.3. Let us mention that Shanks' transformation can also be implemented via other extrapolation algorithms.

As we shall see below, many sequence transformations are defined as a ratio of determinants and thus they need an extrapolation algorithm for their practical implementation. As explained above, such an algorithm usually allows to compute recursively the $T_k^{(n)}$'s. It is obtained, in most cases, by applying determinantal identities to the ratio of determinants defining $T_k^{(n)}$. Although they are now almost forgotten, these determinantal identities are well known and they are named after their discoverers: Gauss, Cauchy, Kronecker, Jacobi, Binet, Laplace, Muir, Cayley, Bazin, Schur, Sylvester, Schweins, ... the last three being the most important ones for our purpose. The interested reader is referred to the paper by Brualdi and Schneider [105] which is on these questions.

There is a case where it is quite easy to construct the sequence transformation T and the corresponding algorithm from a given kernel. It is when the kernel is the set of sequences of the form

$$S_n = S + a_n d_n\,, \quad n = 0, 1, \ldots$$

where (a_n) and (d_n) are sequences such that a linear operator P satisfying

$$P(au_n + b) = aP(u_n) + b$$

for all sequences (u_n) and all a and b and such that, $\forall n$

$$P(a_n) = 0$$

is known.

In this case we have

$$(S_n - S)/d_n = a_n\,, \quad n = 0, 1, \ldots$$

and thus, applying P to both sides gives

$$P((S_n - S)/d_n) = P(a_n) = 0\,, \quad n = 0, 1, \ldots$$

It follows, from the properties of P, that $\forall n$

$$P(S_n/d_n) - S \cdot P(1/d_n) = 0$$

and thus the sequence transformation $T : (S_n) \longmapsto (T_n)$ defined by

$$T_n = P(S_n/d_n)/P(1/d_n)\,, \quad n = 0, 1, \ldots$$

is such that $\forall n, T_n = S$ if and only if $\forall n, S_n = S + a_n d_n$.

For example if $\forall n, a_n = a$ then the operator P can be taken as the forward difference operator Δ and the algorithm for the practical implementation of the transformation T is obvious.

If a_n is a polynomial in n of degree $(k-1)$ then P can be taken as Δ^k. This approach, due to Weniger [457], will be used in section 2.7 where examples of such a situation will be given.

1.4 Quasi-linear sequence transformations

We previously saw that a sequence transformation $T : (S_n) \longmapsto (T_n)$ has the form

$$T_n = F(S_n, S_{n+1}, \ldots, S_{n+k}), \quad n = 0, 1, \ldots$$

Of course, this can also be written as

$$T_n = S_n + D_n, \quad n = 0, 1, \ldots$$

with $D_n = F(S_n, S_{n+1}, \ldots, S_{n+k}) - S_n$ and we have

$$\frac{T_n - S}{S_n - S} = 1 - \frac{D_n}{S - S_n}.$$

Thus a necessary and sufficient condition that $\lim_{n\to\infty}(T_n - S)/(S_n - S) = 0$ is that $\lim_{n\to\infty} D_n/(S - S_n) = 1$.

Such a sequence (D_n) is called a perfect estimation of the error of (S_n) and accelerating the convergence is equivalent to finding a perfect estimation of the error. This very simple observation opened a new approach to extrapolation methods since perfect estimations of the error can be sometimes obtained from the usual convergence criteria for sequences and series, an approach introduced by Brezinski [77] and developed much further by Matos [311]. We shall come back later (see sections 3.1 and 3.2) to this idea but for the moment we shall use it to explain the usefulness of the so-called property of quasi-linearity that almost all the sequence transformations possess.

Let us consider the sequence $(S'_n = S_n + b)$ where b is a constant. If (S_n) converges to S, then (S'_n) converges to $S' = S + b$. If (D_n) is a perfect estimation of the error of (S_n), then it is also a perfect estimation of the error of (S'_n) since $S' - S'_n = S - S_n$. As we saw above D_n depends on $S_n, \ldots, S_{n+k}$. Thus it will be interesting for D_n to remain unchanged if $S_n, \ldots, S_{n+k}$ are replaced by $S_n + b, \ldots, S_{n+k} + b$. In that case D_n is said to be *invariant by translation* and if we denote by (T'_n) the sequence obtained by applying the transformation T to (S'_n) then we have

$$T'_n = S'_n + D_n = S_n + b + D_n = T_n + b$$

and we say, in that case, that T (or F) is *translative*.

Now let us consider the sequence $(S'_n = aS_n)$ where a is a non-zero constant. If (S_n) converges to S, then (S'_n) converges to $S' = aS$. If (D_n) is a perfect estimation of the error of (S_n), then (aD_n) is a perfect estimation of the error of (S'_n) since $S' - S'_n = a(S - S_n)$. Thus it will be interesting that D_n becomes aD_n when $S_n, \ldots, S_{n+k}$ are replaced by $aS_n, \ldots, aS_{n+k}$. In that case D_n is said to be *homogeneous* of degree one (or shortly, homogeneous) and if we denote by (T'_n) the sequence obtained by applying the transformation T to (S'_n) we have

$$T'_n = S'_n + aD_n = aS_n + aD_n = aT_n$$

which shows that T (or F) is also homogeneous.

Gathering both properties gives

$$T(aS_n + b) = aT(S_n) + b.$$

Sequence transformations which are translative and homogeneous are called *quasi-linear*, and we saw that this property is a quite natural requirement. Moreover it gives rise to other important properties that we shall now describe.

F is a real function of $(k+1)$ variables, that is an application of $\mathbf{R}^{k+1}$ into $\mathbf{R}$. We shall assume that it is defined and translative on $A \subset \mathbf{R}^{k+1}$ that is $\forall (x_0, \ldots, x_k) \in A, \forall b \in \mathbf{R}$

$$F(x_0 + b, \ldots, x_k + b) = F(x_0, \ldots, x_k) + b,$$

and that it is twice differentiable on A. Let f be another application of $\mathbf{R}^{k+1}$ into A, defined and twice differentiable on A. Then we have the following important characterization

Theorem 1.1

A necessary and sufficient condition that F be translative on A is that there exists f such that F can be written as

$$F(x_0, \ldots, x_k) = \frac{f(x_0, \ldots, x_k)}{Df(x_0, \ldots, x_k)}$$

with $D^2 f(x_0, \ldots, x_k)$ identically zero on A and where D is the operator $D = \partial/\partial x_0 + \cdots + \partial/\partial x_k$.

This result started from a remark, made by Benchiboun [23] that almost all the sequence transformations have the form f/Df. Then the reason for that particular form was found and studied by Brezinski [81]. It is also possible to state this result by saying that a necessary and sufficient condition for the translativity of F on A is that DF be identically equal to one on A.

For Aitken's Δ^2 process we have

$$f(x_0, x_1, x_2) = x_0 x_2 - x_1^2.$$

Thus $\partial f/\partial x_0 = x_2, \partial f/\partial x_1 = -2x_1, \partial f/\partial x_2 = x_0$ and then $Df = x_2 - 2x_1 + x_0$ which shows that this process is translative since $\partial Df/\partial x_0 = 1, \partial Df/\partial x_1 = -2, \partial Df/\partial x_2 = 1$ and then $D^2 f = 1 - 2 + 1 = 0$.

In section 1.2, we explained how F was obtained from the implicit form of the kernel, that is from R, via another function G. We shall now relate the properties of translativity and invariance by translation (that is $g(x_0 + b, \ldots, x_k + b) = g(x_0, \ldots, x_k)$) of F, R and G. These results, although important, are not vital for the understanding of the book and they can be skipped for a first lecture.

First it is easy to see that a necessary and sufficient condition for a function g to be invariant by translation on A is that Dg be identically zero on A. Moreover if g has the form $g(x_0, \ldots, x_k) = h(\Delta x_0, \ldots, \Delta x_{k-1})$ then it is invariant by translation. Using these two results we can obtain the

Theorem 1.2

A necessary and sufficient condition that R be invariant by translation is that G be translative.

It is easy to check that this result is satisfied by Aitken's process since

$$\begin{aligned} R(x_0, x_1, S) &= a_1(x_0 - S) + a_2(x_1 - S) \\ G(x_0, x_1) &= a_1 x_0 + (1 - a_1)x_1. \end{aligned}$$

The invariance by translation of R, used in theorem 1.2, is an invariance with respect to all its variable that is $x_0, \ldots, x_q$ and S which means

$$R(x_0 + b, \ldots, x_q + b, S + b) = R(x_0, \ldots, x_q, S).$$

Then the translativity of F can be studied from that of G. Since G is translative, the system to be solved for obtaining the unknown parameters $a_1, \ldots, a_p$ is invariant by translation and thus we have the

Theorem 1.3

If G is translative then so is F.

The reciprocal of this result is false. A counter example will be given in section 2.1.

Let us now study the property of homogeneity. We recall that a function g is said to be homogeneous of degree $r \in \mathbb{N}$ if $\forall a \in \mathbb{R}$, $g(ax_0, \ldots, ax_k) = a^r g(x_0, \ldots, x_k)$ and that a result due to Euler holds in that case

$$r \cdot g(x_0, \ldots, x_k) = \sum_{i=0}^{k} x_i g'_i$$

where g'_i denotes the partial derivative of g with respect to x_i. We have the

Theorem 1.4

If f is homogeneous of degree r, then $F = f/Df$ is homogeneous (of degree one).

For Aitken's process, it is easy to see that f is homogeneous of degree 2.

Interesting consequences on the form of F can be deduced from these results.

Theorem 1.5

If f is homogeneous of degree r and if $F = f/Df$ is translative, then

$$\begin{aligned} F &= \sum_{i=0}^{k} x_i f'_i \Big/ r \sum_{i=0}^{k} f'_i \\ &= (r-1) \sum_{i=0}^{k} x_i f'_i \Big/ r \sum_{i=0}^{k} Df'_i \\ &= \sum_{i=0}^{k} x_i F'_i \end{aligned}$$

where f'_i and F'_i denote the partial derivatives with respect to x_i.

These three formulæ can be easily checked for Aitken's process.

When $r = 1$, the first formula is of the barycentric type thus generalizing a well known form for the interpolation polynomial (which is also quasi-linear).

The last formula shows that T_n can be written as a combination of $S_n, \ldots, S_{n+k}$ whose coefficients are not constants but depend also on $S_n, \ldots, S_{n+k}$.

From the results of theorem 1.5, it can be seen that F can be written as

$$F(x_0, \ldots, x_k) = x_0 + (x_1 - x_0) \cdot g(x_0, \ldots, x_k)$$

where g is invariant by translation. Thus we have

$$D_n = (S_{n+1} - S_n) \cdot g(S_n, \ldots, S_{n+k}).$$

More precisely we have

$$g(x_0,\ldots,x_k) = h\left(\frac{\Delta x_1}{\Delta x_0},\ldots,\frac{\Delta x_{k-1}}{\Delta x_{k-2}}\right)$$

which shows that, for a linearly convergent sequence (see section 1.12), that is a sequence such that $\exists a \neq 1$ and $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S) = \lim_{n\to\infty}\Delta S_{n+1}/\Delta S_n = a$, (D_n) is a perfect estimation of the error if and only if $h(a,\ldots,a) = (1-a)^{-1}$. Thus we recover the results given by Germain-Bonne [181] which generalize those of Pennacchi [355] for rational transformations. Other acceleration results for linear sequences will be given later.

For Aitken's process we have

$$\frac{x_0x_2 - x_1^2}{x_2 - 2x_1 + x_0} = x_0 - \frac{(x_1-x_0)^2}{x_2-2x_1+x_0} = x_0 + \Delta x_0 \cdot \frac{1}{1-\Delta x_1/\Delta x_0}.$$

A relation between homogeneity and translativity does not seem to hold. Some functions F are homogeneous but not translative ($F = x_0^2/x_1$) while others are translative but not homogeneous ($F = 1+(x_0+x_1)/2$).

At the end of section 1.3, the question of finding the kernel of a transformation (that is the relation R) from the transformation T (that is from the function F) was raised. We are now able to answer this question. We have the

Theorem 1.6

Let T be a quasi-linear sequence transformation. $(S_n) \in \mathcal{K}_T$ if and only if, $\forall n$

$$\sum_{i=0}^{k}(S_{n+i}-S)\cdot f_i'(S_n-S,\ldots,S_{n+k}-S) = 0$$

or if and only if, $\forall n$

$$\sum_{i=0}^{k}(S_{n+i}-S)\cdot F_i'(S_n-S,\ldots,S_{n+k}-S) = 0.$$

It must be remarked that, since F is translative, F_i' is invariant by translation and thus, in the second condition, $F_i'(S_n-S,\ldots,S_{n+k}-S)$ can be replaced by $F_i'(S_n,\ldots,S_{n+k})$.

The first condition applied to Aitken's process gives $(S_n - S)/(S_{n+2} - S) = (S_{n+1} - S)^2$ that is $\exists a \neq 1$ such that $\forall n, (S_{n+1} - S)/(S_n - S) = a$ which is the relation defining the kernel as seen above.

The second condition leads to the same result but is more difficult to use.

The expressions given in theorem 1.5 can be used to obtain convergence and acceleration results thus answering the questions raised in section 1.1. We have the

Theorem 1.7

Let (S_n) converge to S. If $\exists M \geq 0$ such that $\forall n, |F_i'(S_n, \ldots, S_{n+k})| \leq M$ for $i = 0, \ldots, k$ then (T_n) converges to S.

For convergence acceleration, we have to consider the ratio

$$\frac{T_n - S}{S_n - S} = \sum_{i=0}^{k} \frac{S_{n+i} - S}{S_n - S} \cdot F_i'(S_n, \ldots, S_{n+k}).$$

In the important case where (S_n) converges linearly, that is when $\exists a \neq 1$ such that $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = a$, then obviously if $\exists A_0, A_1, \ldots, A_k$ such that $\lim_{n\to\infty} F_i'(S_n, \ldots, S_{n+k}) = A_i$ and $A_0 + A_1 a + \cdots + A_k a^k = 0$ then (T_n) converges faster than (S_n). However, thanks to the quasi-linearity of F, more complete results can be obtained

Theorem 1.8

Let F be quasi-linear and (S_n) be a linearly converging sequence.

If $Df(1, a, \ldots, a^k) \neq 0$ and if $\exists M \geq 0$ such that $|f(1, a, \ldots, a^k)| \leq M$, then $\lim_{n\to\infty} T_n = S$. Moreover if $f(1, a, \ldots, a^k) = 0$, then (T_n) converges faster than (S_n).

For Aitken's process we have $Df(1, a, a^2) = a^2 - 2a + 1$ which is different from zero if and only if $a \neq 1$ and $f(1, a, a^2) = 1 \cdot a^2 - (a^2) = 0$ which shows that Aitken's process accelerates linear convergence, a well known result.

It can also be seen that the condition $f(1, a, \ldots, a^k) = 0$ is a necessary and sufficient condition that $\forall n, T_n = S$ if (S_n) is a geometric progression which means that $\forall n, (S_{n+1} - S)/(S_n - S) = a$ or, equivalently, $S_n = S + ba^n$. Thus, in that case, geometric progressions belong to the kernel of T. If geometric progressions belong to the kernel of a transformation T, then T accelerates the convergence of linearly converging sequences.

If F is homogeneous then $F(ax_0,\ldots,ax_k) = aF(x_0,\ldots,x_k)$ and thus $F(0,\ldots,0) = 0$. If F is not defined at the point $(0,\ldots,0)$ we shall set $F(0,\ldots,0) = 0$. Since F is translative, we have $F(0+b,\ldots,0+b) = F(0,\ldots,0) = 0+b$ and thus $F(b,\ldots,b) = b$. If F is not defined at the point $(b,\ldots,b)$ we shall set $F(b,\ldots,b) = b$. More generally, if F is not defined at the point $(x_0,\ldots,x_k)$ we shall set $F(x_0,\ldots,x_k) = x_m$ where m is an arbitrary integer between 0 and k (usually 0 or k). However it must be noticed and clearly understood that this convention does not insure the continuity of F.

Transformations which are homogeneous in the limit were recently defined by Graça [188]. They can be useful in some cases. They are only at an early stage of development and this is the reason why they will not be presented here.

1.5 Sequence transformations as ratios of determinants

Most of the sequence transformations actually known can be expressed as ratios of determinants. There are two reasons for that: the first one is, let us say, a mechanical one depending on the way such transformations are usually built while the second one is much more profound since it relates such a determinantal formula with the recursive scheme used for implementing the transformation. The first reason will be examined in this section and the second one in the next section.

Let us come back to the beginning of the section 1.3 where we explained how to construct T_n from G in Aitken's Δ^2 process. We show that we had to solve the system

$$\begin{cases} S_n &= T_n + b\Delta S_n \\ S_{n+1} &= T_n + b\Delta S_{n+1}. \end{cases}$$

Of course if the second equation is replaced by its difference with the first one, we get the equivalent system

$$\begin{cases} S_n &= T_n + b\Delta S_n \\ \Delta S_n &= 0 + b\Delta^2 S_n. \end{cases}$$

Similarly in the system written for Shanks' transformation we can replace each equation, from the second one, by its difference with the

preceding one and the system becomes

$$\begin{cases} S_n & = & T_n + b_1\Delta S_n + \cdots + b_k\Delta S_{n+k-1} \\ \Delta S_n & = & 0 + b_1\Delta^2 S_n + \cdots + b_k\Delta^2 S_{n+k-1} \\ \vdots & & \vdots \\ \Delta S_{n+k-1} & = & 0 + b_1\Delta^2 S_{n+k-1} + \cdots + b_k\Delta^2 S_{n+2k-2}. \end{cases}$$

Thus we have

$$T_n = e_k(S_n) = \frac{\begin{vmatrix} S_n & \Delta S_n & \cdots & \Delta S_{n+k-1} \\ \Delta S_n & \Delta^2 S_n & \cdots & \Delta^2 S_{n+k-1} \\ \vdots & \vdots & & \vdots \\ \Delta S_{n+k-1} & \Delta^2 S_{n+k-1} & \cdots & \Delta^2 S_{n+2k-2} \end{vmatrix}}{\begin{vmatrix} 1 & 0 & \cdots & 0 \\ \Delta S_n & \Delta^2 S_n & \cdots & \Delta^2 S_{n+k-1} \\ \vdots & \vdots & & \vdots \\ \Delta S_{n+k-1} & \Delta^2 S_{n+k-1} & \cdots & \Delta^2 S_{n+2k-2} \end{vmatrix}}.$$

This ratio of determinants and the ratio given in section 1.3 have the common form

$$R_k = \frac{\begin{vmatrix} e_0 & \cdots & e_k \\ a_1^{(0)} & \cdots & a_1^{(k)} \\ \vdots & & \vdots \\ a_k^{(0)} & \cdots & a_k^{(k)} \end{vmatrix}}{\begin{vmatrix} c_0 & \cdots & c_k \\ a_1^{(0)} & \cdots & a_1^{(k)} \\ \vdots & & \vdots \\ a_k^{(0)} & \cdots & a_k^{(k)} \end{vmatrix}}$$

with $e_i = S_{n+i}, a_j^{(i)} = \Delta S_{n+i+j-1}$ and $c_i = 1$ for the ratio of section 1.3 and with $e_0 = S_n, a_i^{(0)} = \Delta S_{n+i-1}, c_0 = 1$ and $e_i = \Delta S_{n+i-1}, a_j^{(i)} = \Delta^2 S_{n+i+j-2}, c_i = 0$ for $i \geq 1$ for the preceding ratio.

If the determinant in the numerator of R_k is developed with respect to its first row then it is easy to see that

$$R_k = \alpha_0 e_0 + \cdots + \alpha_k e_k$$

where the α_i's are solution of the system

$$\begin{cases} \alpha_0 c_0 + \cdots + \alpha_k c_k & = & 1 \\ \alpha_0 a_1^{(0)} + \cdots + \alpha_k a_1^{(k)} & = & 0 \\ \vdots & & \vdots \\ \alpha_0 a_k^{(0)} + \cdots + \alpha_k a_k^{(k)} & = & 0. \end{cases}$$

This interpretation of R_k shows that our ratio of determinants can be extended to the case where the e_i's are vectors (or more generally elements of a vector space), the $a_i^{(j)}$'s and the c_i's being scalars. In that case, the determinant in the numerator of R_k denotes the vector obtained by expanding it with respect to its first row using the classical rule for that purpose.

If the e_i's are scalars, then R_k is also given as the solution of the system

$$\begin{cases} c_0 R_k + \beta_1 a_1^{(0)} + \cdots + \beta_k a_k^{(0)} & = & e_0 \\ \vdots & & \vdots \\ c_k R_k + \beta_1 a_1^{(k)} + \cdots + \beta_k a_k^{(k)} & = & e_k. \end{cases}$$

Let us assume that the c_i's and the $a_i^{(j)}$'s are invariant by translation on the e_i's. Thus, so are the α_i's and the β_i's and, if we set $R_k = \sum_{i=0}^{k} \alpha_i e_i = F(e_0, \ldots, e_k)$, we have

$$F(e_0 + b, \ldots, e_k + b) = \sum_{i=0}^{k} \alpha_i e_i + b \sum_{i=0}^{k} \alpha_i.$$

Thus F is invariant by translation (that is, in other words, $F(1,\ldots,1)= 1$) if and only if $\sum_{i=0}^{k} \alpha_i = 1$. This property is true if $c_0 = \cdots = c_k = 1$, or if $c_0 = 1$ and $c_1 = \cdots = c_k = 0$ and if, in addition

$$\begin{vmatrix} 0 & 1 & \cdots & 1 \\ a_1^{(0)} & a_1^{(1)} & \cdots & a_1^{(k)} \\ \vdots & \vdots & & \vdots \\ a_k^{(0)} & a_k^{(1)} & \cdots & a_k^{(k)} \end{vmatrix} = 0.$$

These two particular cases will be of interest in the sequel.

Such a ratio of determinants for R_k includes almost all the sequence transformations (for sequences of scalar and sequences of vectors) that will be studied in this book. It also includes other important related topics such as Padé approximants, orthogonal polynomials and fixed point methods.

We still have to see how to compute recursively these ratios of determinants. There are two possibilities for such an algorithm: its normal form or its progressive form. They will be discussed in sections 1.7 and 1.8 respectively but let us now explain why ratios of determinants and recursive schemes are obtained.

1.6 Triangular recursive schemes

A unified treatment of sequence transformations in relation with their determinantal expressions and triangular recursive schemes for their implementation was given by Brezinski and Walz [104]. In fact this theory goes far beyond extrapolation methods since it includes in the same framework B-splines, Bernstein polynomials, orthogonal polynomials, divided differences and certainly many other topics.

Let us consider sequence transformations of the form

$$T_k^{(n)} = \sum_{i=0}^{k} \alpha_{k,i}^{(n)} S_{n+i}, \qquad k, n = 0, 1, \ldots$$

with $T_0^{(n)} = S_n$ for all n, where the α's are real or complex numbers which can depend on some terms of the sequence (S_n) itself.

The theory we shall now present includes the case where the S_n's are vectors or, more generally, elements of a vector space F. Let E be a vector space on $K = \mathbb{R}$ or $\mathbb{C}$. Usually E will be $\mathbb{R}$ or $\mathbb{C}$. Let $z_0, z_1, \ldots$ be elements of E and let us denote by $\mathcal{A}(E, F)$ the vector space of (possibly nonlinear) applications from E to F. In the sequel σ will arbitrarily designate either an element of $\mathcal{A}(E, F)$ or an element of $\mathcal{A}(E, K)$. We consider the linear functional on $\mathcal{A}(E, F)$ or $\mathcal{A}(E, K)$ defined by

$$T_k^{(n)}(\sigma) = \sum_{i=0}^{k} \alpha_{k,i}^{(n)} \sigma(z_{n+i})$$

for all k and n.

If $L \in \mathcal{A}(E,F)$ and if we set $S_n = L(z_n)$ then clearly $T_k^{(n)}(L) = T_k^{(n)}$.

We have the following fundamental result

Theorem 1.9

Let $\sigma_0, \ldots, \sigma_k$ be elements of $\mathcal{A}(E,K)$ such that

$$\begin{vmatrix} \sigma_0(z_n) & \cdots & \sigma_0(z_{n+k}) \\ \vdots & & \vdots \\ \sigma_k(z_n) & \cdots & \sigma_k(z_{n+k}) \end{vmatrix} \neq 0$$

and

$$\begin{aligned} T_k^{(n)}(\sigma_0) &= w_k^{(n)} \\ T_k^{(n)}(\sigma_i) &= 0 \quad \text{for } i = 1, \ldots, k \end{aligned}$$

where the $w_k^{(n)}$'s are arbitrary nonzero real or complex numbers.

Then

$$T_k^{(n)}(\sigma) = \frac{\begin{vmatrix} \sigma(z_n) & \cdots & \sigma(z_{n+k}) \\ \sigma_1(z_n) & \cdots & \sigma_1(z_{n+k}) \\ \vdots & & \vdots \\ \sigma_k(z_n) & \cdots & \sigma_k(z_{n+k}) \end{vmatrix}}{\begin{vmatrix} \sigma_0(z_n) & \cdots & \sigma_0(z_{n+k}) \\ \sigma_1(z_n) & \cdots & \sigma_1(z_{n+k}) \\ \vdots & & \vdots \\ \sigma_k(z_n) & \cdots & \sigma_k(z_{n+k}) \end{vmatrix}} \cdot w_k^{(n)}.$$

In particular all the quasi-linear transformations of the form

$$T_k^{(n)} = F_k(S_n, \ldots, S_{n+k})$$

fit into this framework since, as explained in section 1.4, we have

$$T_k^{(n)} = \sum_{i=0}^{k} F'_{k,i}(S_n, \ldots, S_{n+k})\, S_{n+i}$$

with $F'_{k,i}(u_0, \ldots, u_k) = \dfrac{\partial}{\partial u_i} F_k(u_0, \ldots, u_k)$ for $i = 0, \ldots, k$.

The $T_k^{(n)}(\sigma)$'s can be recursively computed since we have the

Theorem 1.10

If $\forall k, n$ *and for* $i = 0$ *and* 1, *we have*

$$\begin{vmatrix} \sigma_i(z_n) & \cdots & \sigma_i(z_{n+k}) \\ \vdots & & \vdots \\ \sigma_{i+k}(z_n) & \cdots & \sigma_{i+k}(z_{n+k}) \end{vmatrix} \neq 0$$

then

$$T_k^{(n)}(\sigma) = \lambda_k^{(n)} T_{k-1}^{(n)}(\sigma) + \mu_k^{(n)} T_{k-1}^{(n+1)}(\sigma)$$

with $T_0^{(n)}(\sigma) = \sigma(z_n)$ *and*

$$\lambda_k^{(n)} = w_k^{(n)} T_{k-1}^{(n+1)}(\sigma_k) \Big/ d_k^{(n)}$$

$$\mu_k^{(n)} = -w_k^{(n)} T_{k-1}^{(n)}(\sigma_k) \Big/ d_k^{(n)}$$

$$d_k^{(n)} = w_{k-1}^{(n)} T_{k-1}^{(n+1)}(\sigma_k) - w_{k-1}^{(n+1)} T_{k-1}^{(n)}(\sigma_k).$$

It is easy to see that if $\forall n, k, \sum_{i=0}^{k} \alpha_{k,i}^{(n)} = \alpha_k$ then $\forall n, k, \lambda_k^{(n)} + \mu_k^{(n)} = \gamma_k$ and $\gamma_0 = \alpha_0 = 1$, $\alpha_k = \prod_{i=1}^{k} \gamma_i$.

Reciprocally if we consider a triangular recursion scheme of the form

$$T_k^{(n)} = \lambda_k^{(n)} T_{k-1}^{(n)} + \mu_k^{(n)} T_{k-1}^{(n)}, \qquad k = 1, 2, \ldots; \; n = 0, 1, \ldots$$

with $T_0^{(n)} = S_n$, then

$$T_k^{(n)} = \sum_{i=0}^{k} \alpha_{k,i}^{(n)} S_{n+i}$$

with

$$\alpha_{k,0}^{(n)} = \lambda_k^{(n)} \alpha_{k-1,0}^{(n)}$$

$$\alpha_{k,i}^{(n)} = \lambda_k^{(n)} \alpha_{k-1,i}^{(n)} + \mu_k^{(n)} \alpha_{k-1,i-1}^{(n+1)}, \qquad i = 1, \ldots, k-1$$

$$\alpha_{k,k}^{(n)} = \mu_k^{(n)} \alpha_{k-1,k-1}^{(n+1)}$$

and $\alpha_{0,0}^{(n)} = 1$ for all n.

Moreover if $\forall n, k$, $\lambda_k^{(n)} + \mu_k^{(n)} = \gamma_k$ then $\forall n, k$, $\sum_{i=0}^{k} \alpha_{k,i}^{(n)} = \alpha_k$ with $\gamma_0 = \alpha_0 = 1$ and $\alpha_k = \prod_{i=1}^{k} \gamma_i$. If, in addition, $w_0^{(n)} = w_0$ for all n then $\forall n, w_k^{(n)} = \alpha_k w_0$.

The algorithm given in the preceding theorem is very close to the E-algorithm which will be studied in section 2.1. It is recovered by setting $E_k^{(n)} = T_k^{(n)}(\sigma) \big/ w_k^{(n)}$ and $g_{k,i}^{(n)} = T_k^{(n)}(\sigma_i) \big/ w_k^{(n)}$. It is also connected to the H-algorithm in the case of vector sequences, see section 4.5 below.

From theorem 2.1 we immediately obtain the

Theorem 1.11

A necessary and sufficient condition that $\forall n$, $T_k^{(n)} \big/ w_k^{(n)} = S$ *is that* $\forall n$, $S_n = S\sigma_0(z_n) + a_1\sigma_1(z_n) + \cdots + a_k\sigma_k(z_n)$.

We recall that $\sigma_0(z_n) = w_0^{(n)}$ and thus these quantities are different from zero.

Let us now look at the possible existence of other recurrence relations of the form

$$T_k^{(n)}(\sigma) = a_k^{(n)} T_{k-i}^{(n+p)}(\sigma) + b_k^{(n)} T_{k-j}^{(n+q)}(\sigma)$$

with $0 < i < j$ and $p, q \in \mathbf{Z}$.

By taking successively $\sigma = \sigma_0, \ldots, \sigma_k$ it is easy to see that such a relation can only hold for $i = j = 1$ and that, in that case we have

$$\begin{aligned} a_k^{(n)} &= w_k^{(n)} T_{k-1}^{(n+q)}(\sigma_k) \big/ d_k^{(n)} \\ b_k^{(n)} &= -w_k^{(n)} T_{k-1}^{(n+p)}(\sigma_k) \big/ d_k^{(n)} \\ d_k^{(n)} &= w_{k-1}^{(n+p)} T_{k-1}^{(n+q)}(\sigma_k) - w_{k-1}^{(n+q)} T_{k-1}^{(n+p)}(\sigma_k) \end{aligned}$$

provided that $d_k^{(n)} \neq 0$.

However we have

$$T_{k+m}^{(n)}(\sigma) = \frac{\begin{vmatrix} T_k^{(n)}(\sigma) & \cdots & T_k^{(n+m)}(\sigma) \\ T_k^{(n)}(\sigma_{k+1}) & \cdots & T_k^{(n+m)}(\sigma_{k+1}) \\ \vdots & & \vdots \\ T_k^{(n)}(\sigma_{k+m}) & \cdots & T_k^{(n+m)}(\sigma_{k+m}) \end{vmatrix}}{\begin{vmatrix} w_k^{(n)} & \cdots & w_k^{(n+m)} \\ T_k^{(n)}(\sigma_{k+1}) & \cdots & T_k^{(n+m)}(\sigma_{k+1}) \\ \vdots & & \vdots \\ T_k^{(n)}(\sigma_{k+m}) & \cdots & T_k^{(n+m)}(\sigma_{k+m}) \end{vmatrix}} \cdot w_{k+m}^{(n)}$$

and a similar formula by inverting k and m. For $k = 0$, this is the first determinantal formula given above. For $m = 1$, it reduces to the triangular recursive scheme given in theorem 1.10. For an arbitrary value of m this formula allows to jump over a breakdown (due to a division by zero) or a near-breakdown.

Let us now show, on a very simple example, how to find the σ_i's from the recursive scheme. It must be noticed first that these σ_i's do not depend on k. Thus, when passing from $k-1$ to k we keep the same $\sigma_0, \ldots, \sigma_{k-1}$ which have already been determined and we only have to find σ_k by writing that

$$T_k^{(n)}(\sigma_i) = 0 \quad \text{for} \quad i = 1, \ldots, k.$$

This leads to a difference equation of order k with $k-1$ solutions already known $(\sigma_0, \ldots, \sigma_{k-1})$ and we have to find its last solution σ_k. Usually this difference equation is nonlinear and has non-constant coefficients and the task of finding σ_k is a difficult one.

Aitken's Δ^2 process can be written as

$$T_1^{(n)} = \frac{S_n \, \Delta S_{n+1} - S_{n+1} \, \Delta S_n}{\Delta S_{n+1} - \Delta S_n}.$$

Taking $z_n = n$ we have

$$T_1^{(n)}(\sigma) = \frac{\sigma(n)\,\Delta S_{n+1} - \sigma(n+1)\,\Delta S_n}{\Delta S_{n+1} - \Delta S_n}.$$

Let us take $\sigma_0(n) = 1$ for all n. Thus $T_1^{(n)}(\sigma_0) = 1$. We see that if σ_1 is defined by $\sigma_1(n) = \Delta S_n$ then $T_1^{(n)}(\sigma_1) = 0$ and thus we have

$$T_1^{(n)} = \frac{\begin{vmatrix} S_n & S_{n+1} \\ \Delta S_n & \Delta S_{n+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta S_n & \Delta S_{n+1} \end{vmatrix}}$$

and we recover the formula given in section 1.3.

Let us say that $T_k^{(n)}(\sigma)$ can also be expressed as a complex contour integral. On this question see Walz [451] for the case of linear extrapolation methods.

1.7 Normal forms of the algorithms

In sequence transformations, the ratio of determinants R_k of section 1.5 depends on a second index n since the e_i's and the $a_i^{(j)}$'s depend on n. Thus, as explained in section 1.2, let us denote it by $T_k^{(n)}$ (or by $e_k(S_n)$ in the case of Shanks' transformation).

By applying determinantal identities to the numerator and the denominator of $T_k^{(n)}$ it is possible to obtain algorithms for computing recursively these $T_k^{(n)}$'s without explicitly computing the determinants involved in their definition. Such a recursive algorithm for Shanks' transformation was given in section 1.3; it was the so-called ε-algorithm. Since we already know it, let us take it as an example but the situation will be similar for the recursive algorithms corresponding to the other sequence transformations that will be studied later.

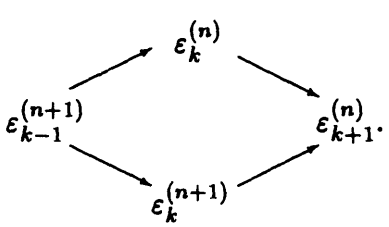
The numbers $\varepsilon_k^{(n)}$ computed by the ε-algorithm are displayed in a

double entry table as follows

$$
\begin{array}{lllll}
\varepsilon_{-1}^{(0)}=0 & & & & \\
& \varepsilon_{0}^{(0)}=S_0 & & & \\
\varepsilon_{-1}^{(1)}=0 & & \varepsilon_{1}^{(0)} & & \\
& \varepsilon_{0}^{(1)}=S_1 & & \varepsilon_{2}^{(0)} & \\
\varepsilon_{-1}^{(2)}=0 & & \varepsilon_{1}^{(1)} & & \varepsilon_{3}^{(0)} \\
& \varepsilon_{0}^{(2)}=S_2 & & \varepsilon_{2}^{(1)} & \ddots \\
\varepsilon_{-1}^{(3)}=0 & & \varepsilon_{1}^{(2)} & & \varepsilon_{3}^{(1)} \\
\vdots & \varepsilon_{0}^{(3)}=S_3 & & \varepsilon_{2}^{(2)} & \ddots \\
\vdots & \vdots & \varepsilon_{1}^{(3)} & & \varepsilon_{3}^{(2)} \\
\vdots & \vdots & \vdots & \varepsilon_{2}^{(3)} & \ddots \\
\vdots & \vdots & \vdots & \vdots & \varepsilon_{3}^{(3)} \\
\vdots & \vdots & \vdots & \vdots & \vdots \quad \ddots
\end{array}
$$

Notice that, in this table, the lower index k denotes a column, the upper index (n) denotes a descending diagonal and that the sum of the lower and upper indexes is constant among an ascending diagonal.

Starting from the first two columns $\left(\varepsilon_{-1}^{(n)}=0\right)$ and $\left(\varepsilon_{0}^{(n)}=S_n\right)$, the rule of the ε-algorithm allows to proceed in this table from left to right and from top to bottom. This rule relates quantities located at the four corners of a rhombus



Thus, knowing $\varepsilon_{k-1}^{(n+1)}, \varepsilon_{k}^{(n)}$ and $\varepsilon_{k}^{(n+1)}$ it is possible to compute $\varepsilon_{k+1}^{(n)}$ as

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_{k}^{(n+1)} - \varepsilon_{k}^{(n)}}.$$

This is the normal form of the ε-algorithm, when proceeding in this way in the table.

However this normal form can suffer from a very serious drawback: cancellation errors due to the computer's arithmetic. The better the ε-algorithm works, the worse are rounding errors. The reason is easy to understand. We saw that the numbers $\varepsilon_{2k}^{(n)}$ are approximations of the limit S of the sequence (S_n) (it is the purpose of any extrapolation method to furnish such approximations) while the $\varepsilon_{2k+1}^{(n)}$ are intermediate results. If the algorithm works very well, then both $\varepsilon_{2k}^{(n)}$ and $\varepsilon_{2k}^{(n+1)}$ are very good approximations of S. Thus when computing $\varepsilon_{2k+1}^{(n)}$, an important cancellation error will occur in the difference $\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}$. Thus $\varepsilon_{2k+1}^{(n)}$ will be large and badly computed. If $\varepsilon_{2k}^{(n+2)}$ is also close to S then, for the same reason, $\varepsilon_{2k+1}^{(n+1)}$ will be large and badly computed also. After that, we want to compute $\varepsilon_{2k+2}^{(n)}$ by

$$\varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + \frac{1}{\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}}$$

and we have, in the denominator, the difference of two large and badly computed quantities thus producing numerical instability in the algorithm.

For example if the ε-algorithm is applied to the sequence $S_n = 1/(n+1)$ then it can be proved that $\varepsilon_{2k}^{(n)} = 1/(k+1)(n+k+1)$. This example is thus very interesting for controlling the numerical stability of the algorithm since all the answers are known.
For $\varepsilon_{24}^{(0)}$ we obtain (on a computer working with 15 decimal figures) $0.591715 \cdot 10^{-2}$ instead of $0.626850 \cdot 10^{-2}$.

It is possible to avoid, to some extent, such a numerical instability by using either the progressive form of the algorithm or its particular rules. These two possibilities will be now studied.

1.8 Progressive forms of the algorithms

Let us assume that, in the ε-algorithm, the first descending diagonal is known: $\varepsilon_0^{(0)}, \varepsilon_1^{(0)}, \varepsilon_2^{(0)}, \varepsilon_3^{(0)}, \varepsilon_4^{(0)}, \ldots$. Then, writing the rule of the ε-algorithm as

$$\varepsilon_{k+1}^{(n+1)} = \varepsilon_{k+1}^{(n)} + \frac{1}{\varepsilon_{k+2}^{(n)} - \varepsilon_k^{(n+1)}}$$

allows to compute all the quantities in the table from the first diagonal $\left(\varepsilon_k^{(0)}\right)$ and the second column $\left(\varepsilon_0^{(n)} = S_n\right)$.

Of course this rule still suffers from numerical instability since, when k is even, we have to compute the difference of two almost equal quantities. However the instability is not so severe since, usually, $\varepsilon_{2k+2}^{(n)}$ is a better approximation of S than $\varepsilon_{2k}^{(n+1)}$ and both quantities have less digits in common than $\varepsilon_{2k}^{(n)}$ and $\varepsilon_{2k}^{(n+1)}$ as was the case in the normal form of the ε-algorithm.

Coming back to the example $S_n = 1/(n+1)$, the following conclusions hold

- For the normal form
  - $\varepsilon_{2k}^{(n)}$ and $\varepsilon_{2k}^{(n+1)}$ have $\log_{10}(n+k+2)$ common decimal digits
  - $\varepsilon_{2k-1}^{(n+1)}$ and $1/\left(\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}\right)$ have $-\log_{10}(1-2/k)$ common digits
  - $\varepsilon_{2k+1}^{(n)}$ and $\varepsilon_{2k+1}^{(n+1)}$ have $-\log_{10}(2/(n+k+3))$ common digits
  - $\varepsilon_{2k}^{(n+1)}$ and $1/\left(\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}\right)$ have opposite signs.
- For the progressive form
  - $\varepsilon_{2k}^{(n)}$ and $\varepsilon_{2k+2}^{(n-1)}$ have $\log_{10}(k+2)$ common decimal digits
  - $\varepsilon_{2k+1}^{(n-1)}$ and $1/\left(\varepsilon_{2k+2}^{(n-1)} - \varepsilon_{2k}^{(n)}\right)$ have $-\log_{10}(1-2/(n+k))$ common digits
  - $\varepsilon_{2k+1}^{(n)}$ and $\varepsilon_{2k+3}^{(n-1)}$ have $-\log_{10}(2/(k+3))$ common digits
  - $\varepsilon_{2k+2}^{(n+1)}$ and $1/\left(\varepsilon_{2k+3}^{(n-1)} - \varepsilon_{2k+1}^{(n)}\right)$ have opposite signs.

Thus all the computations conducted with the progressive form are more stable than the computations realized with the normal form (at least for this example).

We have now to solve another problem: how to compute the first diagonal $\left(\varepsilon_k^{(0)}\right)$? Let us first see how to obtain the subsequence $\left(\varepsilon_{2k}^{(0)}\right)$. In section 1.3 and in section 1.5 we saw that these quantities can be in fact obtained as the first unknown of a system of linear equations. In both cases, although the systems were not identical, the situation was the same: the system to be solved for computing $e_{k+1}(S_0) = \varepsilon_{2k+2}^{(0)}$ can

be obtained by adding one equation and one unknown to the system giving $e_k(S_0) = \varepsilon_{2k}^{(0)}$. Or, in other words, the matrix of the system to be solved has been bordered by a new row and a new column. It is well known in numerical analysis that such a bordered system can be solved recursively by using the solution of the initial system: it is the so-called bordering method (see Faddeeva [154]) which will be now explained.

Let A_k be a regular square matrix of dimension k and d_k a vector of dimension k . Let z_k be the solution of the system

$$A_k z_k = d_k.$$

Let now u_k be a column vector of dimension k, v_k a row vector of dimension k and a_k a scalar. We consider the bordered matrix A_{k+1} of dimension $k+1$ given by

$$A_{k+1} = \begin{pmatrix} A_k & u_k \\ v_k & a_k \end{pmatrix}.$$

We have

$$A_{k+1}^{-1} = \begin{pmatrix} A_k^{-1} + A_k^{-1} u_k v_k A_k^{-1}/\beta_k & -A_k^{-1} u_k/\beta_k \\ -v_k A_k^{-1}/\beta_k & 1/\beta_k \end{pmatrix}$$

with $\beta_k = a_k - v_k A_k^{-1} u_k$.

Let f_k be a scalar and z_{k+1} be the solution of the bordered system

$$A_{k+1} z_{k+1} = d_{k+1} = \begin{pmatrix} d_k \\ f_k \end{pmatrix}.$$

Then we have

$$z_{k+1} = \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \frac{f_k - v_k z_k}{\beta_k} \cdot \begin{pmatrix} -A_k^{-1} u_k \\ 1 \end{pmatrix}.$$

This formula gives the solution of the bordered system in terms of the solution of the initial system. However its use needs the computation and the storage of A_k^{-1}. This drawback can be avoided by setting $q_k = -A_k^{-1} u_k$ and computing it recursively by the same bordering method. Thus we finally obtain the following algorithm.

Let $q_k^{(i)}$ be the solution of the system

$$A_i q_k^{(i)} = -u_k^{(i)}$$

where $u_k^{(i)}$ is the vector formed by the first i components of u_k.
Thus $u_i^{(i)} = u_i$ and $q_i^{(i)} = q_i$ for all i. A_i is the matrix of dimension i formed by the first i rows and columns of A_k.

We have since A_1 is a number

$$q_k^{(1)} = -\frac{u_k^{(1)}}{A_1}$$

$$q_k^{(i+1)} = \begin{pmatrix} q_k^{(i)} \\ 0 \end{pmatrix} - \frac{u_{k,i+1} + v_i q_k^{(i)}}{a_i + v_i q_i^{(i)}} \cdot \begin{pmatrix} q_i^{(i)} \\ 1 \end{pmatrix}, \quad i = 1, \ldots, k-1$$

where $u_{k,i+1}$ is the $(i+1)$th component of u_k. And then $q_k^{(k)} = q_k = -A_k^{-1} u_k$ thus allowing to use the previous formula for z_{k+1}. It must be noticed that this variant of the bordering method needs the storage of A_k instead of that of A_k^{-1} for the original procedure. On these questions see Brezinski [86].
The subroutine BORDER performs this variant of the bordering method.

Of course, the bordering method can only be applied if $\beta_k \neq 0$ for all k. When it is not the case a block bordering procedure has to be used as explained by Brezinski, Redivo Zaglia and Sadok [99]. We now assume that the dimensions of A_k are $n_k \times n_k$, u_k are $n_k \times p_k$, v_k are $p_k \times n_k$ and a_k are $p_k \times p_k$. Thus the dimensions of A_{k+1} are $n_{k+1} \times n_{k+1}$ with $n_{k+1} = n_k + p_k$. We set

$$\beta_k = a_k - v_k A_k^{-1} u_k$$

and we have now

$$A_{k+1}^{-1} = \begin{pmatrix} A_k^{-1} + A_k^{-1} u_k \beta_k^{-1} v_k A_k^{-1} & -A_k^{-1} u_k \beta_k^{-1} \\ -\beta_k^{-1} v_k A_k^{-1} & \beta_k^{-1} \end{pmatrix}.$$

f_k is now a vector with p_k components and we obtain

$$z_{k+1} = \begin{pmatrix} z_k \\ 0 \end{pmatrix} + \begin{pmatrix} -A_k^{-1} u_k \\ I_k \end{pmatrix} \beta_k^{-1} (f_k - v_k z_k)$$

where I_k is the identity matrix of dimensions $p_k \times p_k$.

The subroutine BLBORD performs this block bordering method.

Again $q_k = -A_k^{-1}u_k$ (whose dimensions are $n_k \times p_k$) can be recursively computed by the bordering method.

Let $u_k^{(i)}$ be the $n_i \times p_k$ matrix formed by the first n_i rows of u_k for $i \leq k$, $n_i \leq n_k$. We have $u_i^{(i)} = u_i$. Let $q_k^{(i)}$ be the $n_i \times p_k$ matrix satisfying $A_i q_k^{(i)} = -u_k^{(i)}$ for $i \leq k$. We have $q_i^{(i)} = q_i$.

We set

$$q_k^{(1)} = -A_1^{-1}\, u_k^{(1)}$$

and then we have

$$q_k^{(i+1)} = \begin{pmatrix} q_k^{(i)} \\ 0 \end{pmatrix} - \begin{pmatrix} q_i^{(i)} \\ I_i \end{pmatrix} \beta_i^{-1} \left(u_{k,i+1} + v_i q_k^{(i)} \right), \quad i = 1, \ldots, k-1$$

with $\beta_i = a_i + v_i q_i^{(i)}$ and $u_{k,i+1}$ the matrix formed by the rows $n_i + 1, \ldots, n_i + p_i$ of u_k.

Thus, the algorithm is as follows

1. Set $k = 1$ and $n_1 = 1$.
2. Compute A_1.
3. If A_1 is non-singular, compute z_1 and go to step 4.
   If A_1 is singular, border the system by the next row and the next column, increase n_1 by 1 and go to step 2.
4. Increase k by 1 for solving the next system (if wanted).
5. Set $p_k = 1$.
6. Compute β_k.
7. If β_k is non-singular, compute z_{k+1} and go to step 4.
   If β_k is singular, border the system by the next row and the next column, increase p_k by 1 and go to step 6.

Of course a test for checking the numerical singularity of a matrix is needed. The permutation-perturbation method of La Porte and Vignes [269] is particularly well adapted for that purpose.

Instead of using the previous block bordering method it is possible to use a pivoting strategy. If, for some k, $\beta_k = 0$ then the last row of the

matrix is interchanged with the next one and so on until some $\beta_k \neq 0$ has been obtained. Such a procedure is not adapted to our case where the intermediate solutions are desired and have to be computed. It can be used only if the last solution is needed but not the intermediate ones.

The bordering method can be applied to the computation of R_k of section 1.5. In that case it simplifies since $d_1 = 1$ and $f_k = 0$ for $k \geq 1$. The vector z_{k+1} thus obtained has components $\alpha_0, \ldots, \alpha_k$ and then we have $R_k = \alpha_0 e_0 + \cdots + \alpha_k e_k$ which shows how to use the bordering method when the e_i's are vectors.

Sometimes, due to some special structure of the matrices A_k, the general bordering method as explained above can be made more efficient. This is, in particular, the case for Shanks' transformation where the matrices A_k are Hankel matrices. Let us consider the system

$$\begin{cases} a_0 S_n + a_1 S_{n+1} + \cdots + a_k S_{n+k} &= 1 \\ a_0 S_{n+1} + a_1 S_{n+2} + \cdots + a_k S_{n+k+1} &= 1 \\ \vdots & \vdots \\ a_0 S_{n+k} + a_1 S_{n+k+1} + \cdots + a_k S_{n+2k} &= 1. \end{cases}$$

It was proved by Brezinski [35] that

$$e_k(S_n) = \varepsilon_{2k}^{(n)} = \frac{1}{\sum_{i=0}^{k} a_i}.$$

Two very efficient bordering methods for solving this system were given by Brezinski [54] with the corresponding subroutines.

Now, before being able to apply the progressive form of the ε-algorithm, we need to compute the subsequence $\left(\varepsilon_{2k+1}^{(0)}\right)$. This can be done again with the help of the bordering method since the auxiliary quantities $\varepsilon_{2k+1}^{(n)}$ are in fact related to Shanks' transformation by

$$\varepsilon_{2k+1}^{(n)} = \frac{1}{e_k(\Delta S_n)}.$$

Thus, replacing S_n by ΔS_n in the preceding system and using the bordering methods, leads to the recursive computation of $e_k(\Delta S_0)$ and thus to $\varepsilon_{2k+1}^{(0)}$.

However, since we are not in fact interested by the auxiliary quantities $\varepsilon_{2k+1}^{(n)}$ they can be eliminated from the rule of the ε-algorithm thus leading to the so-called cross rule due to Wynn [479] which only involves quantities with an even lower index. Setting

$$C = \varepsilon_{2k}^{(n+1)},\ N = \varepsilon_{2k}^{(n)},\ S = \varepsilon_{2k}^{(n+2)},\ W = \varepsilon_{2k-2}^{(n+2)},\ E = \varepsilon_{2k+2}^{(n)}$$

this cross rule is

$$(N-C)^{-1} + (S-C)^{-1} = (W-C)^{-1} + (E-C)^{-1}.$$

The notation with C (= center) and the cardinal points comes from the fact that, in the table of the ε-algorithm, these quantities are located as

$$\begin{array}{ccc} & N & \\ W & C & E \\ & S & \end{array}$$

The normal form of the cross rule is

$$E = C + \left[(N-C)^{-1} + (S-C)^{-1} - (W-C)^{-1}\right]^{-1}$$

with $\varepsilon_{-2}^{(n)} = \infty$. Its progressive form, which is more stable, is given by

$$S = C + \left[(W-C)^{-1} + (E-C)^{-1} - (N-C)^{-1}\right]^{-1}.$$

1.9 Particular rules of the algorithms

A crucial point in extrapolation methods is that of the propagation of cancellation errors due to the computer's arithmetic. Let us illustrate this question with Aitken's Δ^2 process. As seen before it is given by

$$T_n = \frac{S_n S_{n+2} - S_{n+1}^2}{S_{n+2} - 2S_{n+1} + S_n}, \qquad n = 0, 1, \ldots$$

However such a formula is highly numerically unstable since, if S_n,S_{n+1} and S_{n+2} are almost equal, cancellation errors arise both in the numerator and in the denominator and T_n is badly computed. Thus instead of the preceding formula we can write

$$T_n = S_n - \frac{(S_{n+1} - S_n)^2}{S_{n+2} - 2S_{n+1} + S_n}, \qquad n = 0, 1, \ldots$$

Reducing to the same denominator it is easy to see that this expression is the same as the first one. Of course in this formula cancellation errors again arise in the computation of $(S_{n+1} - S_n)^2$ and $S_{n+2} - 2\,S_{n+1} + S_n$, but the term $(S_{n+1} - S_n)^2 \big/ (S_{n+2} - 2\,S_{n+1} + S_n)$ is a correcting term to S_n and this second formula is much more stable than the first one. Thus by modifying the rule of the algorithm we were able to obtain a more stable algorithm. The second formula is a particular rule for avoiding propagation of rounding errors due to the computer's arithmetic. The conditioning of Aitken's process is discussed by Bell and Phillips [20].

Let us give two numerical examples for illustrating our purpose. We first consider the sequence

$$\begin{aligned} S_0 &= 1 \\ S_{n+1} &= \exp(-S_n), \qquad n = 0, 1, \ldots \end{aligned}$$

which converges to 0.5671432904097838. We obtain

n	stable formula	unstable formula
15	0.5671433079394927	0.5671433079394927
20	0.5671432904701356	0.5671432904701355
35	0.5671432904097838	0.5671432904101733
40	0.5671432904097838	0.5671432903972257
45	0.5671432904097838	0.5671432900920601
50	0.5671432904097838	0.5671432886117994
55	0.5671432904097838	0.5671434290968433
60	0.5671432904097838	0.5671420162671232
65	0.5671432904097838	0.5671386718750000

Let us now take the sequence $(S_n = \lambda^n)$. Thus we shall obtain $T_n = 0$ for all n even if $|\lambda| > 1$. For $n = 0$ the stable formula gives

λ	simple precision	double precision
0.997	$0.66 \cdot 10^{-2}$	$0.41 \cdot 10^{-11}$
0.9997	-0.51	$0.91 \cdot 10^{-10}$
1.0004	0.33	$-0.53 \cdot 10^{-9}$

More details about these two interesting examples will be given in section 7.2.

Let us now deal with a more complicated situation and, for this purpose, let us come back to the cross rule of the ε-algorithm and introduce

again the quantities with lower odd indexes, denoted by small letters for simplicity

$$\begin{array}{ccccc} & & N & & \\ & a & & b & \\ W & & C & & E \\ & e & & d & \\ & & S & & \end{array}$$

Using the normal form of the algorithm we have

$$\begin{aligned} C &= W + 1/(e-a) \\ b &= a + 1/(C-N) \\ d &= e + 1/(S-C) \\ E &= C + 1/(d-b)\,. \end{aligned}$$

If $N = C$, then b is infinity. If $S = C$, then d is infinity. If b and d are infinity, then E is undefined and the computations have to be stopped. There is a breakdown in the algorithm. The same is true if $a = e$ since then C is infinity. If $N \neq C$, then $b = a = e$. If $S \neq C$, then $d = a = b = e$ and E is undefined. If N is different from C but very close to it, then a cancellation error arises in the computation of b which will be large and badly computed. The same is true for d if S is different from C but close to it. If a is different from e but close to it, C will be large and badly computed. Thus b and d will be almost equal and E will be the difference of two large and badly computed numbers. There is a near-breakdown in the algorithm.

After some algebraic manipulations it can be proved that the cross rule of the ε-algorithm can be equivalently written as

$$E = r(1 + r/C)^{-1}$$

with

$$r = S(1 - S/C)^{-1} + N(1 - N/C)^{-1} - W(1 - W/C)^{-1}.$$

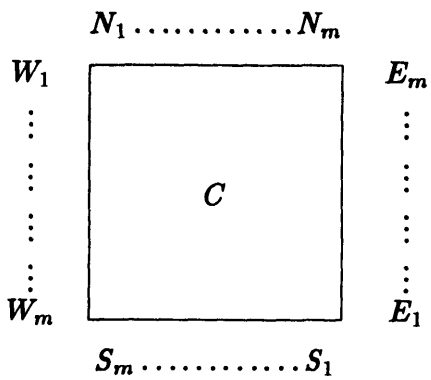
This rule was shown to be more stable than the rule given above for computing E. It is called a particular rule for the ε-algorithm. If C is infinity, it reduces to

$$E = S + N - W$$

thus allowing to compute E by jumping over the singularity (or the breakdown). This rule was obtained by Wynn [475]. It is valid when

there is only one isolated singularity that is when N and S are not infinity, or, equivalently, when only two adjacent quantities in a column (a and e in our example) are equal or almost equal. Wynn's particular rule was extended by Cordellier [119] to the case of an arbitrary number of equal quantities in the ε-algorithm.

If we have a square block of size m containing quantities all equal to C and if we set



then the cross rule become

$$(N_i - C)^{-1} + (S_i - C)^{-1} = (W_i - C)^{-1} + (E_i - C)^{-1}, \quad i = 1, \ldots, m.$$

Using the notion of Schur complement, Brezinski [79] was able to obtain particular rules for the E-algorithm (which contains many well known sequence transformations, such as Shanks', as particular cases), and for some vector sequence transformations, such as the so-called RPA, CRPA and H-algorithm, which will be discussed later. This technique allows to compute directly the elements of the column $m+k$ of the table in terms of the elements of column k without computing the intermediate columns. Thus one can jump over breakdowns or near-breakdowns in order to avoid division by zero or numerical instability due to cancellation errors.

Since Wynn's cross rule also holds between the ε's with an odd lower index, the preceding particular rules are also valid for these quantities.

All the questions concerning the numerical stability of extrapolation processes are treated in details by Cordellier [121].

It must be clearly understood that the particular rules are not the panacea for avoiding the propagation of rounding errors. Let us give two examples with Wynn's particular rules for the ε-algorithm.

Let us first consider the sequence given by

$$S_0 = 1.5999999\,, \quad S_1 = 1.2\,, \quad S_2 = 1$$

and

$$S_i = \frac{S_{i-1}}{2} + \frac{S_{i-2}}{4} + \frac{S_{i-3}}{8}\,, \qquad i = 3, 4, \ldots$$

From the theory of the ε-algorithm we know that (see theorem 2.18) $\forall n, \varepsilon_6^{(n)} = 0$.

Using the ε-algorithm without and with the particular rules gives respectively for $\varepsilon_6^{(n)}$

n	without p. r.	with p. r.
0	$2.15 \cdot 10^{-2}$	$2.22 \cdot 10^{-16}$
1	$3.18 \cdot 10^{-3}$	$-2.16 \cdot 10^{-15}$
2	$-2.51 \cdot 10^{-2}$	$2.89 \cdot 10^{-15}$
3	$-1.31 \cdot 10^{-2}$	$1.86 \cdot 10^{-15}$
4	$-1.38 \cdot 10^{-3}$	$1.05 \cdot 10^{-15}$
5	$-2.57 \cdot 10^{-15}$	$-2.57 \cdot 10^{-15}$

Let us now consider the sequence $S_n = (n+1)^{-1}$, $n = 0, 1, \ldots$. For this case it can be proved that $\forall k, n$ we must obtain

$$\varepsilon_{2k}^{(n)} = (k+1)^{-1} \cdot (n+k+1)^{-1}$$

$$\varepsilon_{2k+1}^{(n)} = -(k+1) \cdot (k+2) \cdot (n+k+1) \cdot (n+k+2).$$

Thus the precision of the whole table can be checked. For this example the results obtained without and with the particular rules are the same. Using $S_0, \ldots, S_{23}$ the results computed have 14 exact digits for $\varepsilon_2^{(21)}$, 11 for $\varepsilon_4^{(19)}$, 7 for $\varepsilon_8^{(15)}$, 3 for $\varepsilon_{16}^{(7)}$ and 2 for $\varepsilon_{23}^{(0)}$.

For both examples the particular rules were used when two successive quantities $\varepsilon_k^{(n)}$ and $\varepsilon_k^{(n+1)}$ had 5 common digits.

1.10 Accelerability and non-accelerability

As explained in section 1.1, a universal sequence transformation for accelerating the convergence of all convergent sequences cannot exist. More precisely, as proved by Delahaye and Germain-Bonne [139], a universal transformation cannot exist for a set of sequences which is remanent. In other words a transformation able to accelerate the convergence of all the sequences of a remanent set cannot exist. This is clearly a very fundamental result in the theory of sequence transformations since it shows the frontier between accelerability and non-accelerability.

A set S of real convergent sequences is said to possess the property of generalized remanence if and only if

1. There exists a convergent sequence $(\hat{S}_n)$ with limit $\hat{S}$ such that $\forall n, \hat{S}_n \neq \hat{S}$ and such that

   i) $\exists (S_n^0) \in \mathbf{S}$ such that $\lim_{n\to\infty} S_n^0 = \hat{S}_0$.

   ii) $\forall m_0 \geq 0, \exists p_0 \geq m_0$ and $(S_n^1) \in \mathbf{S}$ such that $\lim_{n\to\infty} S_n^1 = \hat{S}_1$ and $\forall m \leq p_0, S_m^1 = S_m^0$.

   iii) $\forall m_1 > p_0, \exists p_1 \geq m_1$ and $(S_n^2) \in \mathbf{S}$ such that $\lim_{n\to\infty} S_n^2 = \hat{S}_2$ and $\forall m \leq p_1, S_m^2 = S_m^1$.

   iv) .....

2. $(S_0^0, \ldots, S_{p_0}^0, S_{p_0+1}^1, \ldots, S_{p_1}^1, S_{p_1+1}^2, \ldots, S_{p_2}^2, S_{p_2+1}^3, \ldots) \in \mathbf{S}$.

The diagram in figure 1.1 makes the property more clear.

A deep understanding of this property is not necessary for the sequel. The fundamental result is

Theorem 1.12

If a set of sequences possesses the property of generalized remanence then a universal transformation able to accelerate the convergence of all its sequences cannot exist.

Such a set of sequences is said to be non-accelerable. Techniques similar but different from remanence can also be used to prove the non-accelerability of some sets of sequences. Actually many sets of sequences were proved to be non accelerable by Delahaye [138], Kowalewski [266, 267] and Delahaye and Germain-Bonne [140]. They are the following

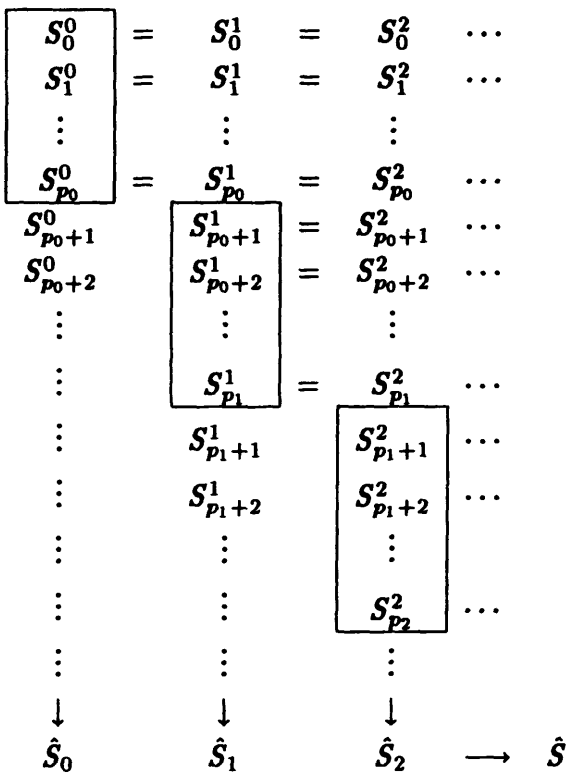


Figure 1.1: The property of generalized remanence.

- The set of convergent sequences of E, where E is a metric space. This set will be denoted $conv(E)$.
- The set of convergent sequences of E such that $\exists N, \forall n \geq N, S_n \neq \lim_{i \to \infty} S_i$. This set will be denoted $conv^*(E)$.
- The subsets of $conv(E)$ such that $\forall n, S_{n+1} \geq S_n$ or $S_{n+1} \leq S_n$ or $S_{n+1} > S_n$ or $S_{n+1} < S_n$.
- The subsets of $conv(E)$ such that $\forall n, (-1)^i \Delta^i S_n \leq 0$ for $i = 1, \ldots, k$ or $(-1)^i \Delta^i S_n \geq 0$.
- The subsets of $conv(\mathbb{R})$ such that $(-1)^n \Delta S_n$ or $(-1)^n (S_n - S)$ has a constant sign.

- The subsets of $conv(\mathbf{R})$ such that $(-1)^n \Delta S_n$ or $(-1)^n (S_n - S)$ is monotone with a constant sign.
- The subsets of $conv^*(\mathbf{R})$ such that $\forall n \geq N, 0 < \lambda \leq (S_{n+1} - S)/(S_n - S) \leq \mu < 1$ or $\lambda \leq \Delta S_{n+1}/\Delta S_n \leq \mu$.
- The subset of $conv^*(\mathbf{R})$ such that $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = 0$.
- The set of logarithmic sequences, $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = 1$. This set is called LOG.
- The subset of logarithmic sequences such that $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = \lim_{n\to\infty} \Delta S_{n+1} \,/\, \Delta S_n = 1$. This set is called LOGSF.

If must be clearly understood that the preceding results do not mean that a particular sequence belonging to a non-accelerable set cannot be accelerated. It means that the same algorithm cannot accelerate all the sequences of the set. The reason is usually because it is too *big* and one has to look for the possibility of accelerating some of its subsets. Of course for obtaining a good idea of the frontier between accelerability and non-accelerability one has to find the *smallest* non-accelerable sets and to complete these negative results by positive ones giving the *biggest* accelerable sets. Such results were also obtained by Delahaye [138] who, for example, proved the

Theorem 1.13

Let $\mathbf{S}^*(E)$ be a set of convergent sequences of elements of a metric space E such that $\forall n, S_n \neq S$ (the limit of (S_n)). A necessary and sufficient condition for $\mathbf{S}^*(E)$ to be accelerable is that E'' be the empty set, where E' designates the set of accumulation points of E and $E'' = (E')'$.

We also have the following positive results

Theorem 1.14

Let $\mathbf{S}$ be a set of convergent real sequences, let $f\mathbf{S} = \{(f(S_n))|(S_n) \in \mathbf{S}, f$ monotone, differentiable and such that $\forall x, f'(x) \neq 0\}$ and let $\mathbf{S}+\lambda = \{(S_n + \lambda)|(S_n) \in \mathbf{S}, \lambda \in \mathbf{R}\}$. Then $\mathbf{S}$ is accelerable if and only if $f\mathbf{S}$ is accelerable. $\mathbf{S}$ is accelerable if and only if $\mathbf{S} + \lambda$ is accelerable.

As we shall see in sections 3.1 and 3.5, accelerable sets of sequences can be obtained by construction of a synchronous transformation or by subsequence extraction.

The result of theorem 1.12 is very general since no assumption on the transformation is made. In particular it holds even for transformations of the form

$$T_n = F_n\left(S_0, \ldots, S_{k(n)}\right)$$

with $k(n) \geq n$ (for example $k(n) = 2n$ or $k(n) = n^n$). A remanent set of sequences cannot be accelerated by such transformations. Other sets are not accelerable only by transformations with $k(n)$ constant.

Let us mention that all the attempts to find a necessary and sufficient condition of non-accelerability failed. The property of remanence, as defined above, is only a sufficient condition of non-accelerability. But it is a very strong sufficient condition since it not only implies the non-accelerability of a remanent set but also the fact that it is impossible to improve the convergence of all its sequences which means that it cannot exist $\lambda \in]0,1[$ such that $\forall n, |T_n - S| \leq \lambda |S_n - S|$. Transformations satisfying such a property, called contractive sequence transformations, will be studied in section 3.9. A universal contractive sequence transformation for a remanent set cannot exist.

The impossibility of accelerating a remanent set of sequences is due to the definition of acceleration which was chosen. This is the reason why some other definitions of acceleration have been studied, see Jacobsen [240], Germain-Bonne [183] and Wang [456].

1.11 Optimality

Another interesting question about sequence transformations is optimality, a word which can be understood under several meanings. The first results on this question were obtained by Pennacchi [355] who considered transformations of the form

$$T_n = S_n + \frac{P\left(\Delta S_n, \ldots, \Delta S_{n+p-1}\right)}{Q\left(\Delta S_n, \ldots, \Delta S_{n+p-1}\right)}, \qquad n = 0, 1, \ldots$$

where P and Q are homogeneous polynomials of degree m and $m-1$ respectively. Such a transformation is called a rational transformation of type (p, m).

We consider the set of sequences for which there exists λ such that

$$\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=\lambda$$

with $0<|\lambda|<1$. This is the set of linearly converging sequences (the set of linear sequences, for short).

Pennacchi proved that a rational transformation of type $(1,m)$ or $(p,1)$ accelerating the convergence of all the linear sequences cannot exist and that the only rational transformation of type $(2,2)$ accelerating this set is Aitken's Δ^2 process. Moreover any rational transformation of type $(2,m)$ with $m\geq 2$ which accelerates this set is equivalent to Aitken's process which means that it gives the same sequence (T_n) (this is due to a common factor between P and Q which cancels out).

Thus Aitken's Δ^2 process is optimal in the algebraic sense since it is the simplest rational transformation which accelerates the set of linear sequences. This is confirmed by a result due to Germain-Bonne [182] which states that this set cannot be accelerated by any transformation of the form

$$T_n=S_n+g(\Delta S_n)\,,\qquad n=0,1,\ldots$$

where g is a function continuous at the point zero.

Other optimality results about Aitken's Δ^2 process were proved by Delahaye [137]. They go in two different directions: first we shall see that it is impossible to accelerate a larger set of sequences (in some sense) than the linear ones and next that it is impossible to improve the acceleration of linear sequences.

A transformation of the form

$$T_n=F_n(S_0,\ldots,S_{n+k})\,,\qquad n=0,1,\ldots$$

is said to be k-normal. For $k=0$, it is said to be normal. Thus, with this definition, Aitken's Δ^2 process is 2-normal. By a shift in the indexes, a k-normal transformation can always be changed into a normal one, since we can set $T'_{n+k}=T_n, n=0,1,\ldots$. The reason for such definitions was explained in section 1.1: for some sequences, (T_n) can converge faster than (S_n) but not faster than (S_{n+k}) when $k\geq 1$. If the computation of T_n involves S_{n+k} it is more appropriate to define acceleration with respect to (S_{n+k}) than to (S_n).

Let us now try to enlarge the set of linear sequences by weakening the condition on λ.

For example we can assume that $0 \leq |\lambda| < 1$. As proved by Delahaye [137] this set of sequences is not accelerable by any normal transformation.

Let us assume that $0 < |\lambda| \leq 1$. This set is not accelerable by any normal or k-normal transformation for all k.

Let us finally assume that $\exists 0 < \alpha < \beta < 1$, $\exists N$ such that $\forall n \geq N$, $\alpha < |S_{n+1} - S|/|S_n - S| < \beta$. This set is not accelerable by any normal or k-normal transformation for all k.

Similar results hold by replacing the ratio $(S_{n+1} - S)/(S_n - S)$ by the ratio $\Delta S_{n+1}/\Delta S_n$.

Thus we tried to enlarge, in three different ways, the set of linear sequences and proved that these extensions were not accelerable. Of course this result does not mean that other extensions are not accelerable. Examples of accelerable extensions of linear sequences will be given in the subsequent chapters.

We saw that Aitken's process accelerates the convergence of linear sequences which means that for such a sequence

$$\lim_{n\to\infty} (T_n - S)/(S_{n+2} - S) = 0.$$

We shall now try to find a transformation having better acceleration properties for linear sequences, namely a k-normal transformation such that $\exists r > 0$ with

$$\lim_{n\to\infty} |T_n - S| \Big/ |S_{n+k} - S|^{1+r} = 0.$$

In that case we shall say that the transformation accelerates (S_n) with the degree $1 + r$. Such a notion was introduced by Germain-Bonne [181] and Delahaye [137] proved that, $\forall r > 0$, a normal or a k-normal transformation accelerating linear sequences with degree $1 + r$ cannot exist. Thus Aitken's Δ^2 process is also optimal in this sense since no other transformation can produce a better acceleration for linear sequences.

These results on the degree of acceleration were refined by Trojan [439]. He considered transformations of the form

$$T_n = F_k(S_n, \ldots, S_{n+k}), \qquad n = 0, 1, \ldots$$

were F_k is a rational function independent of n. Obviously Aitken's Δ^2 process has this form. Let $X_{p,m}$ be the set of sequences such that, $\forall n$

$$\begin{aligned} S_n - S \;=\; & -\Delta S_n + a_1 (-\Delta S_n)^p + \cdots + a_{m-1} (-\Delta S_n)^{p+m-2} + \\ & (-\Delta S_n)^{p+m-1} \cdot b(-\Delta S_n) \end{aligned}$$

with $a_1 \neq 0$, $p \geq 1$ and b a bounded function in a neighborhood of zero. If $p = 1$ we assume moreover that $|a_1/(1 + a_1)| < 1$.

For $p = 1$ we have, $\forall n$

$$S_{n+1} - S = \frac{a_1}{1 + a_1} \cdot (S_n - S) + O\left((S_n - S)^2\right)$$

and for $p \geq 2$, $\forall n$

$$0 < A \leq \frac{|S_{n+1} - S|}{|S_n - S|^p} \leq B < +\infty.$$

The set $X_{p,m}$ contains in particular the sequences generated by $S_{n+1} = F(S_n)$ with F sufficiently differentiable in the neighborhood of $S = F(S)$ and such that $F'(S) = \cdots = F^{(p-1)}(S) = 0$ and $F^{(p)}(S) \neq 0$.

Trojan [439] defined the order of the transformation F_k in a class X of convergent sequences by

$$q(F_k, X) = \sup \left\{ q \mid \forall (S_n) \in X \,, \limsup_{n \to \infty} \frac{|T_n - S|}{|S_n - S|^q} < \infty \right\}.$$

Let $\Phi_k(X)$ be the set of all transformations F_k of the preceding form such that $\forall (S_n) \in X$

$$\limsup_{n \to \infty} \frac{|T_n - S|}{|S_{n+k} - S|} \leq 1.$$

Trojan [439] proved that if $F_k \in \Phi_k(X_{p,t})$ and $t \geq k$ then

$$q(F_k, X_{p,k}) \leq q_k = 1 + p + \cdots + p^{k-2} + p^k.$$

This estimate is sharp. When $t = k$, a transformation attaining this upper bound can be constructed by inverse polynomial interpolation as follows.

Let P_n be the polynomial of degree $p + k - 2$ at most defined by

$$P_n(-\Delta S_i) = S_i \,, \qquad i = n, \ldots, n + k - 1$$

and for $p \geq 2$

$$P_n'(0) = 1 \,, \quad P^{(j)}(0) = 0 \,, \qquad j = 2, \ldots, p - 1.$$

Then put

$$T_n = F_k(S_n, \ldots, S_{n+k}) = P_n(0).$$

This transformation is well defined (which means that such a P_n exists) for S_n sufficiently close to S and its order is equal to q_k. For $p = 1$ it is identical with the method proposed by Germain-Bonne [182] which consists in taking $x_n = \Delta S_n$ in Richardson extrapolation (see section 2.2).

For $p = k = 2$, the following transformation is obtained

$$T_n = S_{n+2} - \frac{(S_{n+2} - S_{n+1})^3}{(S_n + S_{n+2})(S_{n+2} - 2S_{n+1} + S_n)}, \qquad n = 0, 1, \ldots$$

Another method for measuring the acceleration of a transformation is to find a sequence (e_n) tending to zero such that

$$\limsup_{n \to \infty} \left| \frac{T_n - S}{e_n} \right| < \infty.$$

When (S_n) is generated by

$$S_{n+1} = F(S_n), \qquad n = 0, 1, \ldots$$

with $|F'(S)| < 1$ and S_0 sufficiently close to $S = F(S)$ in order to insure convergence, then lower bounds on such a sequence (e_n) were obtained by Trojan [438]. He considered transformations given by

$$T_n = F_n(S_0, \ldots, S_n), \qquad n = 0, 1, \ldots$$

Thus the set $\{F_n\}$ characterizes the transformation and he proved that for every transformation and every constant $c > 0$, there exists an analytic function F such that

$$\limsup_{n \to \infty} \frac{|T_n - S|}{2^{-cn^2}} > 0.$$

This bound is again sharp which means that there exists an algorithm such that for every analytic function F it exists c with

$$\limsup_{n \to \infty} |T_n - S| \leq 2^{-cn^2}.$$

As before this optimal algorithm is obtained by inverse polynomial interpolation.

If we now assume F to be only p-times continuously differentiable, 2^{-cn^2} has to be replaced by 2^{-cn} in the preceding inequalities.

The proofs of these results are based on the notion of remanence explained in the preceding section and on the adversary principle of Traub and Woźniakowski [436] for constructing optimal algorithms.

For the construction of rational transformations see section 2.11.

1.12 Asymptotic behaviour of sequences

Before entering into more details about sequence transformations it is not unnecessary to know some results on the asymptotic behaviour of a sequence and on the asymptotic comparison of sequences. The proofs of these results and more references can be found in Brezinski [55, 91]. Let (S_n) and (T_n) be sequences converging respectively to S and T.

Theorem 1.15

Let λ be a complex number with a modulus different from 1. *A necessary and sufficient condition that*

$$\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=\lambda$$

is that

$$\lim_{n\to\infty}\Delta S_{n+1}/\Delta S_n=\lambda.$$

As shown by counter-examples, the conditions $|\lambda|\neq 1$ cannot be removed but it can be replaced by others as we shall see now.

Theorem 1.16

If (S_n) is monotone and if there exists λ, finite or not, such that $\lim_{n\to\infty}\Delta S_{n+1}/\Delta S_n=\lambda$, then $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=\lambda$.

Theorem 1.17

If $((-1)^n\Delta S_n)$ is monotone and if there exists λ, finite or not, such that $\lim_{n\to\infty}\Delta S_{n+1}\,/\,\Delta S_n=\lambda$ and if $\lim_{n\to\infty}(1+\Delta S_{n+2}/\Delta S_{n+1})/(1+\Delta S_{n+1}/\Delta S_n)=1$ then $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=\lambda$.

Theorem 1.18

Let λ and μ be two real numbers with $0\leq\lambda<\mu<1$.

i) If, $\forall n, \lambda \leq \Delta S_{n+1}/\Delta S_n \leq \mu$, *then* $\forall n, \lambda' \leq (S_{n+1}-S)/(S_n-S) \leq \mu'$ *with* $\lambda' = \lambda(\mu-1)/(\lambda-1)$ *and* $\mu' = \mu(\lambda-1)/(\mu-1)$.

ii) If, $\forall n, \lambda \leq (S_{n+1}-S)/(S_n-S) \leq \mu$, *then* $\forall n, \lambda' \leq \Delta S_{n+1}/\Delta S_n \leq \mu'$ *with* $\lambda' = \lambda(\mu-1)/(\lambda-1)$ *and* $\mu' = \mu(\lambda-1)/(\mu-1)$.

Let us now give some results which allow to compare the asymptotic behaviour of (S_n) and (T_n). We recall that
(T_n) is said to converge faster than (S_n) if and only if

$$\lim_{n\to\infty} (T_n - S)/(S_n - S) = 0.$$

(T_n) is said to converge at the same rate as (S_n) if there exist a and b with $0 < a \leq b$, and N such that $\forall n \geq N$

$$a \leq |T_n - T| \,/\, |S_n - S| \leq b.$$

We have the

Theorem 1.19

Assume that there exist ϱ *and* λ *with* $|\varrho| < 1$ *and* $|\lambda| < 1$ *such that* $\lim_{n\to\infty} \Delta T_{n+1}/\Delta T_n = \varrho$ *and* $\lim_{n\to\infty} \Delta S_{n+1}/\Delta S_n = \lambda$.

i) (T_n) *converges faster than* (S_n) *if and only if* (ΔT_n) *converges faster than* (ΔS_n).

ii) (T_n) *converges at the same rate as* (S_n) *if and only if* (ΔT_n) *converges at the same rate as* (ΔS_n).

The faster convergence of (ΔT_n) obviously implies $|\varrho| < |\lambda|$.

Theorem 1.20

Assume that there exist ϱ *and* λ *with* $0 \leq \varrho < 1$ *and* $0 \leq \lambda < 1/2$ *and* N *such that* $\forall n \geq N, |\Delta T_{n+1}/\Delta T_n| \leq \varrho$ *and* $|\Delta S_{n+1}/\Delta S_n| \leq \lambda$.
If (ΔT_n) *converges faster than* (ΔS_n) *then* (T_n) *converges faster than* (S_n).

In this theorem, $1/2$ cannot be replaced by a greater number as shown by counter-examples.

Theorem 1.21

Assume that there exist a and b with $a < 1 < b$ and N such that $\forall n \geq N$

$$(S_{n+1} - S)/(S_n - S) \notin [a, b].$$

If there exists c such that $\lim_{n\to\infty} (T_n - S)/(S_n - S) = c$, *then* $\lim_{n\to\infty} \Delta T_n/\Delta S_n = c$.

As proved by a counter-example, the reciprocal of this theorem is not true.

Theorem 1.22

If (S_n) is monotone and if there exists c, finite or not, such that $\lim_{n\to\infty} \Delta T_n/\Delta S_n = c$, *then* $\lim_{n\to\infty} (T_n - T)/(S_n - S) = c$.

Theorem 1.23

Assume that there exist a and b with $a < 1 < b$ and N such that $\forall n \geq N$

$$(S_{n+1} - S)/(S_n - S) \notin [a, b].$$

If $|T_n - T| = O(|S_n - S|)$, *then* $|\Delta T_n| = O(|\Delta S_n|)$.

Theorem 1.24

Assume that $\limsup_{n\to\infty} |\Delta T_n|^{\frac{1}{n}} = 1/R$ *and* $\lim_{n\to\infty} |\Delta S_n|^{\frac{1}{n}} = 1/r$. *If $r < R$ then (ΔT_n) converges faster than (ΔS_n).*

If the conditions of theorem 1.19 hold then $|\varrho| = 1/R$ and $|\lambda| = 1/r$ with $|\varrho| < |\lambda|$ and (T_n) converges faster than (S_n).

Let us now generalize theorem 1.19. We have the

Theorem 1.25

Assume that there exist λ and ϱ with $|\lambda| < 1$ and $|\varrho| < 1$, such that $\lim_{n\to\infty} \Delta T_{n+1}/\Delta T_n = \varrho$ *and* $\lim_{n\to\infty} \Delta S_{n+1}/\Delta S_n = \lambda$. *Let a be a complex number. A necessary and sufficient condition that*

$$\lim_{n\to\infty} (T_n - S)/(S_n - S) = a$$

is that

$$\lim_{n\to\infty} \Delta T_n/\Delta S_n = a.$$

Of course by, theorem 1.15, the conditions of this theorem can be replaced by $\lim_{n\to\infty}(T_{n+1}-T)/(T_n-T)=\varrho$ and $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=\lambda$.

Let us now generalize theorem 1.20. We have

Theorem 1.26

Assume that there exist ϱ *and* λ *with* $0\leq\varrho<1/2$ *and* $0\leq\lambda<1/2$ *and* N *such that* $\forall n\geq N, |\Delta T_{n+1}/\Delta T_n|\leq\varrho$ *and* $|\Delta S_{n+1}/\Delta S_n|\leq\lambda$. (ΔT_n) *converges faster than* (ΔS_n) *if and only if* (T_n) *converges faster than* (S_n).

This result shows that theorem 1.20 is a necessary and sufficient condition when $\varrho<1/2$.

Let us finally give a generalization of theorem 1.18.

Theorem 1.27

Let $\alpha,\beta,\alpha',\beta'$ *be real numbers with* $0\leq\alpha<\beta<1$ *and* $0\leq\alpha'<\beta'<1$.

Assume that there exist N *such that* $\forall n\geq N, \Delta T_{n+1}/\Delta T_n\in[\alpha,\beta]$ *and* $\Delta S_{n+1}/\Delta S_n\in[\alpha',\beta']$.

i) If there exist a *and* b *with* $0\leq a\leq b$ *such that* $\forall n\geq N, \Delta T_n/\Delta S_n\in[a,b]$ *then* $\forall n\geq N, (T_n-T)/(S_n-S)\in[a',b']$ *with* $a'=a(1-\beta')/(1-\alpha)$ *and* $b'=b(1-\alpha')/(1-\beta)$.

ii) If there exist a *and* b *with* $0\leq a\leq b$ *such that* $\forall n\geq N, (T_n-T)/(S_n-S)\in[a,b]$ *then* $\forall n\geq N, \Delta T_n/\Delta S_n\in[a',b']$ *with* $a'=a(1-\beta)/(1-\alpha')$ *and* $b'=b(1-\alpha)/(1-\beta')$.

When $T_n=S_{n+1}$ then $\alpha=\alpha'=a, \beta=\beta'=b$ and the first part of this theorem reduces to the first part of theorem 1.18.

Additional results and tools for the asymptotic comparison of sequences could be found in Brezinski [37, 58].

When accelerating the convergence of sequences we are naturally led to use words like *acceleration* or *speed of convergence*. Thus we shall now give precise definitions of such concepts based on kinematical notions and the main corresponding results. The interested reader is referred for more details to Brezinski [72]. Let (S_n) be a sequence converging to S. We assume that $\forall n, S_n$ is different from S and we set

$$d_n=-\log_{10}|S_n-S|, \qquad n=0,1,\ldots$$

d_n represents (up to a constant independent of n) the number of exact decimal digits of S_n.

The *speed of convergence* of (S_n) is the sequence (v_n) defined by

$$v_n = \Delta d_n = -\log_{10}\left|\frac{S_{n+1} - S}{S_n - S}\right|.$$

This notion is quite useful to visualize and understand some properties of convergent sequences. Let us recall that (S_n) is said to have order $r \geq 1$ if $\exists 0 < A \leq B < +\infty$, $\exists N$ such that $\forall n \geq N$

$$A \leq \frac{|S_{n+1} - S|}{|S_n - S|^r} \leq B.$$

If $\lim_{n\to\infty}(S_{n+1} - S)/(S_n - S) = 0$ then (S_n) is said to be super-linear (which does not imply that (S_n) has an order greater than 1). The following results can be proved

Theorem 1.28

i) (S_n) *has order 1 if and only if* $\exists M > 0$, $\exists N$ *such that* $\forall n \geq N$, $v_n < M$.

ii) $\lim_{n\to\infty} |S_{n+1} - S|/|S_n - S| = 1$ *if and only if* $\lim_{n\to\infty} v_n = 0$.

iii) (S_n) *is super-linear if and only if* $\lim_{n\to\infty} v_n = +\infty$.

iv) If (S_n) *has order* $r > 1$ *then* $\lim_{n\to\infty} v_{n+1}/v_n = r$.

v) If (S_n) *has order* 1 *and if* $\lim_{n\to\infty} |S_{n+1} - S|/|S_n - S| = C \neq 1$ *then* $\lim_{n\to\infty} v_{n+1}/v_n = 1$.

It must be noticed that, for some values of n, v_n can be negative.

The *acceleration* of (S_n) is defined as the sequence (γ_n) given by

$$\gamma_n = \Delta v_n = \Delta^2 d_n.$$

The *acceleration* is also a useful notion for the understanding of the behaviour of (S_n). We have the

Theorem 1.29

i) If $\exists N$ *such that* $\forall n \geq N$, $\gamma_n \leq 0$ *then* (S_n) *has order 1.*

ii) A necessary and sufficient condition that (S_n) *has order* $r > 1$ *is that* $\lim_{n\to\infty} \gamma_n = +\infty$.

iii) If $\exists N$ *such that* $\forall n \geq N$, $\gamma_n = \gamma > 0$ *then* (S_n) *is super-linear.*

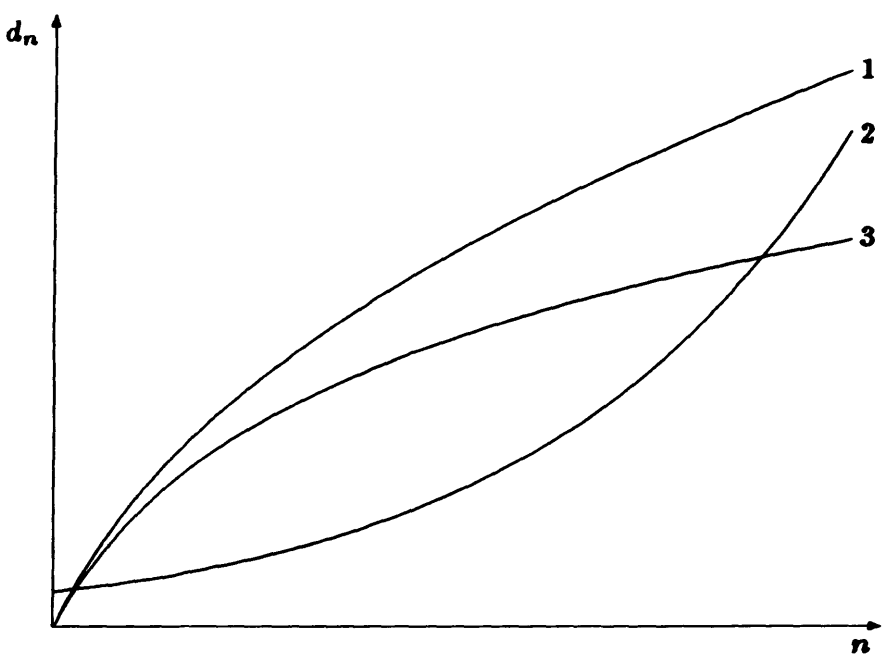
It seems that there is a contradiction between *ii* and *iii*. This is not true since it is not possible to define the order of a sequence for which $\forall n \geq N$, $\gamma_n = \gamma > 0$ since in that case

$$\lim_{n\to\infty} |S_{n+1} - S| / |S_n - S| = 0$$

and $\forall \varepsilon > 0$,

$$\lim_{n\to\infty} |S_{n+1} - S| \Big/ |S_n - S|^{1+\varepsilon} = +\infty.$$

Thus we have the following figure



The curves in the figure were obtained with the three following sequences

1. $S_n = (0.9)^n/(n+1)$ which is linear
2. $S_n = (0.8)^{1.3^n}$ which has order 1.3
3. $S_n = 1/(n+1)$ which is logarithmic

The notions of *speed of convergence* and *acceleration* are useful for comparing two sequences. Let (S'_n) be a second sequence with speed (v'_n) and acceleration (γ'_n). Then we have the

Theorem 1.30

i) If $\exists k > 0, \exists N$ *such that* $\forall n \geq N$, $v'_n \geq v_n + k$ *then* (S'_n) *converges faster than* (S_n).

ii) If $\lim_{n\to\infty} v_n = v$ *and* $\lim_{n\to\infty} v'_n = v' > v$ *then* (S'_n) *converges faster than* (S_n).

Since the reciprocal of this result is not true, we need a sharper one given by the

Theorem 1.31

If $\exists N$ *such that* $\forall n \geq N$, $v'_n > v_n$ *and* $\gamma'_n \geq \gamma_n$ *then* (S'_n) *converges faster than* (S_n).

It is easy to see that the condition $v'_n > v_n > 0$ for all n, implies that the ratio $|S'_n - S'| / |S_n - S|$ is monotonically decreasing and smaller than 1. Thus it converges but not always to zero. Thus *to converge faster* is not equivalent to *to have a greater speed* and the only necessary and sufficient condition is the following

Theorem 1.32

A necessary and sufficient condition that (S'_n) *converges faster than* (S_n) *is that*

$$\lim_{n\to\infty} (d'_n - d_n) = +\infty.$$

(with $d'_n = -\log_{10} |S'_n - S'|$*).*

This page intentionally left blank

Chapter 2

SCALAR EXTRAPOLATION ALGORITHMS

All the sequences considered in this chapter are sequences of real numbers. It will not be difficult to adapt the algorithms and the subroutines to sequences of complex numbers if complex arithmetic is available. If not, the sequences formed by the real and imaginary parts of the numbers can be treated separately as two real sequences. One can also form vectors with two components, the real and imaginary parts of the numbers, and use the vector extrapolation algorithms that will be discussed in the next chapter.

We shall begin with the E-algorithm which covers most of the other algorithms and study it in details. Of course, the drawback of such a generality is that, in a particular case, it will be less powerful (in terms of number of arithmetical operations and storage requirements) than an algorithm particularly adapted to that case. But, on the other hand, its interest lies in its flexibility and generality and in the fact that it can be used for implementing many sequence transformations even those for which a particular recursive algorithm does not exist. After discussing some sequence transformations which are particular cases of the E-transformation (sections 2.2 to 2.7) we shall study others which cannot be included in its framework merely because they were obtained by modifying the rules of an existing algorithm (sections 2.8 to 2.11). The proofs of the results given in this chapter can be found in Brezinski [55] and in the subsequent papers quoted in the text. See also Weniger [457, 458] which contain many interesting results.

2.1 The E-algorithm

The E-transformation is the most general sequence transformation actually known. It contains, as particular cases, almost all the sequence

transformations discovered so far: Richardson polynomial extrapolation, Shanks' transformation and the first generalization of the ε-algorithm, the G-transformation, summation processes, Germain-Bonne transformation, Levin's generalized transforms, the process p and rational extrapolation.

It was obtained independently by several authors but the two more general approaches were given by Håvie [219] and Brezinski [61]. It can also be viewed as a particular case of a generalization of the Neville-Aitken scheme to compute recursively the interpolation polynomial. This generalization, due to Mühlbach [329], allows to compute recursively an interpolating combination of functions forming a complete Chebyshev system. On the various approaches to the E-algorithm and on its numerous applications, see Brezinski [84].

The transformation E is based on the following relation R which is assumed to hold between the members of the sequence (S_n) to be transformed (see section 1.2 on the role of this relation R)

$$S_n - S - a_1 g_1(n) - \cdots - a_k g_k(n) = 0$$

where the $(g_i(n))$'s are given auxiliary sequences which can depend on some terms of the sequence (S_n) itself. In other words, it is assumed that

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n).$$

Writing this relation for the indexes $n, n+1, \ldots, n+k$ and solving the system obtained for the unknown S, which will be denoted by $E_k^{(n)}$ since it usually depends on k and n, as was done in section 1.2, gives

$$E_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}.$$

By construction, the kernel of this transformation is given by

Theorem 2.1
$\forall n, E_k^{(n)} = S$ *if and only if* $\forall n$

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n).$$

Of course, it must be assumed that the determinant in the denominator of $E_k^{(n)}$ does not vanish. It must also be remarked that the kernel of the transformation $E_k : (S_n) \longmapsto \left(E_k^{(n)}\right)_n$ depends on the auxiliary sequences $g_1, \ldots, g_k$. So if the order of $g_1, \ldots, g_k$ is changed the kernels of the transformations $E_1, E_2, \ldots, E_{k-1}$ will be modified accordingly but not that of E_k.

Sequences of the form

$$S_n = \frac{S + a_1 f_1(n) + \cdots + a_p f_p(n)}{1 + a_{p+1} h_1(n) + \cdots + a_{p+q} h_q(n)}$$

with $k = p + q$ are also included in the kernel of E_k since this relation can be written

$$S_n = S + a_1 f_1(n) + \cdots + a_p f_p(n) - a_{p+1} S_n h_1(n) - \cdots - a_{p+q} S_n h_q(n)$$

or

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n)$$

with $g_i(n) = f_i(n)$ for $i = 1, \ldots, p$ and $g_{i+p}(n) = S_n h_i(n)$ for $i = 1, \ldots, q$.

The choice $g_{2i-1}(n) = f_i(n)$, $g_{2i}(n) = S_n h_i(n)$ for $i = 1, \ldots, p$ when $p = q$ can also be made, or the choice $g_i(n) = S_n h_i(n)$ for $i = 1, \ldots, q$ and $g_{i+q}(n) = f_i(n)$ for $i = 1, \ldots, p$, or any other choice. These various choices lead to different $E_i^{(n)}$'s for $i = 1, \ldots, p + q - 1$ but to the same $E_{p+q}^{(n)}$'s thus generalizing techniques due to Larkin [272] and Bulirsch and Stoer [106] for rational interpolation and extrapolation (see also, Stoer and Bulirsch [418]). A different choice in the ordering of the g_i's produces intermediate transformations E_i with different kernels and thus this order can be important and made according to the user's needs. An application of this rational form of the E-algorithm will be given in section 6.2.4.

The E-transformation includes the following particular cases

- Richardson extrapolation process
  $g_i(n) = x_n^i$ where (x_n) is an auxiliary sequence.

- G-transformation
  $g_i(n) = x_{n+i-1}$ where (x_n) is an auxiliary sequence.

- Summation processes
  $g_i(n) = x_i^n$ where (x_i) is an auxiliary sequence.

- Shanks' transformation
  $g_i(n) = \Delta S_{n+i-1}$.

- Germain-Bonne transformation
  $g_i(n) = (\Delta S_n)^i$.

- Levin's generalized transformation
  $g_i(n) = x_n^{i-1} \Delta S_n / y_n$ where (x_n) and (y_n) are auxiliary sequences.

- First generalization of the ε-algorithm
  $g_i(n) = R^i(S_n \Delta x_n)/\Delta x_n$ where (x_n) is an auxiliary sequence and R a difference operator generalizing Δ and defined by

$$R^{k+1} v_n = \Delta \left(\frac{R^k v_n}{\Delta x_n} \right) = R^k \Delta \left(\frac{v_n}{\Delta x_n} \right)$$

  where $v_n = S_n \Delta x_n$.

- Process p
  $g_1(n) = x_n$ and $g_i(n) = \Delta S_{n+i-2}$ for $i \geq 2$ where (x_n) is an auxiliary sequence.

- Thiele rational extrapolation
  $g_i(n) = x_n^i$ for $i = 1, \ldots, p$, $g_{i+p}(n) = S_n x_n^i$ for $i = 1, \ldots, p$, $k = 2p$ and where (x_n) is an auxiliary sequence.

As remarked above if we take in the last case $g_{2i-1}(n) = x_n^i$ and $g_{2i}(n) = S_n x_n^i$ for $i = 1, \ldots, p$, then the kernels of the transformations $E_1, \ldots, E_{k-1}$ are changed according to theorem 2.1 but not that of E_k.

The recursive algorithm which allows to compute the numbers $E_k^{(n)}$ for all k and n without computing the determinants involved in their determinantal definition, is the E-algorithm whose normal rules are

$$E_0^{(n)} = S_n \,, \qquad n = 0, 1, \ldots$$
$$g_{0,i}^{(n)} = g_i(n) \,, \qquad n = 0, 1, \ldots \text{ and } i = 1, 2, \ldots$$

Then for $k = 1, 2, \ldots$ and $n = 0, 1, \ldots$ we have (main rule)

$$E_k^{(n)} = E_{k-1}^{(n)} - \frac{E_{k-1}^{(n+1)} - E_{k-1}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}$$

where the $g_{k-1,k}^{(n)}$'s are auxiliary quantities recursively computed by (auxiliary rule)

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{g_{k-1,i}^{(n+1)} - g_{k-1,i}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}, \qquad i = k+1, k+2, \ldots$$

It can be proved that these auxiliary quantities are also given as a ratio of determinants, namely

$$g_{k,i}^{(n)} = \frac{\begin{vmatrix} g_i(n) & \cdots & g_i(n+k) \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}.$$

Thus $g_{k,i}^{(n)} = 0$ for $i \leq k$ since two rows of the numerator are identical. From the preceding determinantal formulæ for $E_k^{(n)}$ and $g_{k,i}^{(n)}$ we see that we have

$$E_k^{(n)} = \sum_{j=0}^{k} A_j^{(k,n)} S_{n+j}$$

$$g_{k,i}^{(n)} = \sum_{j=0}^{k} A_j^{(k,n)} g_i(n+j)$$

where the coefficients $A_j^{(k,n)}$ are solution of the system given in sec-

tion 1.5 (which gives R_k)

$$\begin{cases} A_0^{(k,n)} + \cdots + A_k^{(k,n)} = 1 \\ A_0^{(k,n)} g_1(n) + \cdots + A_k^{(k,n)} g_1(n+k) = 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ A_0^{(k,n)} g_k(n) + \cdots + A_k^{(k,n)} g_k(n+k) = 0. \end{cases}$$

Thus this system can be solved recursively by the bordering method and for $n = 0$ it furnishes the sequences $\left(E_k^{(0)}\right)$ and $\left(g_{k-1,k}^{(0)}\right)$ for $k = 1, 2, \ldots$.

If $E_{k-1}^{(n)}$ and $g_{k-1,k}^{(n)}$ are replaced by the above expressions in the main rule of the E-algorithm we obtain a recursive scheme for computing the $A_j^{(k,n)}$'s

$$\begin{aligned} A_0^{(0,n)} &= 1 \\ A_0^{(k,n)} \Delta g_{k-1,k}^{(n)} &= A_0^{(k-1,n)} g_{k-1,k}^{(n+1)} \\ A_j^{(k,n)} \Delta g_{k-1,k}^{(n)} &= A_j^{(k-1,n)} g_{k-1,k}^{(n+1)} - A_{j-1}^{(k-1,n+1)} g_{k-1,k}^{(n)}, \; j = 1, \ldots, k-1 \\ A_k^{(k,n)} \Delta g_{k-1,k}^{(n)} &= -A_{k-1}^{(k-1,n+1)} g_{k-1,k}^{(n)} \end{aligned}$$

with $\Delta g_{k-1,k}^{(n)} = g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}$.

The progressive form of the E-algorithm is given by

$$E_{k-1}^{(n+1)} = E_{k-1}^{(n)} + \left(E_k^{(n)} - E_{k-1}^{(n)}\right) \cdot \left(1 - \frac{g_{k-1,k}^{(n+1)}}{g_{k-1,k}^{(n)}}\right)$$

with

$$g_{k-1,i}^{(n+1)} = g_{k-1,i}^{(n)} + \left(g_{k,i}^{(n)} - g_{k-1,i}^{(n)}\right) \cdot \left(1 - \frac{g_{k-1,k}^{(n+1)}}{g_{k-1,k}^{(n)}}\right), \qquad i \geq k.$$

Thus, the second of these formulæ cannot be used in practice since the computation of elements of the descending diagonal $n+1$ uses other elements of the same diagonal for $i \neq k$.

However if the sequences $\left(E_k^{(0)}\right)$ and $\left(g_{k-1,k}^{(0)}\right)$ have been computed by the bordering method as explained above, then the quantities $A_j^{(k-1,0)}$

$(j=0,\ldots,k-1)$ and $A_j^{(k,0)}$ $(j=0,\ldots,k)$ are known. Then we obtain

$$g_{k-1,k}^{(1)} = \frac{A_0^{(k,0)} g_{k-1,k}^{(0)}}{A_0^{(k,0)} - A_0^{(k-1,0)}}$$

and the progressive form of the main rule of the E-algorithm gives the sequence $\left(E_k^{(1)}\right)$. Knowing $E_k^{(1)}$ we can determine the coefficients $A_j^{(k,1)}$ $(j=0,\ldots,k)$ by

$$\begin{aligned} A_j^{(k-1,1)} g_{k-1,k}^{(0)} &= A_{j+1}^{(k-1,0)} g_{k-1,k}^{(1)} - A_{j+1}^{(k,0)} \Delta g_{k-1,k}^{(0)}, \qquad j=0,\ldots,k-2 \\ A_{k-1}^{(k-1,1)} g_{k-1,k}^{(0)} &= -A_k^{(k,0)} \Delta g_{k-1,k}^{(0)}. \end{aligned}$$

Then, from these coefficients, we obtain

$$g_{k-1,k}^{(2)} = \frac{A_0^{(k,1)} g_{k-1,k}^{(1)}}{A_0^{(k,1)} - A_0^{(k-1,1)}}$$

and the progressive form of the main rule of the E-algorithm allows to compute the sequence $\left(E_k^{(2)}\right)$ and so on.

The subroutines EALGO and SEALGO perform the normal form of the E-algorithm.

A more economical algorithm for implementing the E-transformation was obtained by Ford and Sidi [162]. It is based on a quite simple trick consisting in dividing the numerator and the denominator in the determinantal formula of $E_k^{(n)}$ by

$$\begin{vmatrix} g_{k+1}(n) & \cdots & g_{k+1}(n+k) \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}.$$

If we set, for any sequence $u = (u_0, u_1, \ldots)$

$$\Psi_k^{(n)}(u) = \frac{\begin{vmatrix} u_n & \cdots & u_{n+k} \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} g_{k+1}(n) & \cdots & g_{k+1}(n+k) \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}$$

(where the $g_i(n)$'s are kept the same even if they depend on the u_n's and the u_n's are changed) then

$$E_k^{(n)} = \frac{\Psi_k^{(n)}(S)}{\Psi_k^{(n)}(1)}.$$

Using Sylvester's determinantal identity, it can be easily seen that

$$\Psi_k^{(n)}(u) = \frac{\Psi_{k-1}^{(n)}(u) - \Psi_{k-1}^{(n+1)}(u)}{\Psi_{k-1}^{(n)}(g_{k+1}) - \Psi_{k-1}^{(n+1)}(g_{k+1})}.$$

Thus the $E_k^{(n)}$'s can be recursively computed by this algorithm (see Ford and Sidi [162] for a FORTRAN subroutine). However it must be noticed that the computation of $\Psi_k^{(n)}(u)$ requires the knowledge of $(g_{k+1}(n))$ while $E_k^{(n)}$ does not.

A generalization of theorem 2.1 is given by the

Theorem 2.2

If $\forall n, S_n = S + a_1 g_1(n) + a_2 g_2(n) + \cdots$ *then* $\forall k$ *and* $\forall n$

$$E_k^{(n)} = S + a_{k+1}\, g_{k,k+1}^{(n)} + a_{k+2}\, g_{k,k+2}^{(n)} + \cdots$$

The result of theorem 2.1 is recovered if $\forall i > k, a_i = 0$.

Let us now come to the particular rules of the E-algorithm. It was proved by Brezinski [62] that

$$E_{k+m}^{(n)} = \frac{\begin{vmatrix} E_m^{(n)} & \cdots & E_m^{(n+k)} \\ g_{m,m+1}^{(n)} & \cdots & g_{m,m+1}^{(n+k)} \\ \vdots & & \vdots \\ g_{m,m+k}^{(n)} & \cdots & g_{m,m+k}^{(n+k)} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_{m,m+1}^{(n)} & \cdots & g_{m,m+1}^{(n+k)} \\ \vdots & & \vdots \\ g_{m,m+k}^{(n)} & \cdots & g_{m,m+k}^{(n+k)} \end{vmatrix}} = \frac{\begin{vmatrix} E_k^{(n)} & \cdots & E_k^{(n+m)} \\ g_{k,k+1}^{(n)} & \cdots & g_{k,k+1}^{(n+m)} \\ \vdots & & \vdots \\ g_{k,k+m}^{(n)} & \cdots & g_{k,k+m}^{(n+m)} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_{k,k+1}^{(n)} & \cdots & g_{k,k+1}^{(n+m)} \\ \vdots & & \vdots \\ g_{k,k+m}^{(n)} & \cdots & g_{k,k+m}^{(n+m)} \end{vmatrix}}.$$

Similar expressions hold for $g_{k+m,i}^{(n)}$ by replacing the first rows in the numerators by $\left(g_{m,i}^{(n)}, \ldots, g_{m,i}^{(n+k)}\right)$ or $\left(g_{k,i}^{(n)}, \ldots, g_{k,i}^{(n+m)}\right)$ respectively.

In the first expression, if $m = 0$ then the determinantal definition of $E_k^{(n)}$ is recovered. The choice $m = 1$ in the second expression leads to the rules of the E-algorithm. When m is arbitrary we obtain a rule for computing directly the elements of column $k + m$ from those of column k without computing the elements of the intermediate columns thus providing a particular rule for jumping over a singularity or avoiding instability due to cancellation errors.

For example let us use the E-algorithm to implement Shanks' transformation, that is $g_i(n) = \Delta S_{n+i-1}$. Let (S_n) be the partial sums of a power series

$$\begin{aligned} f(x) &= c_0 + c_1 x + c_2 x^2 + \cdots \\ S_n &= c_0 + c_1 x + \cdots + c_n x^n, \qquad n = 0, 1, \ldots \end{aligned}$$

If f is the power series expansion of a rational fraction with a numerator of degree k and a denominator of degree k then, by theorem 2.1, we shall have $\forall n, E_k^{(n)} = f(x)$.

Let us take

$$f(x) = \frac{1 + \varepsilon x}{1 - x^2} = 1 + \varepsilon x + x^2 + \varepsilon x^3 + x^4 + \cdots.$$

If $\varepsilon = 0$ the normal rule of the E-algorithm cannot be used since a division by zero occurs. Applying the particular rule allows to compute $E_2^{(n)}$ directly and we obtain $E_2^{(n)} = f(x)$ for all n.

For small values of ε, numerical instability is present due to cancellation errors. These cancellation errors do not affect the particular rule. For $\varepsilon = 10^{-7}, x = 0.8, f(x) = 2.77778$ we have for $E_2^{(n)}$

n	normal rule	particular rule
0	2.77778	2.77778
1	3.27840	2.77778
2	2.77778	2.77778
3	2.04960	2.77778

The use of the determinantal form of the particular rules is easy in this example since m was small ($m = 2$). For larger values of m it is worthwhile to use the following formula which was proved to be equivalent (Brezinski [79])

$$E_{k+m}^{(n)} = E_k^{(n)} - \left(\Delta E_k^{(n)}, \ldots, \Delta E_k^{(n+m-1)}\right) \cdot \begin{pmatrix} \Delta g_{k,k+1}^{(n)} & \cdots & \Delta g_{k,k+1}^{(n+m-1)} \\ \vdots & & \vdots \\ \Delta g_{k,k+m}^{(n)} & \cdots & \Delta g_{k,k+m}^{(n+m-1)} \end{pmatrix}^{-1} \cdot \begin{pmatrix} g_{k,k+1}^{(n)} \\ \vdots \\ g_{k,k+m}^{(n)} \end{pmatrix}$$

and a similar formula for $g_{k+m,i}^{(n)}$ (Δ operates on the upper indexes). Of course the inverse of the matrix involved in this expression, multiplied by a vector is replaced by the solution of a system of linear equations.

From the determinantal formulæ of the particular rules we have a further generalization of theorems 2.1 and 2.2

Theorem 2.3

$\forall n, E_{k+m}^{(n)} = S$ *if and only if* $\forall n$

$$E_m^{(n)} = S + a_{m+1}\, g_{m,m+1}^{(n)} + \cdots + a_{m+k}\, g_{m,m+k}^{(n)}.$$

If $\forall n, E_m^{(n)} = S + a_{m+1}\, g_{m,m+1}^{(n)} + a_{m+2}\, g_{m,m+2}^{(n)} + \cdots$ *then* $\forall k$ *and* $\forall n$

$$E_{k+m}^{(n)} = S + a_{k+m+1}\, g_{k+m,k+m+1}^{(n)} + a_{k+m+2}\, g_{k+m,k+m+2}^{(n)} + \cdots.$$

The properties of translativity (see section 1.4) of the E-algorithm where studied by Brezinski [81]. We saw above that

$$E_k^{(n)} = F(S_n, \ldots, S_{n+k})$$

with

$$F(u_0, \ldots, u_k) = \sum_{j=0}^{k} A_j u_j$$

where the A_j, which depend on k and n, are solution of a system of linear equations whose coefficients are the $g_i(n)$'s. Thus the A_j's depend on the $g_i(n)$'s and it is easy to see that if they are invariant by translation (that is if they are invariant if (S_n) is replaced by (S_n+b)) then F is translative. But this is not the most general condition for the translativity of the E-algorithm.

Let A be the matrix of the system giving the $A_j^{(k,n)}$'s, let $b = \left(A_0^{(k,n)}, \ldots, A_k^{(k,n)}\right)^T$, let $e = (1, 0, \ldots, 0)$, let $v = (S_n, \ldots, S_{n+k})^T$ and let DA be the matrix obtained by applying the operator D (sum of the partial derivatives, see section 1.4) to the elements of the matrix A. Then we have the

Theorem 2.4

A necessary and sufficient condition that the E-algorithm be translative is that $(v, A^{-1}\, DA\, A^{-1}e) = 0$.

Thus if the g_i's are invariant by translation, then $DA = 0$ and the E-algorithm is translative. If the g_i's are independent of (S_n) then, obviously, the above condition is satisfied. We saw above that the case

$$S_n = \frac{S + a_1 f_1(n) + \cdots + a_p f_p(n)}{1 + a_{p+1} h_1(n) + \cdots + a_{p+q} h_q(n)}$$

fits into the framework of the E-algorithm since this can be transformed into

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n)$$

with $k = p+q$ and where, for example, $g_i(n) = f_i(n)$ for $i = 1, \ldots, p$ and $g_{i+p}(n) = -S_n h_i(n)$ for $i = 1, \ldots, q$. Thus, even if the f_i's and the h_i's are invariant by translation, the g_{i+p}'s are not and thus the function G (see section 1.4) corresponding to the E-algorithm is not translative.

However in some cases the E-algorithm is translative thus providing a counter-example to reciprocal of theorem 1.3. We have the

Theorem 2.5

Let the f_i's and the h_i's be invariant by translation. If $q \leq p$ and if the h_i's are linear combinations of $f_1, \ldots, f_p$ for $i = 1, \ldots, q$, then the E-algorithm is translative.

For the sake of completeness, let us mention other algebraic properties of the E-algorithm. Their proofs can be found in Brezinski [85]. To indicate that $E_k^{(n)}$ depends on (S_n), $(g_1(n)), \ldots, (g_k(n))$, we shall make use of the notation

$$E_k^{(n)} = E_k(S_n; g_1, \ldots, g_k).$$

These properties have been gathered in the

Theorem 2.6

1. $\forall a_i \neq 0$ *for* $i = 1, \ldots, k$

$$E_k(S_n; a_1 g_1, \ldots, a_k g_k) = E_k(S_n; g_1, \ldots, g_k).$$

2. $\forall a_1 \neq 0, \forall a_2, \ldots, a_k$

$$E_k(S_n; a_1 g_1 + \cdots + a_k g_k, g_2, \ldots, g_k) = E_k(S_n; g_1, \ldots, g_k).$$

The same result holds if several g_i are replaced by linear combinations of the others.
$E_k(S_n; g_1, \ldots, g_k)$ is a symmetric function of $g_1, \ldots, g_k$.

3. *If either the g_i's are independent of (S_n) or the E-algorithm is applied with the same g_i's then*

i)

$$E_k(S_n; h_1, \ldots, h_k) = \frac{E_k(S_n d_n^{-1}; g_1, \ldots, g_k)}{E_k\left(d_n^{-1}; g_1, \ldots, g_k\right)}$$

where $h_i(n) = d_n g_i(n)$ and $\forall n, d_n \neq 0$.

ii) If $\forall n, S_n \neq 0$ then

$$E_k(S_n; f_1, \ldots, f_p, g_1, \ldots, g_q) = 1 \Big/ E_k\left(S_n^{-1}; e_1, \ldots, e_p, h_1, \ldots, h_q\right)$$

with $k = p + q$, $g_i(n) = S_n h_i(n)$ and $e_i(n) = S_n^{-1} f_i(n)$.

iii) If $\forall n, S_n \neq 0$ *then*

$$E_k(S_n; h_1, \ldots, h_k) = 1 \Big/ E_k\left(S_n^{-1}; g_1, \ldots, g_k\right)$$

where $h_i(n) = S_n g_i(n)$.

iv) If $\forall n, d_n \neq 0$ *then*

$$E_k(S_n; h_1, g_2, \ldots, g_k) = \frac{E_k\left(S_n d_n^{-1}; g_1, h_2, \ldots, h_k\right)}{E_k\left(d_n^{-1}; g_1, h_2, \ldots, h_k\right)}$$

where $h_1(n) = d_n g_1(n)$ *and* $h_i(n) = d_n^{-1} g_i(n)$ *for* $i = 2, \ldots, k$.

v) If $\forall n, d_n \neq 0$ *then*

$$E_k(d_n S_n; h_1, \ldots, h_k) = \frac{E_k\left(S_n; g_1, \ldots, g_k\right)}{E_k\left(d_n^{-1}; g_1, \ldots, g_k\right)}$$

where $h_i(n) = d_n g_i(n)$.

Property 3–*i*) was obtained by Håvie [219]. It shows how to compute $E_k\left(S_n; h_1, \ldots, h_k\right)$ from $E_k\left(S_n d_n^{-1}; g_1, \ldots, g_k\right)$, $E_k\left(d_n^{-1}; g_1, \ldots, g_k\right)$ and the $g_{k-1,k}^{(n)}$'s obtained from the g_i's without computing the new auxiliary $g_{k,i}^{(n)}$'s from the h_i's.

Let us now come to the convergence and acceleration results which are known for the E-algorithm. These results are few and they only concern the columns (that is k fixed and n tending to infinity). However they are quite interesting since they throw some light on the process of convergence acceleration. They are the following

Theorem 2.7

If $\lim_{n\to\infty} E_{k-1}^{(n)} = S$, *if there exist* α *and* β *such that* $\alpha < 1 < \beta$ *and* $\forall n > N, g_{k-1,k}^{(n+1)} / g_{k-1,k}^{(n)} \notin [\alpha, \beta]$ *then* $\lim_{n\to\infty} E_k^{(n)} = S$.

Of course, this result is not very interesting since it will be difficult in practice to check if the condition is satisfied or not. We shall now give conditions on the initial g_i's such that the preceding condition holds. Thus we have the

Theorem 2.8

If $\lim_{n\to\infty} S_n = S$, *if* $\forall i, \exists b_i \neq 1$ *such that* $\lim_{n\to\infty} g_i(n+1)/g_i(n) = b_i$, *if* $\forall j \neq i, b_j \neq b_i$ *then* $\forall k$, $\lim_{n\to\infty} E_k^{(n)} = S$.

Let us now come to convergence acceleration.

Theorem 2.9

If the conditions of theorem 2.8 are satisfied and if $\lim_{n\to\infty} \left(E_{k-1}^{(n+1)} - S\right) \Big/ \left(E_{k-1}^{(n)} - S\right) = b_k$ *then* $\left(E_k^{(n)}\right)$ *converges to* S *faster than* $\left(E_{k-1}^{(n)}\right)$ *when* n *tends to infinity, that is*

$$\lim_{n\to\infty} \left(E_k^{(n)} - S\right) \Big/ \left(E_{k-1}^{(n)} - S\right) = 0.$$

Moreover if $b_k \neq 0$, $\left(E_k^{(n)}\right)$ *converges to* S *faster than* $\left(E_{k-1}^{(n+1)}\right)$.

As was the case for theorem 2.7, this result is not very useful since the condition is difficult to check in practical situations. Thus, let us now give conditions on (S_n) and the g_i's which insure that the condition of the preceding theorem is satisfied. We have the

Theorem 2.10

If the conditions of theorem 2.8 are satisfied, if $\forall i, g_{i+1}(n) = o(g_i(n))$ $(n \to \infty)$ *and if* $\forall n, S_n = S + a_1 g_1(n) + a_2 g_2(n) + \cdots$ *then* $\forall k$, $\left(E_k^{(n)}\right)$ *converges to* S *faster than* $\left(E_{k-1}^{(n)}\right)$.

Moreover

$$E_k^{(n)} - S \sim a_{k+1} \prod_{i=1}^{k} \frac{b_{k+1} - b_i}{1 - b_i} \cdot g_{k+1}(n) \qquad (n \to \infty)$$

and

$$\frac{E_k^{(n)} - S}{E_{k-1}^{(n)} - S} = O\left(\frac{g_{k+1}(n)}{g_k(n)}\right) \qquad (n \to \infty).$$

These last two assertions were proved by Sidi [404]. They show the gain brought by each column and are the quantitative complement to the previous results.

Another acceleration result was obtained by Matos and Prévost [315]. Its interest is that the conditions of theorem 2.8 are no more required. It is as follows

Theorem 2.11

If $\forall n, S_n = S + a_1 g_1(n) + a_2 g_2(n) + \cdots$ *with* $\forall i, g_i(n) = o(1)$ *and* $g_{i+1}(n) = o(g_i(n))$ $(n \to \infty)$ *and if* $\forall i, \forall p$ *and* $\forall n \geq N$ *we have (with* $g_0(n) \equiv 1$*)*

$$\begin{vmatrix} g_{i+p}(n) & \cdots & g_i(n) \\ \vdots & & \vdots \\ g_{i+p}(n+p) & \cdots & g_i(n+p) \end{vmatrix} \geq 0$$

then $\forall k \geq 0$, $\left(E_{k+1}^{(n)}\right)$ *converges to* S *faster than* $\left(E_k^{(n)}\right)$.

Let us give some examples of sequences satisfying the above assumption on the determinant

- $g_1(n) = g(n)$ where $(g(n))$ is a logarithmic totally monotonic sequence (that is $\forall k$ and n, $(-1)^k \Delta^k g(n) \geq 0$, see section 2.3 for more details) and $g_i(n) = (-1)^i \Delta^i g(n)$ for $i > 1$.
- $g_i(n) = x_n^{\alpha_i}$ with $1 > x_1 > x_2 > \cdots > 0$ and $0 < \alpha_1 < \alpha_2 < \ldots$
- $g_i(n) = \varrho_i^n$ with $1 > \varrho_1 > \varrho_2 > \cdots > 0$.
- $g_i(n) = 1 \Big/ \left((n+1)^{\alpha_i} (\ln(n+2))^{\beta_i}\right)$ with $0 < \alpha_1 \leq \alpha_2 \leq \ldots$ and $\beta_i < \beta_{i+1}$ if $\alpha_i = \alpha_{i+1}$.

Let us comment these results since they show us the way that must be followed for accelerating the convergence of (S_n). The condition $g_{i+1}(n) = o(g_i(n))$ (or, in other words, $\lim_{n \to \infty} g_{i+1}(n)/g_i(n) = 0$) means that $(g_1, g_2, \ldots)$ is an asymptotic sequence and the condition $S_n = S + a_1 g_1(n) + a_2 g_2(n) + \cdots$ means that the error $(S_n - S)$ has an asymptotic expansion with respect to the asymptotic sequence $(g_1, g_2, \ldots)$.

Thus, now, what has to be done for accelerating the convergence of (S_n) becomes clear: one has to find an asymptotic sequence with respect to which the error admits an asymptotic expansion. Sometimes such asymptotic expansions can be obtained from the classical convergence tests for series, Brezinski [77]. Sometimes they can be constructed from a good estimation of the absolute value of the error, Matos [312]. In other cases, they can be obtained from an asymptotic expansion of the inverse of the error, a technique useful for the acceleration of some classes of continued fractions, Matos [313]. In some other cases, they can be deduced from an asymptotic expansion of $\Delta S_{n+1}/\Delta S_n$, Overholt [352],

Matos [310], or of ΔS_n, Matos [309]. These techniques will be described in section 3.3.

It must be noticed that the conditions given in the preceding theorems are only sufficient conditions and that an asymptotic expansion does not always exist.

For example, let us take $S_n = \lambda^n/n$ with $\lambda \in [-1, +1[$ and $g_1(n) = \lambda^n$. Then the conditions of theorems 2.7 and 2.8 are satisfied and $\left(E_1^{(n)}\right)$ converges to zero as (S_n). We have

$$E_1^{(n)} = \frac{\lambda^{n+1}}{(\lambda - 1) \cdot n \cdot (n+1)}.$$

The condition of theorem 2.9 is satisfied and $\left(E_1^{(n)}\right)$ converges faster than (S_{n+1}) since

$$\frac{E_1^{(n)}}{S_{n+1}} = \frac{1}{(\lambda - 1) \cdot n}.$$

However S_n has no asymptotic expansion with respect to an asymptotic sequence beginning with $g_1(n) = \lambda^n$ since

$$\lim_{n \to \infty} S_n / g_1(n) = 0.$$

Let us now take $S_n = \lambda^n$ and $g_1(n) = 1/n$. Then $E_1^{(n)} = \lambda^n[\lambda(n+1) - n]$ which tends to zero although the conditions of theorems 2.7 and 2.8 are not satisfied. In that case also, S_n has no asymptotic expansion since

$$\lim_{n \to \infty} S_n / g_1(n) = 0$$

but the convergence is not accelerated and we have

$$\lim_{n \to \infty} \left| E_1^{(n)} / S_n \right| = \infty.$$

Numerical examples illustrating the preceding theorems will be given in section 6.1.1.

The case where the assumption $b_i \neq b_j, \forall j \neq i$ of theorems 2.8, 2.9 and 2.10 is no more assumed to hold was studied by Fdil [156].

We set $\alpha_1 = 1$ and for $i \geq 2$

$$w_i^{(n)} = \frac{g_1(n+1)}{g_1(n)} - \frac{g_i(n+1)}{g_i(n)}$$

and we have the

Theorem 2.12

If $\lim_{n\to\infty} S_n = S$, *if* $\forall i, \exists b_i \neq 1$ *such that* $\lim_{n\to\infty} g_i(n+1)/g_i(n) = b_i$, *if* $\exists \alpha_i$ *with* $\alpha_j \cdot b_j$ *different from 1 and from* $\alpha_i \cdot b_i, \forall j \neq i$ *and such that* $\lim_{n\to\infty} w_i^{(n+1)}/w_i^{(n)} = \alpha_i$, *then* $\forall k$, $\lim_{n\to\infty} E_k^{(n)} = S$. *Moreover if* $\forall i, g_{i+1}(n) = o(g_i(n))$ *when* $n \to \infty$ *and if* $\forall n, S_n = S + a_1 g_1(n) + a_2 g_2(n) + \cdots$ *with* $a_i \neq 0, \forall i$ *then* $\forall k$, $\left(E_k^{(n)}\right)$ *converges to* S *faster than* $\left(E_{k-1}^{(n)}\right)$.

Prévost is actually extending the result of theorem 2.11 to the diagonals.

To end this section let us present a variant of the E-algorithm which is more economical in a particular case. It is the W-algorithm of Sidi [397] which has many applications in extrapolation methods for infinite integrals as we shall see in section 6.7.4. In this algorithm it is assumed that

$$S_n = S + g(x_n) \cdot \left(a_0 + a_1 x_n^{-1} + a_2 x_n^{-2} + \cdots\right).$$

Thus the E-algorithm can be used with

$$g_i(n) = g(x_n) x_n^{-i+1}, \qquad i = 1, 2, \ldots$$

and we have, for $n \geq 0$

$$S_n = E_k^{(n)} + g(x_n) \cdot \left(a_0 + a_1 x_n^{-1} + \cdots + a_{k-1} x_n^{-(k-1)}\right).$$

From theorem 2.6 $(3-i)$ we have

$$E_k\left(S_n; g(x_n), g(x_n)x_n^{-1}, \ldots, g(x_n)x_n^{-(k-1)}\right) = \frac{E_k\left(S_n/g(x_n); 1, x_n^{-1}, \ldots, x_n^{-(k-1)}\right)}{E_k\left(1/g(x_n); 1, x_n^{-1}, \ldots, x_n^{-(k-1)}\right)}.$$

The W-algorithm is based on this relation and on the fact that for $g_i(n) = x_n^{-i+1}$ the E-algorithm reduces to the Richardson extrapolation scheme. This algorithm is as follows

$$M_{-1}^{(n)} = \frac{S_n}{g(x_n)}, \qquad N_{-1}^{(n)} = \frac{1}{g(x_n)}, \qquad n = 0, 1, \ldots$$

$$M_k^{(n)} = \frac{M_{k-1}^{(n+1)} - M_{k-1}^{(n)}}{x_{n+k+1}^{-1} - x_n^{-1}}$$

$$N_k^{(n)} = \frac{N_{k-1}^{(n+1)} - N_{k-1}^{(n)}}{x_{n+k-1}^{-1} - x_n^{-1}}, \qquad k, n = 0, 1, \ldots$$

$$W_k^{(n)} = \frac{M_k^{(n)}}{N_k^{(n)}}$$

and we have $W_k^{(n)} = E_{k+1}^{(n)}$. The $W_k^{(n)}$'s can also be computed by

$$W_k^{(n)} = \frac{N_{k-1}^{(n+1)} W_{k-1}^{(n+1)} - N_{k-1}^{(n)} W_{k-1}^{(n)}}{N_{k-1}^{(n+1)} - N_{k-1}^{(n)}} = W_{k-1}^{(n+1)} + \frac{W_{k-1}^{(n+1)} - W_{k-1}^{(n)}}{N_{k-1}^{(n+1)} - N_{k-1}^{(n)}} \cdot N_{k-1}^{(n)}.$$

2.2 Richardson extrapolation process

Richardson extrapolation process corresponds to the choice $g_i(n) = x_n^i$ in the E-algorithm, where (x_n) is an auxiliary sequence such that $\forall i$ and $\forall j \neq i, x_i \neq x_j$. Thus the kernel is the set of sequences of the form

$$S_n = S + a_1 x_n + a_2 x_n^2 + \cdots + a_k x_n^k$$

thus showing that Richardson process corresponds to polynomial extrapolation at the point 0.

The numbers $E_k^{(n)}$, denoted usually by $T_k^{(n)}$ in this case, are given by

$$T_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ x_n & \cdots & x_{n+k} \\ \vdots & & \vdots \\ x_n^k & \cdots & x_{n+k}^k \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ x_n & \cdots & x_{n+k} \\ \vdots & & \vdots \\ x_n^k & \cdots & x_{n+k}^k \end{vmatrix}}.$$

It can be proved, by looking directly at their determinantal expression and using the well known value of a Vandermonde determinant, that $g_{k-1,k}^{(n)} = (-1)^{k-1} \cdot x_n \cdot \ldots \cdot x_{n+k-1}$.

Thus the E-algorithm reduces to its main rule which also simplifies and we obtain Richardson extrapolation process

$$T_0 = S_n, \qquad n = 0, 1, \ldots$$

$$T_k^{(n)} = \frac{x_{n+k}T_{k-1}^{(n)} - x_n T_{k-1}^{(n+1)}}{x_{n+k} - x_n}, \qquad k = 1, 2, \ldots ; n = 0, 1, \ldots$$

$T_k^{(n)}$ is the value at the point 0 of the interpolation polynomial $P_k^{(n)}$, of degree at most k, which satisfies

$$P_k^{(n)}(x_{n+i}) = S_{n+i}, \qquad i = 0, \ldots, k.$$

It is well known that these interpolation polynomials $P_k^{(n)}$ can be recursively computed by the Neville-Aitken scheme. Setting $x = 0$ in this scheme leads to Richardson process.

The $T_k^{(n)}$'s can also be obtained directly as the solution of the system of linear equations

$$S_{n+i} = T_k^{(n)} + a_1 x_{n+i} + \cdots + a_k x_{n+i}^k, \qquad i = 0, \ldots, k.$$

Multiplying equation i by b_i and adding them leads to

$$T_k^{(n)} = b_0 S_n + \cdots + b_k S_{n+k}$$

with

$$\begin{cases} b_0 + \cdots + b_k = 1 \\ b_0 x_n + \cdots + b_k x_{n+k} = 0 \\ \vdots \qquad\qquad\qquad\quad \vdots \\ b_0 x_n^k + \cdots + b_k x_{n+k}^k = 0. \end{cases}$$

This system can be solved by the bordering method and thus we obtain a recursive method for computing $T_0^{(n)}, T_1^{(n)}, T_2^{(n)}, \ldots$ for a fixed value of n.

In some particular cases the b_i's can be obtained in closed form as proved by Marchuk and Shaidurov [306]. We have for $n = 0$ and

$$x_i = (i+1)^{-1}, \qquad b_i = \frac{(-1)^{k-i} \cdot (i+1)^{k+1}}{(i+1)! \cdot (k-i)!}, \qquad i = 0, \ldots, k$$

$$x_i = (i+1)^{-2}, \qquad b_i = \frac{2(-1)^{k-i} \cdot (i+1)^{2k+2}}{(k+i+2)! \cdot (k-i)!}, \qquad i = 0, \ldots, k.$$

Moreover

$$\sum_{i=0}^{k} b_i x_i^{k+1} = (-1)^k \cdot x_0 \cdot \ldots \cdot x_k = g_{k,k+1}^{(0)}.$$

If the auxiliary quantities x_n are either independent of (S_n) or invariant by translation then so are the $T_k^{(n)}$'s. This is, in particular, the case when $x_n = \Delta S_n$ thus obtaining an acceleration method due to Germain-Bonne [182].

From the simple structure of the algorithm, we can obtain a quite general convergence result, a weaker form of it being due to Laurent [274].

Theorem 2.13

A necessary and sufficient condition that

$$\forall k \text{ fixed}, \quad \lim_{n\to\infty} T_k^{(n)} = \lim_{n\to\infty} S_n = S$$

$$\forall n \text{ fixed}, \quad \lim_{k\to\infty} T_k^{(n)} = \lim_{k\to\infty} S_k = S$$

for all converging sequences (S_n) is that $\exists \alpha < 1 < \beta$ such that $\forall n \geq N, x_{n+1}/x_n \notin [\alpha, \beta]$.

This result holds even if the auxiliary sequence (x_n) depends on (S_n) as soon as the condition of the theorem is satisfied.

If $S_n = f(x_n)$ and if f is $k+1$ times continuously differentiable in an interval I containing 0 and $x_n, \ldots, x_{n+k}$ then, from the theory of polynomial interpolation, we have

$$S - T_k^{(n)} = (-1)^{k+1} \cdot \frac{x_n \cdot \ldots \cdot x_{n+k}}{(k+1)!} \cdot f^{(k+1)}(\xi)$$

with $\xi \in I$ and depending on the indexes k and n.

Thus, in that case, convergence can occur even if the condition of theorem 2.13 does not hold. We have the

Theorem 2.14

Let (x_n) converge to zero and $S_n = f(x_n)$ where f is $k+1$ times continuously differentiable in an interval I containing 0 and all the x_n's. Then

$$\lim_{n\to\infty} T_k^{(n)} = \lim_{n\to\infty} S_n = S.$$

If f is infinitely many times differentiable in I and if $\exists M$ such that $\forall k$ and $\forall x \in I, \left|f^{(k+1)}(x)\right| \leq M$ then

$$\forall k \text{ fixed}, \quad \lim_{n\to\infty} T_k^{(n)} = \lim_{n\to\infty} S_n = S$$

$$\forall n \text{ fixed}, \quad \lim_{k\to\infty} T_k^{(n)} = \lim_{k\to\infty} S_k = S.$$

More precise results on the convergence of the columns can be obtained, see Crouzeix and Mignot [123] who gave the

Theorem 2.15

Let (x_n) be a sequence of positive terms converging to 0 and $S_n = f(x_n)$ for $n = 0, 1, \ldots$ with f such that, for $k = 0, \ldots, k_0$, we have

$$f(x) = S + a_1 x + \cdots + a_k x^k + R_k(x)$$

with

$$|R_k(x)| \leq C_k x^{k+1}.$$

If $\exists r < 1$ such that $\forall n, x_{n+1}/x_n \leq r$ then $\forall k \leq k_0$

$$T_k^{(n)} - S = O(x_n^{k+1}) \qquad (n \to \infty).$$

If $\forall n, x_n = x_0/(n+1)$ then $\forall k \leq k_0/2$

$$T_k^{(n)} - S = O(x_{n+k}^{k+1}) \qquad (n \to \infty).$$

Regarding convergence acceleration, the following result is a direct consequence of theorem 2.9

Theorem 2.16

Let us assume that $\exists \alpha < 1 < \beta$ such that $\forall n, x_{n+1}/x_n \notin [\alpha, \beta]$. A necessary and sufficient condition that $\left(T_k^{(n)}\right)_n$ converges to S faster than $\left(T_{k-1}^{(n)}\right)_n$ for a fixed value of k, is that

$$\lim_{n\to\infty} \frac{T_{k-1}^{(n+1)} - S}{T_{k-1}^{(n)} - S} = \lim_{n\to\infty} \frac{x_{n+k}}{x_n}.$$

Of course this condition is usually difficult to check in practice. However, from theorem 2.10, we obtain the

Theorem 2.17

Let (x_n) be a sequence converging to 0. We assume that there exists $r \in]-1, +1[$ such that $\lim_{n\to\infty} x_{n+1}/x_n = r$. If $\forall n, S_n = S + a_1 x_n + a_2 x_n^2 + \cdots$ then $\forall k, \left(T_k^{(n)}\right)$ converges to S faster than $\left(T_{k-1}^{(n)}\right)$ (and faster than $\left(T_k^{(n+1)}\right)$ if $r \neq 0$).

Let S_n be the result obtained by applying the trapezoidal rule with a step-size h_n for computing an approximate value of $S = \int_a^b f(x)dx$ where f is sufficiently differentiable in $[a, b]$. If $h_n = h_0/2^n$ and if the Richardson extrapolation process is applied with $x_n = h_n^2$ then the famous Romberg's method described in any textbook of numerical analysis, is recovered. In that case the conditions of theorem 2.16 are satisfied and each column converges to S faster than the preceding one.

Due to its simplicity and effectiveness in many cases, such as in numerical methods for ordinary and partial differential equations, Richardson process has been widely discussed and used. See, for example, Marchuk and Shaidurov [306, 307] which does not contain extrapolation by the E-algorithm since it was published in Russian in 1979.

A generalization of Richardson process was proposed by Christiansen and Petersen [114]. Let us assume that (S_n) has the form

$$S_n = S + a_1 x_n^{k_1} + a_2 x_n^{k_2} + \cdots$$

where now the values of the k_i's are unknown (if they are known the E-algorithm can be applied with $g_i(n) = x_n^{k_i}$). These authors show how to compute approximate $k_i(n)$'s of the k_i's and then extrapolating. First the following functions of the real variable x are defined

$$f_1(n, x) = 1\,, \qquad n = 1, 2, \ldots$$

$$f_{i+1}(n, x) = \frac{\alpha(n, k_i) f_i(n-1, k_i) f_i(n, x) - \alpha(n, x) f_i(n, k_i) f_i(n-1, x)}{\alpha(n, k_i) f_i(n-1, k_i) - f_i(n, k_i)}\,,$$

$$i = 1, 2, \ldots \text{ and } n = i+1, i+2, \ldots$$

with $\alpha(n, x) = (x_{n-1}/x_n)^x$ and $x_0 > x_1 > \cdots > 0$. It is also assumed that $x_n = x_0/(n+1)$ for $n = 0, 1, \ldots$. The algorithm is as follows

1. Set $i = 0$ and $R_0^{(n)} = S_n$, $n = 0, 1, \ldots$

2. Replace i by $i + 1$ and solve the equation

$$\alpha(n, x) \cdot \frac{\alpha(n-1, x) f_i(n-2, x) - f_i(n-1, x)}{\alpha(n, x) f_i(n-1, x) - f_i(n, x)} = \frac{R_{i-1}^{(n-1)} - R_{i-1}^{(n-2)}}{R_{i-1}^{(n)} - R_{i-1}^{(n-1)}}$$

for the unknown x and $n = i+1, i+2, \ldots$ and set $k_i(n) = x$.

3. Compute

$$R_i^{(n)} = \frac{\alpha(n, k_i(n)) f_i(n-1, k_i(n)) R_{i-1}^{(n)} - f_i(n, k_i(n)) R_{i-1}^{(n-1)}}{\alpha(n, k_i(n)) f_i(n-1, k_i(n)) - f_i(n, k_i(n))}$$

for $n = i+2, i+3, \ldots$

4. Go to 2.

Although not yet completely justified from the theoretical point of view, this algorithm seems to work quite well in practice. It can also be used if $x_{n+1} = ax_n$ for $n = 0, 1, \ldots$.

In some applications the k_i's are known. This is, for example, the case for the trapezoidal rule. However due to special properties of the problem some of the coefficients a_i can be zero and in that case the corresponding term $x_n^{k_i}$ disappears. Of course if the E-algorithm is applied it will be less efficient than when the term is present. Thus the preceding procedure could also be applied to check the presence or the absence of the successive terms. After having checked that a term is present, the exact value of the corresponding k_i is used in the extrapolation formula.

To illustrate the method let us compute an approximate value of

$$S = \int_0^1 x^{1/2} dx = \frac{2}{3}$$

by the rectangular method

$$S_n = (n+1)^{-1} \sum_{j=0}^{n} \left(\frac{j+1/2}{n+1} \right)^{1/2}.$$

We have

$$S_n = S + a_1 x_n^{1.5} + a_2 x_n^2 + a_3 x_n^4 + \cdots$$

with $x_n = (n+1)^{-1}$ and we obtain

n	$k_1(n)$	$k_2(n)$	$k_3(n)$	$k_4(n)$	$k_5(n)$
6	1.39	1.98	3.89	5.60	6.80
7	1.40	1.99	3.93	5.74	7.28
8	1.40	1.99	3.95	5.82	7.51
9	1.41	1.99	3.96	5.86	7.65
10	1.42	2.00	3.97	5.90	7.65
11	1.42	2.00	3.97	5.90	

The subroutine RICHAR performs Richardson extrapolation.

2.3 The ε-algorithm

Shanks' transformation (or the ε-algorithm) is certainly, with Richardson process, one of the most familiar convergence acceleration method.

It was found by Shanks [392] as a generalization of Aitken's Δ^2 process. It corresponds to the choice $g_i(n) = \Delta S_{n+i-1}$ in the E-algorithm (or, equivalently, to the choice $g_i(n) = \Delta^i S_n$). Thus its kernel is the set of sequences of the form

$$\begin{aligned} & S_n = S + a_1 \Delta S_n + \cdots + a_k \Delta S_{n+k-1} \\ \text{or} \quad & S_n = S + b_1 \Delta S_n + \cdots + b_k \Delta^k S_n \\ \text{or} \quad & c_0 (S_n - S) + \cdots + c_k (S_{n+k} - S) = 0 \end{aligned}$$

with $c_0 + \cdots + c_k \neq 0$.

We see that this is what was called, in section 1.1, the implicit form of the kernel while, for Richardson process, the explicit form was given. In order to obtain the explicit form of the kernel of Shanks' transformation, one has to solve the above linear difference equation of order k. However, let us first give the determinantal expression of the $E_k^{(n)}$'s, usually denoted by $e_k(S_n)$ in this case. We have

$$e_k(S_n) = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ \Delta S_n & \cdots & \Delta S_{n+k} \\ \vdots & & \vdots \\ \Delta S_{n+k-1} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \Delta S_n & \cdots & \Delta S_{n+k} \\ \vdots & & \vdots \\ \Delta S_{n+k-1} & \cdots & \Delta S_{n+2k-1} \end{vmatrix}}.$$

We also have

$$e_k(S_n) = \frac{H_{k+1}(S_n)}{H_k(\Delta^2 S_n)}$$

where $H_k(u_n)$ denotes the so-called Hankel determinant defined by

$$H_0(u_n) = 1$$

$$H_k(u_n) = \begin{vmatrix} u_n & \cdots & u_{n+k-1} \\ u_{n+1} & \cdots & u_{n+k} \\ \vdots & & \vdots \\ u_{n+k-1} & \cdots & u_{n+2k-2} \end{vmatrix}.$$

Let us give a bit of history. Aitken's Δ^2 process was, in fact, rediscovered by Aitken [6] but it was known to the great Japanese mathematician Seki Kowa (1642?–1708) who used it for accelerating the computation of π by inscribed polygons. It corresponds to $k = 1$ in Shanks' transformation. The case $k = 2$ was used by James Clerk Maxwell [316] (1831–1879) in his treatise on electricity published after his death. Then the general case was treated by Schmidt [387] for solving by iteration a system of linear equations. Finally O'Beirne [341] described a sequence transformation which is Shanks' and used it for accelerating the convergence. An internal report on it was also written by Shanks [391]. Schmidt's discovery remained almost unknown since the problem treated was not convergence acceleration but mostly solution of systems of linear equations. The work of O'Beirne was published as an internal report and remained unknown until it was quoted by Jamieson and O'Beirne [243]. The report of Shanks of 1949 also remained unknown until the paper of Shanks [392] was published although he also wrote several short abstracts of his work between these two dates. These three discoveries of what is now called Shanks' transformation were made completely independently.

The kernel of Shanks' transformation was given by Brezinski and Crouzeix [92] who proved the

Theorem 2.18

A necessary and sufficient condition that $\forall n, e_k(S_n) = S$ *is that* $\exists a_0$, $a_1, \ldots, a_k$ *with* $a_k \neq 0$ *and* $a_0 + \cdots + a_k \neq 0$ *such that* $\forall n$

$$a_0 (S_n - S) + \cdots + a_k (S_{n+k} - S) = 0$$

or, in other words, that $\forall n$

$$S_n = S + \sum_{i=1}^{p} A_i(n) r_i^n + \sum_{i=p+1}^{q} [B_i(n) \cos b_i n + C_i(n) \sin b_i n] e^{w_i n} + \sum_{i=0}^{m} c_i \delta_{in}$$

with $r_i \neq 1$ *for* $i = 1, \ldots, p$ *and where* A_i, B_i *and* C_i *are polynomials in* n *such that if* d_i *is equal to the degree of* A_i *plus one for* $i = 1, \ldots, p$ *and to the maximum of the degrees of* B_i *and* C_i *plus one for* $i = p+1, \ldots, q$ *one has*

$$m + \sum_{i=1}^{p} d_i + 2 \sum_{i=p+1}^{q} d_i = k - 1$$

with the convention that $m = -1$ if there is no term in δ_{in} (Kronecker's symbol).

Shanks' transformation is a generalization of Aitken's Δ^2 process which is recovered for $k = 1$. In that case the kernel is $S_n = S + ar^n$ as seen in section 1.1.

Of course, Shanks' transformation can be implemented via the E-algorithm. Shanks himself computed separately the Hankel determinants $H_{k+1}(S_n)$ and $H_k(\Delta^2 S_n)$ of the numerator and denominator of $e_k(S_n)$ by using their recurrence relationship which is a direct consequence of Sylvester's determinantal identity. It is as follows

$$\begin{aligned} & H_0(u_n) = 1\,, \quad H_1(u_n) = u_n\,, \qquad n = 0, 1, \ldots \\ H_{k+2}(u_n) \cdot H_k(u_{n+2}) \;=\; & H_{k+1}(u_n) \cdot H_{k+1}(u_{n+2}) - [H_{k+1}(u_{n+1})]^2\,, \\ & \qquad\qquad k, n = 0, 1, \ldots \end{aligned}$$

However the most famous algorithm for implementing Shanks' transformation is the ε-algorithm of Wynn [470] which consists in computing the numbers $\varepsilon_k^{(n)}$ by

$$\begin{aligned} & \varepsilon_{-1}^{(n)} = 0\,, \quad \varepsilon_0^{(n)} = S_n\,, \qquad n = 0, 1, \ldots \\ \varepsilon_{k+1}^{(n)} \;=\; & \varepsilon_{k-1}^{(n+1)} + \frac{1}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}\,, \qquad k, n = 0, 1, \ldots \end{aligned}$$

It is related to Shanks' transformation by

$$\varepsilon_{2k}^{(n)} = e_k(S_n)\,, \qquad k, n = 0, 1, \ldots$$

A proof of this relation, based on the E-algorithm, was given by Beckermann [16].

The numbers $\varepsilon_{2k+1}^{(n)}$ are intermediate results and it can be proved that

$$\varepsilon_{2k+1}^{(n)} = \frac{H_k(\Delta^3 S_n)}{H_{k+1}(\Delta S_n)} = \frac{1}{e_k(\Delta S_n)}.$$

We also have

$$\varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n)} - \frac{[H_{k+1}(\Delta S_n)]^2}{H_k(\Delta^2 S_n) \cdot H_{k+1}(\Delta^2 S_n)}.$$

Obviously Shanks' transformation is quasi-linear, that is

$$e_k(aS_n + b) = a\, e_k(S_n) + b.$$

This means that if (S_n) is replaced by $(aS_n + b)$ then $\varepsilon_{2k}^{(n)}$ becomes $a\varepsilon_{2k}^{(n)} + b$ while $\varepsilon_{2k+1}^{(n)}$ becomes $\varepsilon_{2k+1}^{(n)}/a$ since $\varepsilon_{2k+1}^{(n)} = 1/e_k(\Delta S_n)$.

As explained in section 1.7, these numbers $\varepsilon_k^{(n)}$ are displayed in a double entry table, the ε-array, as follows

$$\begin{array}{lllll}
\varepsilon_{-1}^{(0)} = 0 & & & & \\
& \varepsilon_0^{(0)} = S_0 & & & \\
\varepsilon_{-1}^{(1)} = 0 & & \varepsilon_1^{(0)} & & \\
& \varepsilon_0^{(1)} = S_1 & & \varepsilon_2^{(0)} & \\
\varepsilon_{-1}^{(2)} = 0 & & \varepsilon_1^{(1)} & & \varepsilon_3^{(0)} \\
& \varepsilon_0^{(2)} = S_2 & & \varepsilon_2^{(1)} & \ddots \\
\varepsilon_{-1}^{(3)} = 0 & & \varepsilon_1^{(2)} & & \varepsilon_3^{(1)} \\
\vdots & \varepsilon_0^{(3)} = S_3 & & \varepsilon_2^{(2)} & \ddots \\
\vdots & \vdots & \varepsilon_1^{(3)} & & \varepsilon_3^{(2)} \\
\vdots & \vdots & \vdots & \varepsilon_2^{(3)} & \ddots \\
\vdots & \vdots & \vdots & \vdots & \varepsilon_3^{(3)} \\
\vdots & \vdots & \vdots & \vdots & \vdots \quad \ddots
\end{array}$$

Since the $\varepsilon_{2k+1}^{(n)}$'s are intermediate results they can be eliminated thus leading to the so-called cross rule of Wynn [479]

$$\left[\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}\right]^{-1} + \left[\varepsilon_{2k-2}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right]^{-1} = \left[\varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right]^{-1} + \left[\varepsilon_{2k}^{(n)} - \varepsilon_{2k}^{(n+1)}\right]^{-1}$$

which can be used to compute recursively $\varepsilon_{2k+2}^{(n)}$ from the initial conditions

$$\varepsilon_0^{(n)} = S_n\,, \qquad \varepsilon_{-2}^{(n)} = \infty\,, \qquad n = 0, 1, \ldots$$

Let us mention that the cross rule was also proved to hold between the ε's with an odd lower index, that is

$$\left[\varepsilon_{2k+1}^{(n)} - \varepsilon_{2k-1}^{(n+1)}\right]^{-1} + \left[\varepsilon_{2k-3}^{(n+2)} - \varepsilon_{2k-1}^{(n+1)}\right]^{-1} =$$

$$\left[\varepsilon_{2k-1}^{(n+2)} - \varepsilon_{2k-1}^{(n+1)}\right]^{-1} + \left[\varepsilon_{2k-1}^{(n)} - \varepsilon_{2k-1}^{(n+1)}\right]^{-1}$$

with

$$\varepsilon_{-1}^{(n)} = 0\,, \qquad \varepsilon_{1}^{(n)} = 1/\Delta S_n\,, \qquad n = 0, 1, \ldots$$

The particular rules for avoiding numerical instability (see section 1.9) are based on this rule which can also be obtained by Schur complements, see Tempelmeier [431].

Other algebraic properties of Shanks' transformation can be found in Brezinski [55].

The progressive form of the ε-algorithm (see section 1.8) is given by

$$\varepsilon_{k+1}^{(n+1)} = \varepsilon_{k+1}^{(n)} + \frac{1}{\varepsilon_{k+2}^{(n)} - \varepsilon_{k}^{(n+1)}}.$$

Thus knowing $\varepsilon_0^{(1)} = S_1$, $\varepsilon_1^{(0)}$ and $\varepsilon_2^{(0)}$, $\varepsilon_1^{(1)}$ can be computed. Then, from $\varepsilon_1^{(1)}, \varepsilon_2^{(0)}$ and $\varepsilon_3^{(0)}$, $\varepsilon_2^{(1)}$ can be obtained. Then $\varepsilon_1^{(2)}$ can be calculated from $\varepsilon_0^{(2)} = S_2, \varepsilon_1^{(1)}$ and $\varepsilon_2^{(1)}$ and so on.

Similarly the progressive form of the cross rule is

$$\varepsilon_{2k}^{(n+2)} = \varepsilon_{2k}^{(n+1)} + \left[\left(\varepsilon_{2k-2}^{(n+2)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} + \left(\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)}\right)^{-1} - \left(\varepsilon_{2k}^{(n)} - \varepsilon_{2k}^{(n+1)}\right)^{-1}\right]^{-1}.$$

Thus, knowing the first two diagonals $\left(\varepsilon_{2k}^{(0)}\right)$ and $\left(\varepsilon_{2k}^{(1)}\right)$ allows to compute the whole table.

The first diagonal $\left(\varepsilon_{k}^{(0)}\right)$ of the ε-array or the first two diagonals $\left(\varepsilon_{2k}^{(0)}\right)$ and $\left(\varepsilon_{2k}^{(1)}\right)$ of the even part of the ε-array can be obtained by the bordering methods described in section 1.8, which are given in more details in Brezinski [54] and Brezinski [56]. They are based on a modification due to Trench [437] of the bordering method for the special case of Hankel matrices.

Of course the use of such a bordering method requires that all the Hankel determinants are different from zero. When it is not the case, an extension of the process has been proposed by Piñar and Ramirez [332]. It is as follows

Initializations:

$u_j^{(-1)} = 0, \qquad j = 0, 1, \ldots$

$u_0^{(0)} = 1,\ u_j^{(0)} = 0, \qquad j = 1, 2, \ldots$

$d^{(-1)} = 0,\ d^{(0)} = 1$

$s(0) = 1 + \min\{i \in \mathbb{N} \mid S_{n+i} \neq 0, n \text{ fixed}\}$

$p(0) = -1$

$\lambda_{-1}^{(-1)} = 1,\ \lambda_j^{(-1)} = 0 \qquad \text{for } j \neq 0$

$\lambda_j^{(0)} = S_j \qquad \text{for } j = s(0)-1, \ldots, 2s(0)-1$

$\left(\varepsilon_{-2}^{(n)}\right)^{-1} = 0.$

Computation of $\varepsilon_{2s(k)-2}^{(n)}$ from $\varepsilon_{2k-2}^{(n)}$ for n fixed:

1. Compute $\lambda_j^{(k)} = \sum_{i=0}^{k} S_{n+j+i} u_i^{(k)}$ for $j \geq k$ until a non-zero value has been obtained.

2. Set
$$s(k) = 1 + \min\left\{j \geq k \,\middle|\, \lambda_j^{(k)} \neq 0\right\}.$$

3. Set $h = s(k) - k$ and compute $\lambda_i^{(k)}$ and $\lambda_j^{(p(k))}$ for $i = s(k) - 1, \ldots, s(k) + h - 1$ and $j = k - 1, \ldots, k + h - 1$.

4. Solve the system
$$\left(\lambda_{s(k)-1}^{(k)}\right)^{-1} \cdot \begin{pmatrix} & & & \lambda_{s(k)-1}^{(k)} \\ & 0 & \cdots & \vdots \\ & \cdots & & \vdots \\ \lambda_{s(k)-1}^{(k)} & \cdots & \cdots & \lambda_{s(k)+h-1}^{(k)} \end{pmatrix}^{-1} \cdot \begin{pmatrix} B_0^{(k)} \\ \vdots \\ B_h^{(k)} \end{pmatrix} = \begin{pmatrix} \lambda_{k-1}^{(p(k))} \\ \vdots \\ \lambda_{k+h-1}^{(p(k))} \end{pmatrix} \cdot \left(\lambda_{k-1}^{(p(k))}\right)^{-1}.$$

5. Set
$$d^{(s(k))} = d^{(k)} \sum_{i=0}^{k} B_i^{(k)} - \left(\lambda_{k-1}^{(p(k))}\right)^{-1} \lambda_{s(k)-1}^{(k)}\, d^{(p(k))}.$$

6. Compute

$$\left(\varepsilon^{(n)}_{2s(k)-2}\right)^{-1} = \left(\varepsilon^{(n)}_{2k-2}\right)^{-1} + \left(\lambda^{(k)}_{s(k)-1}\right)^{-1} \left(d^{(k)}\right)^2 \sum_{i=1}^{h} i\, B^{(k)}_i.$$

7. Compute the $u_j^{(s(k))}$'s by the recurrence formula

$$u^{(s(k))}_{i+h} = \sum_{j=0}^{h} B^{(k)}_j u^{(k)}_{i+h-j} - \left(\lambda^{(p(k))}_{k-1}\right)^{-1} \lambda^{(k)}_{s(k)-1}\, u^{(p(k))}_{i+h}$$

for $i = -h, \ldots, k$.

8. Replace $p(k)$ by k and k by $s(k)$ and go to 1.

To end with the implementation of Shanks' transformation let us mention that a more general ε-algorithm was studied by Carstensen [112] who gave a cross rule for the E-algorithm.

Let us now come to convergence and acceleration results for the ε-algorithm.

As mentioned in section 1.1, Aitken's Δ^2 process can produce a sequence having several points of accumulation when applied to a convergent sequence. Let us give two examples of this situation. The first one was considered by Lubkin [301] who took

$$S_n = \sum_{i=0}^{n} (-1)^{\lfloor i/2 \rfloor}/(i+1), \qquad n = 0, 1, \ldots$$

where $\lfloor i/2 \rfloor$ is the integer part of $i/2$. This sequence converges to $S = \pi/4 + 0.5 \cdot \ln 2$ and we obtain

$$\varepsilon_2^{(2n)} = S_{2n} + (-1)^n \cdot \frac{2n+3}{(2n+2)\cdot(4n+5)}$$

and

$$\varepsilon_2^{(2n+1)} = S_{2n+1} + (-1)^n \cdot \frac{2n+4}{2n+3}.$$

Thus $\left(\varepsilon_2^{(n)}\right)$ has three points of accumulations $S, S+1$ and $S-1$.

Marx [308] considered

$$S = \sum_{i=0}^{\infty} c_i = 0$$

with $c_{3n-3} = 1/2n$, $c_{3n-2} = 1/(2n+1)$ and $c_{3n-1} = -(4n+1)/2n(2n+1)$. $\left(\varepsilon_2^{(3n-1)}\right)$ converges to 1 while $\left(\varepsilon_2^{(3n-2)}\right)$ and $\left(\varepsilon_2^{(3n)}\right)$ tend to zero.

However, as remarked by Lillich [288], if the ε-algorithm is applied to these two sequences, the other columns and the diagonals of the ε-array converge to the right answer.

Of course, for the ε-algorithm, the general convergence and acceleration theorems 2.7 to 2.12 for the E-algorithm remain valid. Due to the non linearity of Shanks' transformation such results are difficult to obtain and only a few are known. The first ones, due to Wynn [477], deal with sequences of a very special form. They have been gathered in the following theorem

Theorem 2.19

If the ε-algorithm is applied to a sequence (S_n) such that

1. $S_n \sim S + \sum_{i=1}^{\infty} a_i(n+b)^{-i}$ *with* $a_1 \neq 0$ *when* $n \to \infty$, *then, for* k *fixed and* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + \frac{a_1}{(k+1)\cdot(n+b)}.$$

2. $S_n \sim S + (-1)^n \sum_{i=1}^{\infty} a_i(n+b)^{-i}$ *with* $a_1 \neq 0$ *when* $n \to \infty$, *then, for* k *fixed and* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + (-1)^n \cdot \frac{(k!)^2 \cdot a_1}{4^k \cdot (n+b)^{2k+1}}.$$

3. $S_n \sim S + \sum_{i=1}^{\infty} a_i \lambda_i^n$ *with* $1 > \lambda_1 > \lambda_2 > \cdots > 0$ *when* $n \to \infty$, *then, for* k *fixed and* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + \frac{a_{k+1}(\lambda_{k+1}-\lambda_1)^2 \cdot \ldots \cdot (\lambda_{k+1}-\lambda_k)^2}{(1-\lambda_1)^2 \cdot \ldots \cdot (1-\lambda_k)^2} \cdot \lambda_{k+1}^n.$$

4. $S_n \sim S + (-1)^n \sum_{i=1}^{\infty} a_i \lambda_i^n$ *with* $1 > \lambda_1 > \lambda_2 > \cdots > 0$ *when* $n \to \infty$, *then, for* k *fixed and* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + (-1)^n \cdot \frac{a_{k+1}(\lambda_{k+1}-\lambda_1)^2 \cdot \ldots \cdot (\lambda_{k+1}-\lambda_k)^2}{(1+\lambda_1)^2 \cdot \ldots \cdot (1+\lambda_k)^2} \cdot \lambda_{k+1}^n.$$

We recall that the notation $u_n \sim v_n$ means that $\lim_{n\to\infty} u_n/v_n = 1$ where (u_n) and (v_n) are sequences converging to zero.

Other results of this type were given by Garibotti and Grinstein [173]

Theorem 2.20

If the ε-algorithm is applied to a sequence (S_n) such that

1. $S_n \sim S + (n+a)^b \lambda^n \sum_{i=0}^{\infty} a_i (n+a)^{-i}$ *with* $a_0 = 1$, $a \geq 0$, $|\lambda| \leq 1$, $\lambda \neq 1$ *and* $b \in \mathbb{C}$, *then, when* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + \lambda^{n+2k} [b]_k (-1)^k k! (n+a)^{b-2k} (\lambda - 1)^{-2k}$$

for $k \geq 0$ *and* $b \neq 0, 1, 2, \ldots$ *or for* $0 \leq k \leq b$ *if* $b \in \mathbb{N}$.
If $b \in \mathbb{N}$ *and* $k \geq b+1$ *then, for* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + a_{b+1} \lambda^{n+2k} (k-b-1)! (k+b+1)! (\lambda - 1)^{-2k} (n+a)^{-2k-1}$$

where $[b]_j = b(b-1)\cdot\ldots\cdot(b-j+1)$ *if* $j > 0$ *and* $[b]_0 = 1$.

2. $S_n \sim S + (n+a)^b \lambda^{n+1} \sin a_n$ *with* $a \geq 0$, $0 < \lambda \leq 1$, $b \in \mathbb{C} \setminus \mathbb{N}$, $a_n = n\theta + \delta$ *with* $0 < \theta < \pi$ *and* $\delta \in \mathbb{R}$ *then, for* k *fixed and* $n \to \infty$, $n \in M_k$

$$\varepsilon_{2k}^{(n)} \sim S + (-q)^{-k} \lambda^{n+k+1} 2^{2N} (\sin\theta)^{2(k-N)} \cdot [b]_N N! (n+a)^{b-2N} \cdot \frac{(\sin a_{n+k})^{2N-k+1}}{\left(\sin a'_{n+k}\right)^{k-2N}}$$

where $N = k/2$ *if* k *is even,* $N = (k-1)/2$ *if* k *is odd,* $q = 2(A^+ - \cos\theta)$, $A^{\pm} = (q^{-1} \pm q)/2$, $a'_n = a_n - \xi$ *with* $\xi \in]0, \pi]$ *defined by* $\tan\xi = A^- \sin\theta / (A^+ \cos\theta - 1)$ *and the set* M_k *given by*

$$M_k = \{n \mid \sin a_{n+k} \neq 0 \text{ if } k \text{ is even, } \sin a'_{n+k} \neq 0 \text{ if } k \text{ is odd}\}.$$

3. $S_n \sim S + (-1)^n \left[(n+a)^b + c\right]$ *with* $a \geq 0$, $c \in \mathbb{C}$, $b \in \mathbb{C}$, $b \neq 0$ *then, for* k *fixed and* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + (-1)^{n+k} 2^{-2k} (n+a)^{b-2k} c_k$$

where $c_k = -(k-1)! [b]_{k+1}$ *if* $Re\ b < 0$ *and* $c \neq 0$ *and* $c_k = k! [b]_k$ *if* $Re\ b > 0$ *and* c *arbitrary or* $Re\ b \leq 0$ *and* $c = 0$.

4. $S_n \sim S + a_i(n+a)^{-i}$ *with* $a_i \neq 0$, $i \geq 1$, $a \geq 0$ *then, for* k *fixed and* $n \to \infty$

$$\varepsilon_{2k}^{(n)} \sim S + a_i i!k! \Big/ \left[(k+i)!(n+a)^i\right] .$$

In case 2 the asymptotic expansion given for $\varepsilon_{2k}^{(n)}$ is only valid for the indexes n belonging to the infinite set M_k.

We shall now look at convergence and acceleration results for two important classes of sequences, namely the so-called totally monotonic and totally oscillating sequences.

A sequence (S_n) is said to be totally monotonic if $\forall k, \forall n$

$$(-1)^k \Delta^k S_n \geq 0$$

and we shall write $(S_n) \in TM$.

Thus since $S_n \geq 0$ and $S_{n+1} - S_n \leq 0$ we have $0 \leq S_{n+1} \leq S_n$ which shows that a totally monotonic sequence converges and that $\forall n, S_n \geq S \geq 0$.

It was proved that a necessary and sufficient condition that $(S_n) \in TM$ is that there exists α bounded and non decreasing in $[0,1]$ such that

$$S_n = \int_0^1 x^n d\alpha(x) , \quad n = 0, 1, \ldots$$

Thus the sequences $(S_n = \lambda^n)$ for $\lambda \in]0,1[$ and $(S_n = 1/(n+1))$ are totally monotonic.

Totally monotonic sequences have some important and interesting properties. First of all they satisfy the inequality, $\forall n$

$$S_{r(n+k)}^{1/r} \leq S_{np}^{1/p} \cdot S_{kq}^{1/q}$$

where r, p, q are rational numbers such that $r(n+k), np$ and kq are integers and $p^{-1} + q^{-1} = r^{-1}$.

For example, if $p = q = 2$ and $r = 1$ then

$$S_{n+k}^2 \leq S_{2n} \cdot S_{2k} , \quad n, k = 0, 1, \ldots$$

If $k = 0$ and $r = 1$, then

$$S_n^p \leq S_{np} \cdot S_0^{p-1} , \quad n, k = 0, 1, \ldots$$

From these inequalities we can deduce that if $(S_n) \in TM$ and if $\forall n, S_n$ is different from its limit S (if this is not the case then $\forall k \geq n, S_k = S$ or, equivalently, α has a finite number of points of increase) then $\exists \lambda \in]0,1[$ such that $\lambda^n = O(S_n - S)$. Conversely, if (S_n) is a sequence converging to S such that $\forall n, S_n > S$ and if $\forall \lambda \in]0,1[, S_n - S = o(\lambda^n)$ then $(S_n - S)$ cannot be a totally monotonic sequence. This result shows that a totally monotonic sequence cannot decrease faster than a geometric sequence.

Certain sequences deduced from a totally monotonic sequence are also totally monotonic sequences under some conditions. The first results of this kind were proved by Wynn [482]. Later Brezinski [52] showed that if $(S_n) \in TM$ then $(f(S_n)) \in TM$ provided that f has a series expansion with all the coefficients positive and $S_0 < R$, the radius of convergence of f. Of course $(f(-S_n)) \in TM$ if the coefficients of f have alternate signs and $S_0 < R$. These results are gathered in the

Theorem 2.21

Let $(S_n) \in TM$. Then the following sequences are also totally monotonic provided that the given conditions are satisfied:

$(1 - S_n)^{-1}$	*if* $S_0 < 1$.
$\prod_{i=0}^{n} S_i^{-1}$	*if* $S_0 \geq 1$.
$\prod_{i=0}^{n}(1 - S_i)$	*if* $S_0 \leq 1$.
$a^{(-1)^{k+1}\Delta^k S_n}$	*if* $0 < a \leq 1,\ k \geq 0$.
$a^{\sum_{i=0}^{n} S_i}$	*if* $0 \leq a \leq 1$.
$a^r - (a - S_n)^r$	*if* $0 \leq r \leq 1$ *and* $S_0 \leq a$.
$(a - S_n)^{-r}$	*if* $r \geq 0$ *and* $S_0 < a$.
$\tan(S_n)$	*if* $S_0 < \pi/2$.
a^{S_n}	*if* $a > 1$.
$-\ln(1 - S_n)$	*if* $S_0 < 1$.
$\arcsin(S_n)$	*if* $S_0 < 1$.
$\arccos(S_n)$	*if* $S_0 < 1$.
$\sinh(S_n)$	*no condition.*
$\cosh(S_n)$	*no condition.*

Of course this list is not limitative. One can also use the fact (used in the proof of the result mentioned above) that if (S_n) and (V_n) are two totally monotonic sequences then so are $(S_n V_n)$ and $(aS_n + bV_n)$ if a and b are positive.

Moreover if $(S_n) \in TM$, then $\forall k$ fixed $\left((-1)^k \Delta^k S_n\right) \in TM$. Wynn [477] proved that, for a totally monotonic sequence, all the Hankel determinants $H_k(S_n)$ are positive. Using these results, Brezinski [39, 45] proved the

Theorem 2.22

If the ε-algorithm is applied to a sequence (S_n) which converges to S and if there exist a and b such that $(aS_n + b) \in TM$ then, $\forall k$ and $\forall n$

$$
\begin{aligned}
&0 \leq a\varepsilon_{2k+2}^{(n)} + b \leq a\varepsilon_{2k}^{(n)} + b \\
&0 \leq a\varepsilon_{2k}^{(n+1)} + b \leq a\varepsilon_{2k}^{(n)} + b \\
&0 \leq a\varepsilon_{2k+2}^{(n)} + b \leq a\varepsilon_{2k}^{(n+1)} + b \\
&0 \leq a\varepsilon_{2k+2}^{(n)} + b \leq a\varepsilon_{2k}^{(n+2)} + b \\
&\forall k \text{ fixed}, \lim_{n\to\infty} \varepsilon_{2k}^{(n)} = S \\
&\forall n \text{ fixed}, \lim_{k\to\infty} \varepsilon_{2k}^{(n)} = S.
\end{aligned}
$$

Thus, all the columns and all the descending diagonals converge to S, the limit of (S_n). Moreover the inequalities show which is the best approximation of S among all the ε's which have been computed: if only $S_0, \ldots, S_{2k}$ are known then $\varepsilon_{2k}^{(0)}$ is the best approximation of S, while it is $\varepsilon_{2k}^{(1)}$ if $S_0, \ldots, S_{2k+1}$ are used.

If $(S_n) \in TM$, then $\exists r \in]0, 1]$ such that $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = r$ and the following acceleration result was obtained by Brezinski [57]

Theorem 2.23

If the ε-algorithm is applied to a sequence (S_n) which converges to S, if there exist a and b such that $(aS_n + b) \in TM$ and if $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) \neq 1$ then

$$
\begin{aligned}
&\forall k \text{ fixed}, \quad \lim_{n\to\infty} \left(\varepsilon_{2k}^{(n)} - S\right) \Big/ (S_{n+2k} - S) = 0 \\
&\forall n \text{ fixed}, \quad \lim_{k\to\infty} \left(\varepsilon_{2k}^{(n)} - S\right) \Big/ (S_{n+2k} - S) = 0.
\end{aligned}
$$

It is not yet known if each column converges or not faster than the preceding column and if each diagonal converges or not faster than the preceding one.

For logarithmic totally monotonic sequences these results are certainly all not true. For example, if $S_n = 1/(n+1)$ then $\varepsilon_{2k}^{(n)} = 1/(k+1)(n+k+1)$ and $\lim_{n\to\infty} \varepsilon_{2k}^{(n)}/S_{n+2k} = 1/(k+1)$ while $\lim_{k\to\infty} \varepsilon_{2k}^{(n)}/S_{n+2k} = 0$. In this case we also have $\lim_{n\to\infty} \varepsilon_{2k+2}^{(n)}/\varepsilon_{2k}^{(n+2)} = (k+1)/(k+2)$ and $\lim_{k\to\infty} \varepsilon_{2k+2}^{(n)}/\varepsilon_{2k}^{(n+2)} = 1$.

The second class of sequences for which almost complete results have been obtained is that of totally oscillating sequences. A sequence (S_n) is said to be totally oscillating if the sequence $((-1)^n S_n)$ is totally monotonic and we shall write $(S_n) \in TO$.

Contrarily to totally monotonic sequences, not every totally oscillating sequence converges but if it does, then its limit is zero. Thus one can think that it is uninteresting to accelerate the convergence of convergent totally oscillating sequences since their limit is known. However one must remember that, due to the quasi-linearity of the ε-algorithm, all the results hold if there exist two unknown constants a and b such that $(aS_n + b) \in TO$ in which case the limit of (S_n) is also unknown.

First the following inequalities and convergence results hold, see Wynn [477] and Brezinski [39, 45, 57]

Theorem 2.24

If the ε-algorithm is applied to a sequence (S_n) which converges to S and if there exist a and b such that $(aS_n + b) \in TO$ then, $\forall k$ and $\forall n$

$$
\begin{aligned}
&0 \le a\varepsilon_{2k+2}^{(2n)} + b \le a\varepsilon_{2k}^{(2n)} + b \\
&a\varepsilon_{2k}^{(2n+1)} + b \le a\varepsilon_{2k+2}^{(2n+1)} + b \le 0 \\
&a\left(\varepsilon_{2k}^{(2n+1)} - \varepsilon_{2k}^{(2n)}\right) \le a\left(\varepsilon_{2k+2}^{(2n+1)} - \varepsilon_{2k+2}^{(2n)}\right) \le 0 \\
&0 \le a\left(\varepsilon_{2k+2}^{(2n+2)} - \varepsilon_{2k+2}^{(2n+1)}\right) \le a\left(\varepsilon_{2k}^{(2n+2)} - \varepsilon_{2k}^{(2n+1)}\right) \\
&0 \le a\varepsilon_{2k+2}^{(2n)} + b \le a\varepsilon_{2k}^{(2n+2)} + b \\
&a\varepsilon_{2k}^{(2n+3)} + b \le a\varepsilon_{2k+2}^{(2n+1)} + b \le 0 \\
&\forall k \text{ fixed}, \lim_{n\to\infty} \varepsilon_{2k}^{(n)} = S \\
&\forall n \text{ fixed}, \lim_{k\to\infty} \varepsilon_{2k}^{(n)} = S.
\end{aligned}
$$

It can be proved that if $(S_n) \in TO$, then $\exists r \in [-1, 0[$ such that

$\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S) = r$ and thus the following result can be proved, see Brezinski [57]

Theorem 2.25

If the ε-algorithm is applied to a sequence (S_n) which converges to S and if there exist a and b such that $(aS_n + b) \in TO$ then

$$\forall k \text{ fixed}, \quad \lim_{n\to\infty} \left(\varepsilon_{2k}^{(n)} - S\right) \Big/ (S_{n+2k} - S) = 0$$

$$\forall n \text{ fixed}, \quad \lim_{k\to\infty} \left(\varepsilon_{2k}^{(n)} - S\right) \Big/ (S_{n+2k} - S) = 0.$$

Additional results dealing with *TM* and *TO* sequences and their generalizations were obtained by Prévost [367]. Let us first consider the sequence

$$S_n = S + \int_a^b x^n \, d\alpha(x), \qquad n = 0, 1, \ldots$$

where $-1 \leq a < b < 1$, α bounded and non decreasing in $[a, b]$ and with no jump at -1. Such a sequence converges to S and it is clearly a generalization of *TM* and *TO* sequences. We have the

Theorem 2.26

Let (n_i) and (k_i) be two infinite non decreasing sequences of integers such that $\lim_{i\to\infty} k_i = +\infty$, $\lim_{i\to\infty} k_i/n_i = +\infty$ (in particular if $n_i = n, \forall i$ the second condition is satisfied even if $n = 0$). If the ε-algorithm is applied to the above sequence then

$$\lim_{i\to\infty} \frac{\varepsilon_{2k_i}^{(n_i)} - S}{S_{n_i+2k_i} - S} = 0.$$

Prévost also considered the Cauchy product of a *TM* and a *TO* sequence and obtained the

Theorem 2.27

Let $(c_n) \in TM$, $(d_n) \in TO$ and convergent (thus $\lim_{n\to\infty} d_n = 0$) and

$$S_n = \sum_{i=0}^{n} c_{n-i} d_i, \qquad n = 0, 1, \ldots$$

If (c_n) is non-logarithmic then (S_n) tends to zero and its convergence can be accelerated by the ε-algorithm in the sense of the preceding theorem.

We shall now apply the ε-algorithm to the sequence of the partial sums of a power series. Let f be a formal power series

$$f(z) = \sum_{i=0}^{\infty} c_i z^i.$$

We apply to ε-algorithm to the sequence (S_n), denoted by $(S_n(z))$ since it depends on the variable z, given by

$$S_n(z) = \sum_{i=0}^{n} c_i z^i, \qquad n = 0, 1, \ldots$$

Thus $\Delta S_n(z) = c_{n+1} z^{n+1}$ and it follows from the determinantal expression of $\varepsilon_{2k}^{(n)}$ that

$$\varepsilon_{2k}^{(n)} = \frac{\begin{vmatrix} z^k S_n(z) & z^{k-1} S_{n+1}(z) & \cdots & S_{n+k}(z) \\ c_{n+1} & c_{n+2} & \cdots & c_{n+k+1} \\ \vdots & \vdots & & \vdots \\ c_{n+k} & c_{n+k+1} & \cdots & c_{n+2k} \end{vmatrix}}{\begin{vmatrix} z^k & z^{k-1} & \cdots & 1 \\ c_{n+1} & c_{n+2} & \cdots & c_{n+k+1} \\ \vdots & \vdots & & \vdots \\ c_{n+k} & c_{n+k+1} & \cdots & c_{n+2k} \end{vmatrix}}.$$

Since $S_n(z) = f(z) - \sum_{i=n+1}^{\infty} c_i z^i$ then, replacing in the above numerator, we see that, provided $H_k(c_{n+1}) \neq 0$

$$\varepsilon_{2k}^{(n)} = f(z) + O\left(z^{n+2k+1}\right).$$

$\varepsilon_{2k}^{(n)}$ is a rational function with a numerator of degree $n + k$ and a denominator of degree k so that its power series expansion (obtained by dividing the numerator by the denominator according to the increasing powers of z) agrees with the series f as far as possible, that is up to the degree of the numerator plus the degree of the denominator inclusively. Such a rational function is called a *Padé approximant* of f and is denoted by $[n + k/k]_f(z)$.

Padé approximants have very many applications in applied sciences such as physics, chemistry, mechanics, fluid dynamics, circuit theory, were the solution of problems is often obtained as a (divergent or convergent) power series whose coefficients can be hardly computed. They also have applications in numerical analysis in connection with the approximation of special functions, the z-transform, the inversion of the Laplace transform, the construction of A-stable methods for the numerical integration of ordinary differential equations, fixed point methods, the Borel transform, the reconstruction of a weight function from its moments, ... Padé approximants have been the subject of an enormous literature (see the bibliography gathered by Brezinski [88] containing more than 6000 references), their history is quite long since they are connected with continued fractions, one of the oldest subjects in mathematics (Brezinski [87]), and with many other subjects such as formal orthogonal polynomials (Brezinski [63]). It is not our purpose here to develop this topics. We refer the interested reader to Brezinski and van Iseghem [103] for the theory and to Draux and van Ingelandt [146] for the subroutines. Applications to physics are described by Baker and Graves-Morris [11] and Guttmann [207]. Because of the connections between the ε-algorithm, continued fractions, orthogonal polynomials and Padé approximants, software for one subject can be used for an other one. On these questions one can consult Brezinski [60].

In the floppy disk given with this book, the ε-algorithm can be implemented with the subroutines EPSRHO and EPSRHA which contain the particular rules of Wynn [475] (see section 1.9). The main diagonal of the ε-array can be computed by the bordering method, as explained above, via the subroutines BORDER and BLBORD. This diagonal is used as an initialization for implementing the progressive form of the ε-algorithm.

Let us also mention that, in Brezinski [56], two subroutines, EBOR1 and EBOR2, contain two different bordering methods for computing the sequence $\left(\varepsilon_{2k}^{(0)}\right)$. An improvement of the second bordering method used in EBOR2 was given by Brezinski [63], p. 173. Similar techniques were used by Brezinski [59] in the subroutine PADE for computing the diagonal sequence $([k/k])$ of Padé approximants.

In Brezinski [63] a conversational program for computing any sequence of Padé approximants is given. This program assumes that all the Padé approximants exist, that is $H_k(c_n) \neq 0$ for all k and n. The most general program for Padé approximation can be found in Draux and

van Ingelandt [146]. It is a conversational program which computes recursively any sequence of Padé approximants, detects and jumps over the singularities even if they are not isolated and furnishes the exact values (calculated in rational arithmetic using several words) of their coefficients. It is the only subroutine of this type actually published and it has needed 12 years of research (for the theory and the construction of the codes) by different people. Paszkowski is actually writing another one.

Before ending this section, let us describe another algorithm which is related to Shanks' transformation, among others.

Let (a_n) be a given sequence. We consider the following ratios of determinants

$$\omega_{2k}^{(n)} = H_{k+1}(a_n)/H_k(a_{n+2}).$$

If $a_n = \Delta^n S_i$ then $\omega_{2k}^{(0)} = e_k(S_i) = \varepsilon_{2k}^{(i)}$ and $\omega_{2k}^{(1)} = 1/\varepsilon_{2k+1}^{(i)}$. It was proved by Wynn [484] that these ratios can be recursively computed by the so-called ω-algorithm

$$\begin{aligned}
&\omega_{-1}^{(n)} = 0\,, \quad \omega_0^{(n)} = a_n\,, && n = 0,1,\ldots \\
\omega_{2k+1}^{(n)} &= \omega_{2k-1}^{(n+1)} + \frac{\omega_{2k}^{(n)}}{\omega_{2k}^{(n+1)}}\,, && k,n = 0,1,\ldots \\
\omega_{2k+2}^{(n)} &= \omega_{2k}^{(n+1)} \cdot \left(\omega_{2k+1}^{(n)} - \omega_{2k+1}^{(n+1)}\right), && k,n = 0,1,\ldots
\end{aligned}$$

As in the ε-algorithm, the ω's with a lower odd index are intermediate computations. These intermediate results can be eliminated and we obtain

$$\begin{aligned}
&\omega_0^{(n)} = a_n\,, \quad \omega_2^{(n)} = a_n - \frac{a_{n+1}^2}{a_{n+2}}\,, \quad n = 0,1,\ldots \\
\omega_{2k+2}^{(n)} &= \omega_{2k}^{(n)} + \left[\omega_{2k}^{(n+1)}\right]^2 \cdot \frac{\omega_{2k}^{(n+2)} - \omega_{2k-2}^{(n+2)}}{\omega_{2k-2}^{(n+2)} \cdot \omega_{2k}^{(n+2)}}\,, \quad k = 1,2,\ldots\,; n = 0,1,\ldots
\end{aligned}$$

which can also be written as

$$\begin{aligned}
&\omega_{-2}^{(n)} = \infty\,, \quad \omega_0^{(n)} = a_n\,, && n = 0,1,\ldots \\
\omega_{2k+2}^{(n)} &= \omega_{2k}^{(n)} - \left[\omega_{2k}^{(n+1)}\right]^2 \cdot \left(\frac{1}{\omega_{2k}^{(n+2)}} - \frac{1}{\omega_{2k-2}^{(n+2)}}\right), && k,n = 0,1,\ldots
\end{aligned}$$

The ω-algorithm can be used to implement the confluent forms of the ε and ρ-algorithms which will be discussed in chapter 5 (see Brezinski [55, 60]).

It can also be used for the implementation of new sequence transformations defined, for example, by the choices

$$\begin{aligned} & a_n = \Delta^n S_i / n! \\ \text{or} \quad & a_n = [x_i \dots x_{i+n}] \\ \text{or} \quad & a_n = n! \cdot [x_i \dots x_{i+n}] \end{aligned}$$

where $[x_i \dots x_{i+n}]$ is the usual divided difference of order n of the function f such that $\forall i, f(x_i) = S_i$ and where (x_i) is a given auxiliary sequence. These new transformations have not yet been studied.

The subroutine OMEGA performs the ω-algorithm.

2.4 The G-transformation

The most general form of the G-transformation was introduced by Gray, Atchinson and McWilliams [197]. It corresponds to the particular choice $g_i(n) = x_{n+i-1}$ in the E-algorithm where (x_n) is a given auxiliary sequence. The G-transformation is intermediate between the E-algorithm and the ε-algorithm which corresponds to the particular choice $x_n = \Delta S_n$.

Thus its kernel is the set of sequences of the form

$$S_n = S + a_1 x_n + \cdots + a_k x_{n+k-1}.$$

The numbers $E_k^{(n)}$, usually denoted by $G_k^{(n)}$ in this case, are given by

$$G_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ x_n & \cdots & x_{n+k} \\ \vdots & & \vdots \\ x_{n+k-1} & \cdots & x_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ x_n & \cdots & x_{n+k} \\ \vdots & & \vdots \\ x_{n+k-1} & \cdots & x_{n+2k-1} \end{vmatrix}}.$$

Obviously this transformation is quasi-linear if the x_n's are invariant by translation of the sequence (S_n) or if the same x_n's are used.

If we set

$$r_k^{(n)} = \frac{\begin{vmatrix} x_n & \cdots & x_{n+k-1} \\ \vdots & & \vdots \\ x_{n+k-1} & \cdots & x_{n+2k-2} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ x_n & \cdots & x_{n+k-1} \\ \vdots & & \vdots \\ x_{n+k-2} & \cdots & x_{n+2k-3} \end{vmatrix}}$$

then, obviously, $r_k^{(n)} = (-1)^{k-1} \cdot g_{k-1,k}^{(n)}$.

If we set now

$$s_k^{(n)} = \frac{\begin{vmatrix} 1 & \cdots & 1 \\ x_n & \cdots & x_{n+k} \\ \vdots & & \vdots \\ x_{n+k-1} & \cdots & x_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} x_n & \cdots & x_{n+k-1} \\ \vdots & & \vdots \\ x_{n+k-1} & \cdots & x_{n+2k-2} \end{vmatrix}}$$

and if we apply Sylvester's determinantal identity, then the $r_k^{(n)}$'s and the $s_k^{(n)}$'s can be recursively computed by the so-called rs-algorithm due to Pye and Atchison [369]

$$s_0^{(n)} = 1\,, \quad r_1^{(n)} = x_n\,, \qquad n = 0, 1, \ldots$$

$$s_{k+1}^{(n)} = s_k^{(n+1)} \cdot \left(\frac{r_{k+1}^{(n+1)}}{r_{k+1}^{(n)}} - 1 \right)\,, \qquad k, n = 0, 1, \ldots$$

$$r_{k+1}^{(n)} = r_k^{(n+1)} \cdot \left(\frac{s_k^{(n+1)}}{s_k^{(n)}} - 1 \right)\,, \qquad k = 1, 2, \ldots\,; n = 0, 1, \ldots$$

Then the $G_k^{(n)}$'s are obtained by the main rule of the E-algorithm which, in this case, becomes the G-algorithm

$$G_k^{(n)} = G_{k-1}^{(n)} - \frac{G_{k-1}^{(n+1)} - G_{k-1}^{(n)}}{r_k^{(n+1)} - r_k^{(n)}} \cdot r_k^{(n)}\,, \qquad k = 1, 2, \ldots\,; n = 0, 1, \ldots$$

with $G_0^{(n)} = S_n$ for $n = 0, 1, \ldots$.

The sequences $\left(G_k^{(n)}\right)_k$ and $\left(r_{k+1}^{(n)}\right)_k$, for a fixed value of n, can be obtained by the bordering method as explained in section 2.1.

We have

$$G_k^{(n)} = \sum_{j=0}^{k} A_j^{(k,n)} S_{n+j}$$

$$r_{k+1}^{(n)} = (-1)^k \sum_{j=0}^{k} A_j^{(k,n)} x_{n+k+j}$$

where the coefficients $A_j^{(k,n)}$ are solution of the system

$$\begin{cases} A_0^{(k,n)} & + \cdots + & A_k^{(k,n)} & = 1 \\ A_0^{(k,n)} x_n & + \cdots + & A_k^{(k,n)} x_{n+k} & = 0 \\ \vdots & & & \vdots \\ A_0^{(k,n)} x_{n+k-1} & + \cdots + & A_k^{(k,n)} x_{n+2k-1} & = 0. \end{cases}$$

The progressive form of the G-algorithm is given by

$$G_{k-1}^{(n+1)} = G_{k-1}^{(n)} + \left(G_k^{(n)} - G_{k-1}^{(n)}\right) \cdot \left(1 - \frac{r_k^{(n+1)}}{r_k^{(n)}}\right).$$

It can be implemented exactly as explained in section 2.1 for the progressive form of the E-algorithm by using alternately the preceding rule and the recurrence for the coefficients $A_j^{(k,n)}$.

The rs-algorithm has many interesting applications. For example if $S_n = x^n$, if we set

$$G_k^{(n)} = x^n P_k^{(n)}(x)$$

and if we define the linear functional c on the space of polynomials by

$$c(x^n) = x_n$$

and the linear functionals $c^{(i)}$ by

$$c^{(i)}(x^n) = x_{n+i}$$

then $P_k^{(n)}$ satisfies

$$c^{(n)}\left(x^i P_k^{(n)}(x)\right) = c\left(x^{n+i} P_k^{(n)}(x)\right) = 0\,, \qquad \text{for } i = 0, \ldots, k-1$$

since two rows in the numerator of the determinantal formula are identical. Thus the G and rs-algorithms can be used to compute recursively the polynomials $P_k^{(n)}$'s which form adjacent families of orthogonal polynomials with respect to the functional c. These recurrence relations are fully developed in Brezinski [60].

If $x_n = \Delta S_n$, then the G-transformation reduces to Shanks' and the G and rs-algorithms can thus be used for its implementation. Moreover many useful relations can be proved. They provide alternative methods for computing recursively the $e_k(S_n)$'s of Shanks' transformation. We have

$$\varepsilon_{2k}^{(n+1)} = \frac{s_k^{(n)}}{s_k^{(n+1)}} \cdot \left[\varepsilon_{2k}^{(n)} + \frac{r_{k+1}^{(n)}}{r_k^{(n+1)}} \cdot \varepsilon_{2k-2}^{(n+1)}\right]$$

$$\varepsilon_{2k+2}^{(n)} = \frac{s_k^{(n+1)}}{s_{k+1}^{(n)}} \cdot \left[\frac{r_{k+1}^{(n+1)}}{r_{k+1}^{(n)}} \cdot \varepsilon_{2k}^{(n)} - \varepsilon_{2k}^{(n+1)}\right]$$

$$\varepsilon_{2k}^{(n)} = \left(1 + \frac{r_{k+1}^{(n)}}{r_k^{(n+2)}}\right) \cdot \varepsilon_{2k}^{(n+1)} - \frac{r_{k+1}^{(n)}}{r_k^{(n+2)}} \cdot \varepsilon_{2k-2}^{(n+2)}$$

$$\varepsilon_{2k+2}^{(n-1)} = \left(1 + \frac{s_k^{(n+1)}}{s_{k+1}^{(n-1)}}\right) \cdot \varepsilon_{2k}^{(n)} - \frac{s_k^{(n+1)}}{s_{k+1}^{(n-1)}} \cdot \varepsilon_{2k}^{(n+1)}$$

$$\varepsilon_{2k}^{(n+1)} = \left(1 + \frac{r_{k+1}^{(n)} s_k^{(n)}}{s_k^{(n+1)} r_{k+1}^{(n-1)}}\right) \cdot \varepsilon_{2k}^{(n)} - \frac{r_{k+1}^{(n)} s_k^{(n)}}{s_k^{(n+1)} r_{k+1}^{(n-1)}} \cdot \varepsilon_{2k}^{(n-1)}$$

$$\varepsilon_{2k+2}^{(n-1)} = \left(1 + \frac{r_{k+1}^{(n)} s_k^{(n)}}{r_k^{(n+1)} s_{k+1}^{(n-1)}}\right) \cdot \varepsilon_{2k}^{(n)} - \frac{r_{k+1}^{(n)} s_k^{(n)}}{r_k^{(n+1)} s_{k+1}^{(n-1)}} \cdot \varepsilon_{2k-2}^{(n+1)}$$

$$\varepsilon_{2k+2}^{(n-2)} = \left(E_k^{(n)} - D_k^{(n)}\right) \cdot \varepsilon_{2k}^{(n)} - F_k^{(n)} \varepsilon_{2k-2}^{(n+2)}$$

with

$$D_k^{(n)} = \frac{r_{k+2}^{(n-2)} s_{k+1}^{(n-2)} - r_k^{(n+2)} s_{k-1}^{(n+2)}}{r_k^{(n+2)} s_{k-1}^{(n+2)} - r_{k+1}^{(n)} s_k^{(n)}} \cdot \frac{s_k^{(n)}}{s_{k+1}^{(n-2)}}$$

$$
\begin{aligned}
F_k^{(n)} &= \frac{r_{k+1}^{(n)} s_k^{(n)} - r_{k+2}^{(n-2)} s_{k+1}^{(n-2)}}{r_k^{(n+2)} s_{k-1}^{(n+2)} - r_{k+1}^{(n)} s_k^{(n)}} \cdot \frac{s_{k-1}^{(n+2)}}{s_{k+1}^{(n-2)}} \\
E_k^{(n)} &= 1 + D_k^{(n)} + F_k^{(n)} \\
\varepsilon_{2k+2}^{(n)} &= -\left(I_k^{(n)} + J_k^{(n)}\right) \cdot \varepsilon_{2k}^{(n)} - K_k^{(n)} \varepsilon_{2k-2}^{(n)}
\end{aligned}
$$

with

$$
\begin{aligned}
I_k^{(n)} &= \frac{s_k^{(n)}}{s_{k+1}^{(n)}} \\
J_k^{(n)} &= -K_k^{(n)} - I_k^{(n)} - 1 \\
K_k^{(n)} &= \frac{r_{k+1}^{(n)} s_k^{(n)}}{r_k^{(n)} s_{k+1}^{(n)}}.
\end{aligned}
$$

These eight relations can be used to follow an arbitrary road in the ε-array. They relate three ε's. There also exist relations between two adjacent ε's which can be used. They are

$$
\begin{aligned}
\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n)} &= -\frac{r_{k+1}^{(n)}}{s_{k+1}^{(n)}} \\
\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} &= \frac{r_{k+1}^{(n)}}{s_k^{(n+1)}} \\
\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+1)} &= -\frac{r_{k+1}^{(n+1)}}{s_{k+1}^{(n)}} \\
\varepsilon_{2k+2}^{(n)} - \varepsilon_{2k}^{(n+2)} &= -\frac{r_{k+2}^{(n)} + r_{k+1}^{(n+2)}}{s_{k+1}^{(n+1)}}.
\end{aligned}
$$

Of course these twelve relations can be used for computing recursively sequences of Padé approximants.

As we shall see in section 4.5, the rs-algorithm can also be used for implementing some vector sequence transformations.

For these reasons two subroutines have been written. One for the G-transformation using the G and the rs-algorithm, GTRAN, and one only for the rs-algorithm, RSALGO.

The qd-algorithm is an algorithm which is related to orthogonal polynomials, continued fractions, Padé approximants, the rs and ε-algo-

rithms. It consists in computing recursively the following ratios of determinants

$$q_{k+1}^{(n)} = \frac{H_{k+1}(x_{n+1}) \cdot H_k(x_n)}{H_k(x_{n+1}) \cdot H_{k+1}(x_n)}$$
$$e_{k+1}^{(n)} = \frac{H_{k+2}(x_n) \cdot H_k(x_{n+1})}{H_{k+1}(x_n) \cdot H_{k+1}(x_{n+1})}.$$

This algorithm, due to Rutishauser [378], is the following

$$e_0^{(n)} = 0\,, \quad q_1^{(n)} = \frac{x_{n+1}}{x_n}\,, \qquad n = 0, 1, \ldots$$
$$e_k^{(n)} = q_k^{(n+1)} + e_{k-1}^{(n+1)} - q_k^{(n)}\,, \qquad k = 1, 2, \ldots\,; n = 0, 1, \ldots$$
$$q_{k+1}^{(n)} = \frac{e_k^{(n+1)} q_k^{(n+1)}}{e_k^{(n)}}\,, \qquad k = 1, 2, \ldots\,; n = 0, 1, \ldots$$

It has been widely studied by several authors, see, for example, Henrici [226] for a quite complete exposition. It has applications in the computation of poles of meromorphic functions and was the starting point for the development of the LR algorithm for computing the eigenvalues of a matrix. It is related to the rs-algorithm by

$$q_{k+1}^{(n)} = \frac{r_{k+1}^{(n+1)} s_k^{(n+1)}}{r_{k+1}^{(n)} s_k^{(n)}}$$
$$e_{k+1}^{(n)} = \frac{r_{k+2}^{(n)} s_{k+1}^{(n)}}{r_{k+1}^{(n+1)} s_k^{(n+1)}}.$$

Thus, because of its multiple connections with other algorithms (described in Brezinski [60]), a particular subroutine, QDALGO, was devoted to it. For example if the qd-algorithm is applied to $x_n = \Delta^n S_0$ then

$$\varepsilon_{2k}^{(n)} = S_n \prod_{i=0}^{k} \frac{e_i^{(n)}}{q_i^{(n+1)}} \quad \text{and} \quad \varepsilon_{2k+1}^{(n)} = \frac{1}{\Delta S_n} \cdot \prod_{i=0}^{k} \frac{q_i^{(n+2)}}{e_i^{(n+1)}}.$$

There is, of course, a connection between the qd and the ω-algorithm which was studied in section 2.3. We have

$$\omega_{2k}^{(n)} = \omega_{2k-2}^{(n)} - \frac{[H_k(a_{n+1})]^2}{H_k(a_{n+2}) \cdot H_{k-1}(a_{n+2})}.$$

Thus if the qd-algorithm is applied to the sequence (a_n), then

$$\omega_{2k}^{(n)} = a_n \prod_{i=1}^{k} \frac{e_i^{(n)}}{q_i^{(n+1)}}$$

and

$$\omega_{2k+2}^{(n)} = \omega_{2k}^{(n)} \cdot \frac{e_{k+1}^{(n)}}{q_{k+1}^{(n+1)}}.$$

Particular rules for the rs and ω-algorithms were recently obtained by Fang Chen [155]. They seem to be quite interesting but they still need to be studied in more details.

2.5 Rational extrapolation

We shall now construct a sequence transformation whose kernel is the set sequences of the form

$$S_n = \frac{Sx_n^k + a_1 x_n^{k-1} + \cdots + a_k}{x_n^k + b_1 x_n^{k-1} + \cdots + b_k}$$

where (x_n) is an auxiliary given sequence. Of course, this relation can be written as

$$S_n = S - b_1 x_n^{-1} S_n + a_1 x_n^{-1} - \cdots - b_k x_n^{-k} S_n + a_k x_n^{-k}$$

which shows that this transformation can be put into the framework of the E-algorithm by setting

$$g_{2i-1}(n) = x_n^{-i} S_n\,, \qquad g_{2i}(n) = x_n^{-i}\,, \qquad i = 1, \ldots, k.$$

The numbers $E_k^{(n)}$, denoted by $\varrho_{2k}^{(n)}$ in this particular case, are given by

$$\varrho_{2k}^{(n)} = \frac{\begin{vmatrix} 1 & S_n & x_n & x_n S_n & \cdots & x_n^{k-1} & x_n^{k-1} S_n & x_n^k S_n \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 1 & S_{n+2k} & x_{n+2k} & x_{n+2k} S_{n+2k} & \cdots & x_{n+2k}^{k-1} & x_{n+2k}^{k-1} S_{n+2k} & x_{n+2k}^k S_{n+2k} \end{vmatrix}}{\begin{vmatrix} 1 & S_n & x_n & x_n S_n & \cdots & x_n^{k-1} & x_n^{k-1} S_n & x_n^k \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 1 & S_{n+2k} & x_{n+2k} & x_{n+2k} S_{n+2k} & \cdots & x_{n+2k}^{k-1} & x_{n+2k}^{k-1} S_{n+2k} & x_{n+2k}^k \end{vmatrix}}.$$

Thus, by construction, the rational function

$$R_{2k}(x) = \frac{\varrho_{2k}^{(n)} x^k + a_1 x^{k-1} + \cdots + a_k}{x^k + b_1 x^{k-1} + \cdots + b_k}$$

satisfies the interpolation conditions

$$R_{2k}(x_{n+i}) = S_{n+i}\,, \qquad \text{for } i = 0, \ldots, 2k.$$

Since $\lim_{x \to \infty} R_{2k}(x) = \varrho_{2k}^{(n)}$, this shows that the sequence transformation $(S_n) \longmapsto \left(\varrho_{2k}^{(n)}\right)$ performs rational extrapolation at infinity thus generalizing polynomial extrapolation obtained by Richardson's process (see section 2.2).

The numbers $\varrho_{2k}^{(n)}$, together with the intermediate $\varrho_{2k+1}^{(n)}$'s, are called reciprocal differences and they are related by an algorithm very much similar to the ε-algorithm. It is the so-called ϱ-algorithm which was first used by Wynn with $x_n = n$ [471] for the purpose of convergence acceleration. It is as follows

$$\varrho_{-1}^{(n)} = 0\,, \quad \varrho_0^{(n)} = S_n\,, \qquad n = 0, 1, \ldots$$

$$\varrho_{k+1}^{(n)} = \varrho_{k-1}^{(n+1)} + \frac{x_{n+k+1} - x_n}{\varrho_k^{(n+1)} - \varrho_k^{(n)}}\,, \qquad k, n = 0, 1, \ldots$$

These interpolating rational functions can be constructed by means of continued fractions. It is an interpolation process due to Thiele [432] and studied by Nörlund [337]. Let us consider the continued fraction

$$C^{(n)}(x) = \alpha_0^{(n)} + \cfrac{x - x_n}{\alpha_1^{(n)}} + \cfrac{x - x_{n+1}}{\alpha_2^{(n)}} + \cdots$$

with $\alpha_k^{(n)} = \varrho_k^{(n)} - \varrho_{k-2}^{(n)}$ for $k = 1, 2, \ldots$ and $\alpha_0^{(n)} = \varrho_0^{(n)} = f(x_n)$. Let $C_k^{(n)}(x)$ be its k-th convergent (on continued fractions see, for example, Jones and Thron [250] or Cuyt and Wuytack [131]), then the following interpolation property holds

$$C_k^{(n)}(x_i) = f(x_i)\,, \qquad \text{for } i = n, \ldots, n + k.$$

If we set $C_k^{(n)}(x) = A_k^{(n)}(x)/B_k^{(n)}(x)$ then $A_{2k-1}^{(n)}, A_{2k}^{(n)}, B_{2k}^{(n)}$ and $B_{2k+1}^{(n)}$ are polynomials of degree k in x and we have

$$A_{-1}^{(n)}(x) = 1 \qquad A_0^{(n)}(x) = \alpha_0^{(n)}$$
$$B_{-1}^{(n)}(x) = 0 \qquad B_0^{(n)}(x) = 1$$

and for $k = 1, 2, \ldots$

$$A_k^{(n)}(x) = \alpha_k^{(n)} A_{k-1}^{(n)}(x) + (x - x_{n+k-1})\, A_{k-2}^{(n)}(x)$$

$$B_k^{(n)}(x) = \alpha_k^{(n)} B_{k-1}^{(n)}(x) + (x - x_{n+k-1})\, B_{k-2}^{(n)}(x).$$

If we set

$$\begin{aligned} A_{2k}^{(n)}(x) &= \sum_{i=0}^{k} a_i^{(2k)} x^i \\ A_{2k-1}^{(n)}(x) &= \sum_{i=0}^{k} a_i^{(2k-1)} x^i \\ B_{2k+1}^{(n)}(x) &= \sum_{i=0}^{k} b_i^{(2k+1)} x^i \\ B_{2k}^{(n)}(x) &= \sum_{i=0}^{k} b_i^{(2k)} x^i \end{aligned}$$

then the preceding relations can be used to compute recursively the coefficients $a_i^{(k)}$ and $b_i^{(k)}$ (which depend also on the index n but it was omitted for simplicity) and they give

$$a_0^{(0)} = \alpha_0^{(n)}, \qquad a_0^{(-1)} = 1$$

$$b_0^{(0)} = 1, \qquad b_0^{(-1)} = 0$$

$$\begin{aligned} a_0^{(2k-1)} &= \alpha_{2k-1}^{(n)} a_0^{(2k-2)} - x_{n+2k-2} a_0^{(2k-3)} \\ a_i^{(2k-1)} &= \alpha_{2k-1}^{(n)} a_i^{(2k-2)} - x_{n+2k-2} a_i^{(2k-3)} + a_{i-1}^{(2k-3)}, \quad i = 1, \ldots, k-1 \\ a_k^{(2k-1)} &= a_{k-1}^{(2k-3)} \\ a_0^{(2k)} &= \alpha_{2k}^{(n)} a_0^{(2k-1)} - x_{n+2k-1} a_0^{(2k-2)} \\ a_i^{(2k)} &= \alpha_{2k}^{(n)} a_i^{(2k-1)} - x_{n+2k-1} a_i^{(2k-2)} + a_{i-1}^{(2k-2)}, \quad i = 1, \ldots, k-1 \\ a_k^{(2k)} &= \alpha_{2k}^{(n)} a_k^{(2k-1)} + a_{k-1}^{(2k-2)} \\ b_0^{(2k-1)} &= \alpha_{2k-1}^{(n)} b_0^{(2k-2)} - x_{n+2k-2} b_0^{(2k-3)} \\ b_i^{(2k-1)} &= \alpha_{2k-1}^{(n)} b_i^{(2k-2)} - x_{n+2k-2} b_i^{(2k-3)} + b_{i-1}^{(2k-3)}, \quad i = 1, \ldots, k-2 \\ b_{k-1}^{(2k-1)} &= \alpha_{2k-1}^{(n)} b_{k-1}^{(2k-2)} + b_{k-2}^{(2k-3)} \\ b_0^{(2k)} &= \alpha_{2k}^{(n)} b_0^{(2k-1)} - x_{n+2k-1} b_0^{(2k-2)} \end{aligned}$$

$$\begin{aligned} b_i^{(2k)} &= \alpha_{2k}^{(n)} b_i^{(2k-1)} - x_{n+2k-1} b_i^{(2k-2)} + b_{i-1}^{(2k-2)}, \quad i = 1, \ldots, k-1 \\ b_k^{(2k)} &= b_{k-1}^{(2k-2)}. \end{aligned}$$

It is easy to see that

$$\begin{aligned} a_k^{(2k-1)} &= 1 & b_{k-1}^{(2k-1)} &= \varrho_{2k-1}^{(n)} \\ a_k^{(2k)} &= \varrho_{2k}^{(n)} & b_k^{(2k)} &= 1. \end{aligned}$$

Let us give a numerical example showing that this method can be affected by rounding errors due to the computer's arithmetic. For that we start from a function f which is a rational function

$$f(x) = \frac{0.75x^{k+1} + 2x^k + 0.3}{x^k + 1.2x^{k-1} + 1.8} = \frac{a_{k+1}x^{k+1} + a_k x^k + a_0}{b_k x^k + b_{k-1}x^{k-1} + b_0}.$$

Of course C_{2k+1} must be identical to f. We obtained the following results with a computer working with 16 decimal digits

$k = 1$

$$\begin{aligned} a_0 &= 3.000000000000010 \cdot 10^{-1} & b_0 &= 3.000000000000010 \\ a_1 &= 2.000000000000006 & b_1 &= 1.0 \\ a_2 &= 7.500000000000004 \cdot 10^{-1} \end{aligned}$$

$k = 2$

$$\begin{aligned} a_0 &= 3.000000000000084 \cdot 10^{-1} & b_0 &= 1.800000000000138 \\ a_1 &= -1.17 \cdot 10^{-13} & b_1 &= 1.200000000000010 \\ a_2 &= 2.000000000000003 & b_2 &= 1.0 \\ a_3 &= 7.500000000000037 \cdot 10^{-1} \end{aligned}$$

$k = 3$

$$\begin{aligned} a_0 &= 3.000000000000049 \cdot 10^{-1} & b_0 &= 1.800000000000036 \\ a_1 &= -8.68 \cdot 10^{-15} & b_1 &= -5.06 \cdot 10^{-17} \\ a_2 &= -6.62 \cdot 10^{-15} & b_2 &= 1.199999999999997 \\ a_3 &= 2.000000000000036 & b_3 &= 1.0 \\ a_4 &= 7.499999999999916 \cdot 10^{-1} \end{aligned}$$

$k = 4$

$$\begin{aligned} a_0 &= 3.000000000005043 \cdot 10^{-1} & b_0 &= 1.800000000003028 \\ a_1 &= -5.36 \cdot 10^{-13} & b_1 &= -3.22 \cdot 10^{-12} \\ a_2 &= -8.19 \cdot 10^{-13} & b_2 &= -5.10 \cdot 10^{-12} \\ a_3 &= -1.04 \cdot 10^{-12} & b_3 &= 1.199999999995994 \\ a_4 &= 2.000000000002907 & b_4 &= 1.0 \\ a_5 &= 7.499999999991787 \cdot 10^{-1} \end{aligned}$$

$k = 5$

$a_0 = 2.999999996357963 \cdot 10^{-1}$ $b_0 = 1.799999997814911$
$a_1 = 1.18 \cdot 10^{-10}$ $b_1 = 7.08 \cdot 10^{-10}$
$a_2 = 1.63 \cdot 10^{-10}$ $b_2 = 9.61 \cdot 10^{-10}$
$a_3 = 8.61 \cdot 10^{-10}$ $b_3 = 5.19 \cdot 10^{-9}$
$a_4 = 6.27 \cdot 10^{-10}$ $b_4 = 1.200000002753679$
$a_5 = 1.999999997779250$ $b_5 = 1.0$
$a_6 = 7.500000004052835 \cdot 10^{-1}$

$k = 7$

$a_0 = 2.99925 \cdot 10^{-1}$ $b_0 = 1.79955$
$a_1 = 4.87 \cdot 10^{-5}$ $b_1 = 2.92 \cdot 10^{-4}$
$a_2 = -3.73 \cdot 10^{-5}$ $b_2 = -2.24 \cdot 10^{-4}$
$a_3 = -4.39 \cdot 10^{-5}$ $b_3 = -2.63 \cdot 10^{-4}$
$a_4 = 1.12 \cdot 10^{-4}$ $b_4 = 6.76 \cdot 10^{-4}$
$a_5 = 3.24 \cdot 10^{-4}$ $b_5 = 1.94 \cdot 10^{-3}$
$a_6 = 2.34 \cdot 10^{-4}$ $b_6 = 1.2010$
$a_7 = 1.99950$ $b_7 = 1.0$
$a_8 = 7.50041 \cdot 10^{-1}$

Then the precision downgrades much more rapidly since for $k = 9$ the non-zero coefficients have no exact digits.

Let us now come back to the ϱ-algorithm.

If the x_n's are invariant by translation of the sequence (S_n) or if the ϱ-algorithm is computed with the same x_n's then, if (S_n) is replaced by $(aS_n + b)$, $\varrho_{2k}^{(n)}$ becomes $a\varrho_{2k}^{(n)} + b$ while $\varrho_{2k+1}^{(n)}$ becomes $\varrho_{2k+1}^{(n)}/a$. Moreover if (S_n) is replaced by $((aS_n + b)/(cS_n + d))$ then $\varrho_{2k}^{(n)}$ becomes $\left(a\varrho_{2k}^{(n)} + b\right)/\left(c\varrho_{2k}^{(n)} + d\right)$.

For a fixed value of n, the sequence $\left(\varrho_{2k}^{(n)}\right)_k$ can be computed by the bordering method. However if we want to use the progressive form of the ϱ-algorithm, the sequence $\left(\varrho_{2k+1}^{(n)}\right)_k$ must also be known which is possible since these quantities are related to the solution of a system of linear equations.

Let R_{2k+1} be the rational function with a numerator of degree $k+1$ and a denominator of degree k satisfying the interpolation conditions

$$R_{2k+1}(x_{n+i}) = S_{n+i}\,, \qquad \text{for } i = 0, \ldots, 2k+1,$$

then it can be proved that R_{2k+1} has the form

$$C_{2k+1}^{(0)}(x) = R_{2k+1}(x) = \frac{x^{k+1} + a_1 x^k + \cdots + a_{k+1}}{\varrho_{2k+1}^{(n)} x^k + b_1 x^{k-1} + \cdots + b_k}.$$

Thus the system

$$S_{n+i} x_{n+i}^k \varrho_{2k+1}^{(n)} + b_1 S_{n+i} x_{n+i}^{k-1} + \cdots + b_k S_{n+i} - a_1 x_{n+i}^k - \cdots - a_{k+1} = x_{n+i}^{k+i}$$

for $i = 0, \ldots, 2k+1$ can be solved by the bordering method for a fixed value of n and it gives the sequence $\left(\varrho_{2k+1}^{(n)}\right)_k$. Then the progressive form of the ϱ-algorithm can be used

$$\varrho_{k+1}^{(n+1)} = \varrho_{k+1}^{(n)} + \frac{x_{n+k+2} - x_n}{\varrho_{k+2}^{(n)} - \varrho_k^{(n+1)}}.$$

Since the $\varrho_{2k+1}^{(n)}$ are only intermediate quantities they can be eliminated from the rule of the algorithm, thus leading to a cross rule for the ϱ-algorithm. But before doing this, let us generalize the rule of the ϱ-algorithm since it will be useful later. We consider an algorithm (the γ-algorithm) of the form

$$\gamma_{-1}^{(n)} = 0\,, \quad \gamma_0^{(n)} = S_n\,, \qquad n = 0, 1, \ldots$$

$$\gamma_{k+1}^{(n)} = \gamma_{k-1}^{(n+1)} + \frac{a_k^{(n)}}{\gamma_k^{(n+1)} - \gamma_k^{(n)}}\,, \qquad k, n = 0, 1, \ldots$$

This form includes the ε and the ϱ-algorithm since $\gamma_k^{(n)} = \varepsilon_k^{(n)}$ if $a_k^{(n)} = 1\ \forall k, n$, while $\gamma_k^{(n)} = \varrho_k^{(n)}$ if $a_k^{(n)} = x_{n+k+1} - x_n$.

In section 2.6 we shall study two other algorithms of this form. Thus it will be useful to derive a cross rule for all these algorithms. Such a rule was obtained by Brezinski [40] who proved that the following relation holds

$$a_{2k+1}^{(n)} \left[\gamma_{2k+2}^{(n)} - \gamma_{2k}^{(n+1)}\right]^{-1} + a_{2k-1}^{(n+1)} \left[\gamma_{2k-2}^{(n+2)} - \gamma_{2k}^{(n+1)}\right]^{-1} =$$

$$a_{2k}^{(n+1)} \left[\gamma_{2k}^{(n+2)} - \gamma_{2k}^{(n+1)}\right]^{-1} + a_{2k}^{(n)} \left[\gamma_{2k}^{(n)} - \gamma_{2k}^{(n+1)}\right]^{-1}$$

with $\gamma_{-2}^{(n)} = \infty$ and $\gamma_0^{(n)} = S_n$ for $n = 0, 1, \ldots$.

Thus if, for a fixed value of n, the sequences $\left(\gamma_{2k}^{(n)}\right)_k$ and $\left(\gamma_{2k}^{(n+1)}\right)_k$ have been computed by the bordering method then the progressive form of the above cross rule can be used to compute the whole array of the γ's. It is

$$\gamma_{2k}^{(n+2)} = \gamma_{2k}^{(n+1)} + a_{2k}^{(n+1)} \left\{ a_{2k+1}^{(n)} \left[\gamma_{2k+2}^{(n)} - \gamma_{2k}^{(n+1)}\right]^{-1} + a_{2k-1}^{(n+1)} \left[\gamma_{2k-2}^{(n+2)} - \gamma_{2k}^{(n+1)}\right]^{-1} - a_{2k}^{(n)} \left[\gamma_{2k}^{(n)} - \gamma_{2k}^{(n+1)}\right]^{-1} \right\}^{-1}.$$

As was the case for the ε-algorithm (see section 1.9) a particular rule for an isolated singularity can be obtained for these algorithms. It is

$$\gamma_{k+1}^{(n)} = r \left(a_k^{(n)} + \frac{r}{\gamma_{k-1}^{(n+1)}} \right)^{-1}$$

where

$$r = \left(a_{k-1}^{(n+1)} + a_{k-1}^{(n)} - a_{k-2}^{(n+1)} - a_k^{(n)}\right)\gamma_{k-1}^{(n+1)} + a_{k-1}^{(n+1)}\gamma_{k-1}^{(n+2)} \left[1 - \frac{\gamma_{k-1}^{(n+2)}}{\gamma_{k-1}^{(n+1)}}\right]^{-1} + a_{k-1}^{(n)}\gamma_{k-1}^{(n)} \left[1 - \frac{\gamma_{k-1}^{(n)}}{\gamma_{k-1}^{(n+1)}}\right]^{-1} - a_{k-2}^{(n+1)}\gamma_{k-3}^{(n+2)} \left[1 - \frac{\gamma_{k-3}^{(n+2)}}{\gamma_{k-1}^{(n+1)}}\right]^{-1}.$$

It holds for odd and even values of k.

The first term in the expression of r vanishes if and only if, $\forall k, n$

$$a_{k-1}^{(n+1)} + a_{k-1}^{(n)} = a_{k-2}^{(n+1)} + a_k^{(n)}.$$

Algorithms satisfying this property (as is the case for the ε and ϱ-algorithms) can be transformed by successive inversions into more stable forms which can be used even in the case of several adjacent singularities, a technique due to Cordellier [120].

This property is called homographic invariance.

If $\gamma_{k-2}^{(n+1)} = \gamma_{k-2}^{(n+2)}$ then $\gamma_{k-1}^{(n+1)}$ is infinity and, for an algorithm satisfying the homographic invariance property, the singular rule reduces to

$$\gamma_{k+1}^{(n)} = \left(a_{k-1}^{(n+1)}\gamma_{k-1}^{(n+2)} + a_{k-1}^{(n)}\gamma_{k-1}^{(n)} - a_{k-2}^{(n+1)}\gamma_{k-3}^{(n+2)}\right) \Big/ a_k^{(n)}.$$

The subroutines EPSRHO and EPSRHA contain the above particular rule.

On this question and on a more general ϱ-algorithm, see Carstensen [113]. Other techniques for rational extrapolation are described by Wuytack [468], Bulirsch and Stoer [106] and Larkin [273].

2.6 Generalizations of the ε-algorithm

We shall now study two generalizations of the ε-algorithm. These generalizations, due to Brezinski [43], were not obtained from the kernel as a starting point but by modifying directly the rule of the ε-algorithm and then looking for the properties of the new sequence transformations thus obtained. As explained in section 1.2 it is much more difficult to proceed like that and only a few properties of these transformations are actually known. However, since they provide, in some examples, better numerical results than other algorithms, we shall now discuss them.

In section 2.5 we consider the γ-algorithm which covered the ε-algorithm (for $a_k^{(n)} = 1$) and the ϱ-algorithm (for $a_k^{(n)} = x_{n+k+1} - x_n$). Thus the idea came to use other choices for the $a_k^{(n)}$'s related to a given auxiliary sequence (x_n).

The first generalization of the ε-algorithm corresponds to the choice

$$a_k^{(n)} = \Delta x_n = x_{n+1} - x_n.$$

The second generalization of the ε-algorithm corresponds to the choice

$$a_k^{(n)} = \Delta x_{n+k} = x_{n+k+1} - x_{n+k}.$$

As explained in the first chapter, the study of a given algorithm has to begin with the kernel of the transformation which it implements. For finding this kernel one has first to express the quantities which are computed by the algorithm as ratios of determinants. Then the implicit form of the kernel can be obtained (or sometimes its explicit form if the determinants involved in the ratios are not too much complicated), usually as a (linear or nonlinear) difference equation. Solving this difference equation leads to the explicit form of the kernel. As shall see below it was possible to obtain a ratio of determinants for the first generalization

of the ε-algorithm and then the implicit form of the kernel. It was possible to solve the corresponding difference equation only for $k = 1$ thus giving the kernel of the sequence transformation corresponding to the first even column of the array of this generalization. For the second generalization of the ε-algorithm it has not yet been possible to express the quantities computed as ratios of determinants. However it was possible to obtain directly from the rule of the algorithm the explicit form of the kernel for the transformation corresponding to the second column.

Let us begin with the first generalization of the ε-algorithm. Its rules are

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0\,, \quad \varepsilon_0^{(n)} = S_n\,, && n = 0,1,\ldots \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \frac{x_{n+1} - x_n}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}\,, && k,n = 0,1,\ldots \end{aligned}$$

Of course, although we used the same letter ε, the quantities computed by this algorithm are different from those obtained by the ε-algorithm unless $x_{n+1} - x_n = 1$ for all n.

To obtain determinantal formulæ for the ε's computed by this generalization, we first have to generalize the operator Δ and its successive powers. We know that if f is a linear function of $S_{p_1}, \ldots, S_{p_n}$ where $p_1, \ldots, p_n$ are arbitrary integer indexes, then

$$\Delta f(S_{p_1}, \ldots, S_{p_n}) = f(r_{p_1}, \ldots, r_{p_n})$$

where $r_j = \Delta S_j$.

Let us now set $r_j = \Delta S_j / \Delta x_j$ and define the operator R in the same way by

$$Rf(S_{p_1}, \ldots, S_{p_n}) = f(r_{p_1}, \ldots, r_{p_n})\,.$$

Clearly R generalizes Δ which is recovered if $\forall n, \Delta x_n = 1$. R is a linear operator which transforms a constant sequence into zero.

Let $v_n = S_n \cdot \Delta x_n$, then the successive powers of R are such that

$$R^{k+1} v_n = \Delta\left(\frac{R^k v_n}{\Delta x_n}\right) = R^k\left(\Delta\left(\frac{v_n}{\Delta x_n}\right)\right),$$

and it is possible to prove the

Theorem 2.28

For the first generalization of the ε-algorithm if we set $\varepsilon_{2k}^{(n)} = e_k(S_n)$ *then*

$$e_k(S_n) = \frac{\begin{vmatrix} v_n & \cdots & v_{n+k} \\ Rv_n & \cdots & Rv_{n+k} \\ \vdots & & \vdots \\ R^k v_n & \cdots & R^k v_{n+k} \end{vmatrix}}{\begin{vmatrix} \Delta x_n & \cdots & \Delta x_{n+k} \\ Rv_n & \cdots & Rv_{n+k} \\ \vdots & & \vdots \\ R^k v_n & \cdots & R^k v_{n+k} \end{vmatrix}}$$

and

$$\varepsilon_{2k+1}^{(n)} = 1/e_k(r_n) = \frac{\begin{vmatrix} \Delta x_n & \cdots & \Delta x_{n+k} \\ R^2 v_n & \cdots & R^2 v_{n+k} \\ \vdots & & \vdots \\ R^{k+1} v_n & \cdots & R^{k+1} v_{n+k} \end{vmatrix}}{\begin{vmatrix} Rv_n & \cdots & Rv_{n+k} \\ \vdots & & \vdots \\ R^{k+1} v_n & \cdots & R^{k+1} v_{n+k} \end{vmatrix}}.$$

Thus this generalization can be put into the framework of the E-algorithm with $g_i(n) = R^i v_n / \Delta x_n$ and thus we have the

Theorem 2.29

For the first generalization of the ε-algorithm, a necessary and sufficient condition that $\forall n, \varepsilon_{2k}^{(n)} = S$ *is that* $\forall n$

$$(S_n - S) \cdot \Delta x_n = \sum_{i=1}^{k} a_i R^i v_n.$$

Since R reduces to Δ if $\forall n, \Delta x_n = 1$ then clearly this kernel is a generalization of the kernel of Shanks' transformation. It has not yet been possible to solve this difference equation for an arbitrary value of k. However for $k = 1$ we have the

Theorem 2.30

For the first generalization of the ε-algorithm, a necessary and sufficient condition that $\forall n, \varepsilon_2^{(n)} = S$ *is that* $\forall n$

$$S_n = S + a \prod_{i=0}^{n-1} \lambda_i$$

with $\lambda_i = 1 + c\Delta x_i$.

Clearly if $\forall i, \Delta x_i = 1$, the kernel is the set of sequences of the form $S_n = S + a\lambda^n$ which is the kernel of Aitken's Δ^2 process.

The first generalization of the ε-algorithm is quasi-linear. More precisely we have for the even columns

$$e_k(aS_n + b) = a\, e_k(S_n) + b$$

and for the odd ones

$$\frac{1}{e_k(ar_n + b)} = \frac{1}{a\, e_k(r_n)}$$

if the auxiliary sequence (x_n) is invariant by translation of (S_n) or if the same auxiliary sequence is used in both cases.

Let us now come to the second generalization of the ε-algorithm whose rules are (still using the same letter ε)

$$\varepsilon_{-1}^{(n)} = 0\,, \quad \varepsilon_0^{(n)} = S_n\,, \qquad n = 0, 1, \ldots$$

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + \frac{x_{n+k+1} - x_{n+k}}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}\,, \qquad k, n = 0, 1, \ldots$$

For this second generalization, it has not yet been possible to obtain results similar to those of theorems 2.28 and 2.29. However, from the rule of the algorithm, it can be seen directly that the same quasi-linearity property holds and that we have the

Theorem 2.31

For the second generalization of the ε-algorithm, a necessary and sufficient condition that $\forall n, \varepsilon_2^{(n)} = S$ *is that* $\forall n$

$$S_n = S + a \prod_{i=0}^{n-1} \lambda_i$$

with $\lambda_i = 1/(1 + c\Delta x_i)$.

Clearly we recover again the kernel of Aitken's Δ^2 process if $\forall i, \Delta x_i = 1$ (or any arbitrary constant different from zero).

These two generalizations can produce much better results than the ε-algorithm for some sequences.

Let us take $S_n = 1 + 3e^{-1.4x_n}$ with $x_n = 1.1^{n-1}$. This sequence converges to 1 and we have $S_0 = 1.73979$, $S_1 = 1.64314, \ldots$, $S_4 = 1.38631$. We obtain

with the ε-algorithm	$\varepsilon_4^{(0)} = 1.73799$
with its first generalization	$\varepsilon_4^{(0)} = 1.00272$
with its second generalization	$\varepsilon_4^{(0)} = 1.00224$

Let us take now $S_n = 1 + 1/(n+1)$ and $x_n = \ln(n+2)$. We have $S_{36} = 1.027$ and

with the ε-algorithm	$\varepsilon_{36}^{(0)} = 1.004$
with its first generalization	$\varepsilon_{36}^{(0)} = 1.000008$
with its second generalization	$\varepsilon_{36}^{(0)} = 1.00000002$

Finally let us consider $S_n = (n+1)\sin(n+1)^{-1}$ and $x_n = \ln(n+2)$. We have $S_{36} = 0.99987$ and

with the ε-algorithm	$\varepsilon_{36}^{(0)} = 0.9999938$
with its first generalization	$\varepsilon_{36}^{(0)} = 1.0000000027$
with its second generalization	$\varepsilon_{36}^{(0)} = 0.99999999976$

For the first and the second generalization of the ε-algorithm, we have respectively

$$\frac{\varepsilon_2^{(n)} - S}{S_{n+1} - S} = 1 - \frac{(S_{n+2} - S)/(S_{n+1} - S) - 1}{\Delta S_{n+1}/\Delta S_n - \Delta x_{n+1}/\Delta x_n} \qquad (1^{st} \text{ generalization})$$

$$\frac{\varepsilon_2^{(n)} - S}{S_{n+1} - S} = 1 - \frac{\Delta x_{n+1}}{\Delta x_n} \cdot \frac{(S_{n+2} - S)/(S_{n+1} - S) - 1}{\Delta S_{n+1}/\Delta S_n - \Delta x_{n+1}/\Delta x_n} \quad (2^{nd} \text{ generalization}).$$

Thus it can be immediately seen that if $\exists a \neq 1$ such that $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = a$ and if $\lim_{n\to\infty} \Delta x_{n+1}/\Delta x_n = 1$ then, for both generalizations, $\left(\varepsilon_2^{(n)}\right)$ converges to S faster than (S_{n+1}) where S is the limit

of (S_n). Moreover if (S_n) converges to S logarithmically, that is if $\lim_{n\to\infty}(S_{n+1} - S)/(S_n - S) = 1$, then the convergence can be accelerated in some cases, namely if and only if the ratios in the right hand sides of the above expressions tend to 1 when n tends to infinity.

The first and second generalizations of the ε-algorithm are included in the subroutines EPSRHO and EPSRHA. Their particular rules were given in section 2.5 (γ-algorithm).

2.7 Levin's transforms

Levin's transforms can be considered as generalizations of Aitken's Δ^2 process and of the transformation E_1 corresponding to the first column of the E-algorithm.

In section 1.3, we saw that the kernel of Aitken's process is the set of sequences of the form

$$S_n - S = a \cdot \Delta S_n\,, \qquad n = 0, 1, \ldots$$

In section 2.1, were saw that the kernel of the transformation E_1 is the set of sequences of the form

$$S_n - S = a \cdot g(n)\,, \qquad n = 0, 1, \ldots$$

where $(g(n))$ is a given auxiliary sequence.

In both cases the constant a can be considered as a polynomial of degree 0. Let us now generalize and replace this polynomial of degree 0 by an arbitrary polynomial of degree $k-1$ in $(n+b)^{-1}$ where b is any non-zero real constant which is not a negative integer. Thus we are now considering sequences of the form

$$S_n - S = g(n) \cdot \left(a_1 + a_2(n+b)^{-1} + \cdots + a_k(n+b)^{-(k-1)}\right), \quad n=0, 1, \ldots$$

Of course, such sequences fit into the general framework of the E-algorithm since they correspond to the choice $g_i(n) = g(n)(n+b)^{-(i-1)}$ for $i = 1, \ldots, k$. Thus the numbers $E_k^{(n)}$, that we shall denote by $L_k^{(n)}$ in this case, can be computed by the E-algorithm. But the E-algorithm does not simplify in general and it is too complicated since a much simpler algorithm can be derived.

Multiplying both sides of the preceding relation by $(n+b)^{k-1}$ gives

$$(n+b)^{k-1} \cdot \frac{S_n - S}{g(n)} = a_1(n+b)^{k-1} + a_2(n+b)^{k-2} + \cdots + a_k.$$

If the operator Δ^k is applied to the right hand side of this expression we obtain zero identically since it is a polynomial of degree $k-1$ in n. Thus, $\forall n$

$$\Delta^k \left((n+b)^{k-1} \cdot \frac{S_n - S}{g(n)} \right) = 0.$$

Since Δ^k is a linear operator we have

$$\Delta^k \left((n+b)^{k-1} \cdot \frac{S_n}{g(n)} \right) = S \cdot \Delta^k \left(\frac{(n+b)^{k-1}}{g(n)} \right),$$

and $L_k^{(n)}$ is given by

$$L_k^{(n)} = \frac{\Delta^k \left((n+b)^{k-1} \cdot S_n / g(n) \right)}{\Delta^k \left((n+b)^{k-1} / g(n) \right)}, \qquad n = 0, 1, \ldots$$

According to the various choices of the auxiliary sequence $(g(n))$ we recover the transformations of Levin [283]

- for $g(n) = (n+b) \cdot \Delta S_{n-1}$, it is his u-transform,
- for $g(n) = \Delta S_{n-1}$, it is his t-transform,
- for $g(n) = -\Delta S_{n-1} \Delta S_n / \Delta^2 S_{n-1}$, it is his v-transform.

For these choices we set $S_{-1} = 0$.

Of course many other choices are possible, such as $g(n) = \Delta S_n$ which was proposed by Smith and Ford [413] and provides the best simple remainder estimate for a sequence with strictly alternating differences ΔS_n. The various Levin's transforms were experimentally showed by Smith and Ford [414] to accelerate the convergence of a wide collection of sequences.

If $(g(n))$ is invariant by translation or if the same $(g(n))$ is used then Levin's transforms are translative.

The numbers $L_k^{(n)}$ can be computed directly by using the well known formula

$$\Delta^k u_n = \sum_{i=0}^{k} (-1)^i C_k^i u_{n+i}$$

with $C_k^i = k!/i!(k-i)!$ and we have

$$L_k^{(n)} = \frac{\sum_{i=0}^{k} (-1)^i \cdot C_k^i \cdot (n+i+b)^{k-1} \cdot S_{n+i}/g(n+i)}{\sum_{i=0}^{k} (-1)^i \cdot C_k^i \cdot (n+i+b)^{k-1}/g(n+i)}.$$

Due to the properties of the operator Δ^k and of the binomial coefficients C_k^i, the numerators and the denominators of the $L_k^{(n)}$'s can be separately recursively computed by an algorithm due to Fessler, Ford and Smith [158]. We set

$$L_k^{(n)} = \frac{N_k^{(n)}}{D_k^{(n)}}.$$

Then

$$N_0^{(n)} = \frac{S_n}{g(n)}, \qquad n = 0, 1, \ldots$$

$$N_{k+1}^{(n)} = N_k^{(n+1)} - \frac{(n+b) \cdot (n+k+b)^{k-1}}{(n+k+b+1)^k} \cdot N_k^{(n)}, \qquad k, n, = 0, 1, \ldots$$

and

$$D_0^{(n)} = \frac{1}{g(n)}, \qquad n = 0, 1, \ldots$$

$$D_{k+1}^{(n)} = D_k^{(n+1)} - \frac{(n+b) \cdot (n+k+b)^{k-1}}{(n+k+b+1)^k} \cdot D_k^{(n)}, \qquad k, n, = 0, 1, \ldots$$

The factor $(n+k+b)^{k-1}$ and $(n+k+b+1)^k$ have been arbitrarily introduced into the algorithm in order to make the computations more stable. Let us mention that a subroutine can be found in Fessler, Ford and Smith [157].

The following result was proved by Sidi [404] when $b = 1$

Theorem 2.32

If $S_n = S + g(n)f(n)$

where $f(n) \sim \sum_{i=0}^{\infty} \beta_i / n^i$ *with* $\beta_0 \neq 0$ *when* $n \to \infty$,

and $g(n) \sim \sum_{i=0}^{\infty} \delta_i / n^{a+i}$ *with* $a > 0, \delta_0 \neq 0$ *when* $n \to \infty$

then, if $\beta_k \neq 0$

$$L_k^{(n)} - S \sim \frac{\delta_0 \beta_k}{\binom{-a}{k}} \cdot n^{-a-k} \qquad (n \to \infty).$$

We can generalize Levin's transform one step further by considering a polynomial of degree $k-1$ in x_n^{-1} instead of $(n+b)^{-1}$. Thus we are assuming that (S_n) has the form

$$S_n - S = g(n) \cdot \left(a_1 + a_2 x_n^{-1} + \cdots + a_k x_n^{-(k-1)}\right)$$

or equivalently

$$x_n^{k-1} \cdot \frac{S_n - S}{g(n)} = a_1 x_n^{k-1} + a_2 x_n^{k-2} + \cdots + a_k.$$

Applying k times the divided differences operator, that is applying the operator δ^k leads to

$$\delta^k \left(x_n^{k-1} \cdot \frac{S_n - S}{g(n)} \right) = 0\,, \qquad n = 0, 1, \ldots$$

Since δ^k is a linear operator we have the following generalized Levin's transform

$$P_k^{(n)} = \frac{\delta^k \left(x_n^{k-1} \cdot S_n / g(n) \right)}{\delta^k \left(x_n^{k-1} / g(n) \right)}.$$

We recall that the divided differences operator is defined by

$$\begin{aligned} \delta^0(u_n) &= u_n\,, & n &= 0, 1, \ldots \\ \delta^{i+1}(u_n) &= \frac{\delta^i(u_{n+1}) - \delta^i(u_n)}{x_{n+i+1} - x_n}\,, & i, n &= 0, 1, \ldots \end{aligned}$$

This is a slight generalization of the so-called generalized Richardson extrapolation process due to Sidi [397] (it reduces to a method due to Drummond [147] if $x_n = n$). In this process we take $g(n) = b_n x_n^{k-1}$ and thus we have

$$P_k^{(n)} = \frac{\delta^k (S_n/b_n)}{\delta^k (1/b_n)} = \frac{N_k^{(n)}}{D_k^{(n)}}$$

with

$$\begin{aligned} N_0^{(n)} &= \frac{S_n}{b_n}, & n = 0, 1, \ldots \\ N_{k+1}^{(n)} &= \frac{N_k^{(n+1)} - N_k^{(n)}}{x_{n+k+1} - x_n}, & k, n, = 0, 1, \ldots \end{aligned}$$

and

$$\begin{aligned} D_0^{(n)} &= \frac{1}{b_n}, & n = 0, 1, \ldots \\ D_{k+1}^{(n)} &= \frac{D_k^{(n+1)} - D_k^{(n)}}{x_{n+k+1} - x_n}, & k, n, = 0, 1, \ldots \end{aligned}$$

which are exactly the rules for computing divided differences. Other results on these transformations are given by Weniger [458], who also derived new sequence transformations based on the forms

$$S_n = S + g(n) \sum_{i=0}^{k-1} \frac{a_i}{(n+b)_i}$$

or

$$S_n = S + g(n) \sum_{i=0}^{k-1} \frac{a_i}{(-n-b)_i}$$

where $(a)_j = a \cdot (a+1) \cdot \ldots \cdot (a+j-1)$ denotes the Pochhammer symbol. Clearly, they are also particular cases of the E-algorithm.

In first case, we have

$$(n+b)_{k-1} \cdot \frac{S_n - S}{g(n)} = \sum_{i=0}^{k-1} a_i (n+i+b)_{k-i-1}.$$

Since the right hand side is a polynomial of degree $k-1$ in n, it will be annihilated by applying the operator Δ^k thus leading to the sequence

transformation

$$S_k^{(n)} = \frac{\Delta^k\left((n+b)_{k-1}\cdot S_n/g(n)\right)}{\Delta^k\left((n+b)_{k-1}/g(n)\right)}.$$

Setting $S_k^{(n)} = N_k^{(n)}/D_k^{(n)}$ we have

$$N_0^{(n)} = \frac{S_n}{g(n)}, \qquad n = 0,1,\ldots$$

$$N_{k+1}^{(n)} = N_k^{(n+1)} - \frac{(n+k+b)\cdot(n+k+b-1)}{(n+2k+b)\cdot(n+2k+b-1)}\cdot N_k^{(n)}, \quad k,n,=0,1,\ldots$$

and

$$D_0^{(n)} = \frac{1}{g(n)}, \qquad n = 0,1,\ldots$$

$$D_{k+1}^{(n)} = D_k^{(n+1)} - \frac{(n+k+b)\cdot(n+k+b-1)}{(n+2k+b)\cdot(n+2k+b-1)}\cdot D_k^{(n)}, \quad k,n,=0,1,\ldots$$

Of course, $S_k^{(n)}$ can also be computed directly by using the expression for Δ^k.

In the second case, we have

$$(-n-b)_{k-1}\cdot\frac{S_n - S}{g(n)} = \sum_{i=0}^{k-1} a_i(-n+i-b)_{k-i-1}.$$

The right hand side is again a polynomial of degree $k-1$ in n and by applying Δ^k we obtain the following sequence transformation

$$M_k^{(n)} = \frac{\Delta^k\left((-n-b)_{k-1}\cdot S_n/g(n)\right)}{\Delta^k\left((-n-b)_{k-1}/g(n)\right)} = \frac{N_k^{(n)}}{D_k^{(n)}}$$

with

$$\begin{aligned} N_0^{(n)} &= \frac{S_n}{g(n)}, & n &= 0,1,\ldots \\ N_{k+1}^{(n)} &= N_k^{(n+1)} - \frac{n-k+b+1}{n+k+b+1}\cdot N_k^{(n)}, & k,n,&=0,1,\ldots \end{aligned}$$

and

$$\begin{aligned} D_0^{(n)} &= \frac{1}{g(n)}, & n &= 0,1,\ldots \\ D_{k+1}^{(n)} &= D_k^{(n+1)} - \frac{n-k+b+1}{n+k+b+1}\cdot D_k^{(n)}, & k,n,&=0,1,\ldots \end{aligned}$$

$M_k^{(n)}$ can also be computed directly.

The subroutine LEVINT performs Levin's transforms.

To end this section, let us mention that a generalization of Levin's t-transform, called the d-transformation, was proposed by Levin and Sidi [286]. It works particularly well on sequences satisfying a recurrence relation with variable coefficients of a special form.

2.8 Overholt's process

Overholt's process, which was proposed by Overholt [351], is very close to a particular case of the E-algorithm but it does not fit exactly in its framework since, as we shall see below, it is an approximation of it.

Let us assume that the sequence (S_n) to be accelerated satisfies, for all n

$$d_{n+1} = a_1 d_n + a_2 d_n^2 + \cdots$$

where $d_n = S_n - S$ and where a_1 is a constant different from 1.

Thus, only theoretically since S is not known, the E-algorithm can be applied with $g_i(n) = d_n^i$. In that case, as we saw in section 2.2, it reduces to Richardson's process since $g_{k-1,k}^{(n)} = (-1)^{k-1} \cdot d_n \cdot \ldots \cdot d_{n+k-1}$ and we have

$$E_k^{(n)} = \frac{d_{n+k} E_{k-1}^{(n)} - d_n E_{k-1}^{(n+1)}}{d_{n+k} - d_n} = \frac{(d_{n+k}/d_n) E_{k-1}^{(n)} - E_{k-1}^{(n+1)}}{d_{n+k}/d_n - 1}.$$

Since d_n and d_{n+k} are not known we shall replace d_{n+k}/d_n by an approximation.

We have

$$\frac{d_{n+1}}{d_n} = a_1 + a_2 d_n + a_3 d_n^2 + \cdots$$

and thus

$$\frac{d_{n+k}}{d_n} = \frac{d_{n+k}}{d_{n+k-1}} \cdot \frac{d_{n+k-1}}{d_{n+k-2}} \cdot \ldots \cdot \frac{d_{n+1}}{d_n} = a_1^k + o(1).$$

Thus if an approximation $\bar{a}_1$ of a_1 can be obtained, then $\bar{a}_1^k$ will be an approximation of d_{n+k}/d_n. Let us set

$$\bar{a}_1 = \frac{\Delta S_{n+k}}{\Delta S_{n+k-1}}.$$

We have

$$\bar{a}_1 = \frac{\Delta d_{n+k}}{\Delta d_{n+k-1}} = \frac{d_{n+k}}{d_{n+k-1}} \cdot \frac{1 - d_{n+k+1}/d_{n+k}}{1 - d_{n+k}/d_{n+k-1}} = (a_1 + o(1)) \cdot \frac{a_1 - 1 + o(1)}{a_1 - 1 + o(1)}.$$

Thus, since $a_1 \neq 1$, $\bar{a}_1 = a_1 + o(1)$. Replacing, in the rule of the algorithm, d_{n+k}/d_n by $\bar{a}_1^k$ leads to Overholt's process (the $E_k^{(n)}$'s are usually denoted $V_k^{(n)}$'s)

$$V_0^{(n)} = S_n, \qquad n = 0, 1, \ldots$$

$$V_k^{(n)} = \frac{(\Delta S_{n+k})^k V_{k-1}^{(n)} - (\Delta S_{n+k-1})^k V_{k-1}^{(n+1)}}{(\Delta S_{n+k})^k - (\Delta S_{n+k-1})^k}, \quad k = 1, 2, \ldots; \; n = 0, 1, \ldots$$

The following result holds

Theorem 2.33

If Overholt's process is applied to a sequence (S_n) such that,

$$d_{n+1} = a_1 d_n + a_2 d_n^2 + \cdots$$

with $d_n = S_n - S$ and $a_1 \neq 1$, then $\forall k$ and $\forall n$

$$V_k^{(n)} = S + a_{k,k} d_n^{k+1} + a_{k,k+1} d_n^{k+2} + \cdots$$

with $a_{0,i} = a_i$ and

$$a_{k+1,k+1} = \frac{a_1^{k+1}}{1 - a_1^{k+1}} \cdot [(a_1 - 1) a_{k,k+1} - (k+1) a_2 a_{k,k}].$$

Due to this result, Overholt's process is particularly well adapted to the acceleration of fixed point iterations as we shall see in section 6.2.4.

Obviously it is a quasi-linear transformation but until now no determinantal formula for the $V_k^{(n)}$ is known although it exists as it can be proved from the theory given in section 1.6.

From the rule of the algorithm, we immediately have the

Theorem 2.34

If Overholt's process is applied to a sequence (S_n) which converges to S and if there exist $\alpha < 1 < \beta$ such that $\forall n, \Delta S_{n+1}/\Delta S_n \notin [\alpha, \beta]$ then, $\lim_{n \to \infty} V_k^{(n)} = S$.

If $\lim_{n \to \infty} (\Delta S_{n+k}/\Delta S_{n+k-1})^k = \lim_{n \to \infty} \left(V_{k-1}^{(n+1)} - S\right) \Big/ \left(V_{k-1}^{(n)} - S\right) \neq 1$ *then* $\left(V_k^{(n)}\right)$ *converges to S faster than* $\left(V_{k-1}^{(n)}\right)$.

Let us mention that a variant of Overholt's process was considered by Meinardus [318]. It consists in replacing in the rule of the algorithm ΔS_{n+k-1} by ΔS_n and ΔS_{n+k} by ΔS_{n+1}. For this modification we have the

Theorem 2.35

If $S_n = S + a_1\lambda^n + a_2\lambda^{2n} + \cdots + a_r\lambda^{rn} + O\left(|\lambda|^{(r+1)n}\right)$ *then when* n *tends to infinity*

$$V_k^{(n)} = S + O\left(|\lambda|\lambda^{(k+1)n}\right), \quad \text{for } k = 0, \ldots, r-1.$$

This is in particular the case if $S_{n+1} = F(S_n)$ with (S_n) converging to $S = F(S)$, $F \in \mathbb{C}^{r+1}[a,b]$, $\lambda = F'(S)$ and $|F'(x)| \leq \varrho < 1$, $\forall x \in [a,b]$.

The subroutine OVERHO performs Overholt's process.

2.9 Θ-type algorithms

Let us consider again the ε-algorithm. It can be written as

$$\varepsilon_{k+1}^{(n)} = \varepsilon_{k-1}^{(n+1)} + D_k^{(n)}$$

with $D_k^{(n)} = \left(\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right)^{-1}$. We have

$$\Delta\varepsilon_{k+1}^{(n)} = \Delta\varepsilon_{k-1}^{(n+1)} + \Delta D_k^{(n)}$$

where the operator Δ operates on the upper indexes. Thus we shall have

$$\lim_{n\to\infty} \Delta\varepsilon_{2k+2}^{(n)} \Big/ \Delta\varepsilon_{2k}^{(n+1)} = 0$$

(which implies, with some further assumptions given in section 1.12, that $\lim_{n\to\infty} \left(\varepsilon_{2k+2}^{(n)} - S\right) \Big/ \left(\varepsilon_{2k}^{(n+1)} - S\right) = 0$) if and only if

$$\lim_{n\to\infty} \Delta D_{2k+1}^{(n)} \Big/ \Delta\varepsilon_{2k}^{(n+1)} = -1.$$

If this condition is not satisfied then the sequence $\left(\Delta\varepsilon_{2k+2}^{(n)}\right)$ does not converge faster than the sequence $\left(\Delta\varepsilon_{2k}^{(n)}\right)$. In that case, we can introduce a parameter ω_k in the rule of the algorithm which becomes

$$\varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + \omega_k D_{2k+1}^{(n)}.$$

If ω_k is chosen such that

$$\omega_k = -\lim_{n\to\infty} \Delta\varepsilon_{2k}^{(n+1)} \Big/ \Delta D_{2k+1}^{(n)}$$

then the new sequence $\left(\Delta\varepsilon_{2k+2}^{(n)}\right)$ thus obtained will converge faster than $\left(\Delta\varepsilon_{2k}^{(n)}\right)$ and also faster the old sequence $\left(\Delta\varepsilon_{2k+2}^{(n)}\right)$ produced by the unmodified algorithm.

However, in practical situations, the computation of ω_k is difficult since it involves a limit. Thus we shall replace ω_k as given above by

$$\omega_k^{(n)} = -\Delta\varepsilon_{2k}^{(n+1)} \Big/ \Delta D_{2k+1}^{(n)}$$

and the rule of the algorithm becomes (using the new letter Θ to make the distinction)

$$\begin{aligned}
& \Theta_{-1}^{(n)} = 0\,, \quad \Theta_0^{(n)} = S_n\,, && n = 0,1,\ldots \\
\Theta_{2k+1}^{(n)} &= \Theta_{2k-1}^{(n+1)} + D_{2k}^{(n)}\,, && k,n = 0,1,\ldots \\
\Theta_{2k+2}^{(n)} &= \Theta_{2k}^{(n+1)} - \frac{\Delta\Theta_{2k}^{(n+1)}}{\Delta D_{2k+1}^{(n)}} \cdot D_{2k+1}^{(n)}\,, && k,n = 0,1,\ldots
\end{aligned}$$

with $D_k^{(n)} = \left(\Theta_k^{(n+1)} - \Theta_k^{(n)}\right)^{-1}$.

This algorithm, called the Θ-algorithm, was first proposed by Brezinski [36]. The particular rules of the Θ-algorithm are under progress, see Redivo Zaglia [373]. The numerical experiments conducted by Smith and Ford [413, 414] show that the Θ-algorithm is among the algorithms which provide almost always a good answer. Of course, such numerical experiments do not replace a theoretical proof and we shall now try to understand the reason for its success and study the sequence $\left(\Theta_2^{(n)}\right)$. It was observed numerically that the sequences $\left(\varepsilon_2^{(n)}\right)$ and $\left(\varrho_2^{(n)}\right)$, obtained respectively as the first even columns of the ε and ϱ-algorithms, almost never give a good result simultaneously. If one of them works well, the other one does not. On the contrary the results obtained with $\left(\Theta_2^{(n)}\right)$ are, most of the time, almost as good as the best result provided by $\left(\varepsilon_2^{(n)}\right)$ and $\left(\varrho_2^{(n)}\right)$. In the limit, if $S_n = S + \alpha\lambda^n$, then $\left(\varepsilon_2^{(n)}\right)$ is optimal since $\forall n, \varepsilon_2^{(n)} = S$ while $\varrho_2^{(n)} \neq S$. On the other hand, if

$S_n = S + \alpha(n+b)^{-1}$ then $\left(\varrho_2^{(n)}\right)$ is optimal since $\forall n, \varrho_2^{(n)} = S$ while $\varepsilon_2^{(n)} \neq S$. But, in both cases, $\forall n, \Theta_2^{(n)} = S$. Thus the kernel of the transformation $\Theta_2 : (S_n) \longmapsto \left(\Theta_2^{(n)}\right)$ contains the kernel of the transformations $\varepsilon_2 : (S_n) \longmapsto \left(\varepsilon_2^{(n)}\right)$ and $\varrho_2 : (S_n) \longmapsto \left(\varrho_2^{(n)}\right)$. More precisely, as proved by Cordellier [117], we have the

Theorem 2.36

A necessary and sufficient condition that $\forall n, \Theta_2^{(n)} = S$ *is that* (S_n) *has one of the following form*

1. $S_n = S + (S_0 - S)\lambda^n$ *where* $S_0 \neq S$ *and* $\lambda \neq 0$ *and* 1.
2. $S_n = S + (S_0 - S)\prod_{i=0}^{n-1}\left[1 - d(i-m)^{-1}\right]$ *where* $S_0 \neq S, d \neq 1$ *and* m *and* $m+d$ *are not integers.*
3. $S_0 = S, S_n = S + (S_1 - S)\prod_{i=0}^{n-1}\left(1 - di^{-1}\right)$ *for* $n \geq 1$ *where* $S_1 \neq S$ *and* d *is not an integer.*

Let us remark that in the first case (S_n) converges if and only if $|\lambda| < 1$ while, in the two other cases, it converges if and only if the real part of d is strictly positive. When convergent, (S_n) tends to S when n tends to infinity.

Of course it would be much interesting to know the kernels of the transformation $\Theta_{2k} : (S_n) \longmapsto \left(\Theta_{2k}^{(n)}\right)_n$ corresponding to the other columns of the Θ-algorithm. Up to now, this has not yet been possible due to the lack of a determinantal formula for the $\Theta_{2k}^{(n)}$'s and the difficulty of the direct study from the rule of the algorithm. However, from the theory given in section 1.6, it can be proved that $\Theta_{2k}^{(n)}$ can be expressed as a ratio of two determinants.

Concerning the convergence and acceleration, we have the

Theorem 2.37

If $\lim_{n\to\infty} \Theta_{2k}^{(n)} = S$ *and if there exist* α *and* β *such that* $\alpha < 1 < \beta$ *and if* $\forall n > N$, $D_{2k+1}^{(n+1)} \Big/ D_{2k+1}^{(n)} \notin [\alpha, \beta]$ *then* $\lim_{n\to\infty} \Theta_{2k+2}^{(n)} = S$.

Moreover if there exists $a_k \neq 0$ *such that* $\lim_{n\to\infty} \left(\Theta_{2k}^{(n+1)} - S\right) \Big/ D_{2k+1}^{(n)} = \lim_{n\to\infty} \Delta\Theta_{2k}^{(n+1)} \Big/ \Delta D_{2k+1}^{(n)} = a_k$ *then* $\lim_{n\to\infty} \left(\Theta_{2k+2}^{(n)} - S\right) \Big/ \left(\Theta_{2k}^{(n+1)} - S\right) = 0$.

The condition for acceleration clearly shows what is gained by using the Θ-algorithm instead of the ε-algorithm. For the ε-algorithm, $\left(\varepsilon_{2k+2}^{(n)}\right)$ converges faster than $\left(\varepsilon_{2k}^{(n+1)}\right)$ if and only if $\lim_{n\to\infty}\left(\varepsilon_{2k}^{(n+1)}-S\right)/D_{2k+1}^{(n)}=-1$. For the Θ-algorithm, it is sufficient that this ratio has a limit a_k different from zero and that $\left(\Delta\Theta_{2k}^{(n+1)}/\Delta D_{2k+1}^{(n)}\right)_n$ also tends to the same limit, which is true under mild additional conditions given, for example, in Brezinski [91] (see section 1.12).

Another way of formulating the acceleration condition of theorem 2.37 is to assume that there exists $b_k \neq 1$ such that $\lim_{n\to\infty}\left(\Theta_{2k}^{(n+1)}-S\right)/\left(\Theta_{2k}^{(n)}-S\right)=\lim_{n\to\infty} D_{2k+1}^{(n+1)}/D_{2k+1}^{(n)}=b_k$. Under this condition $\left(\Theta_{2k+2}^{(n)}\right)_n$ converges to S faster than $\left(\Theta_{2k}^{(n+1)}\right)_n$.

The subroutine THETA performs the Θ-algorithm.

The procedure used to obtain the Θ-algorithm from the ε-algorithm can also be applied to other algorithms in order to improve their acceleration properties. This procedure, called the procedure Θ, is fully developed in Brezinski [65]. Let us consider a sequence transformation $T:(S_n)\longmapsto(T_n)$ written under the form

$$T_n = S_n + D_n\,, \qquad n=0,1,\ldots$$

The Θ-type algorithm associated with T consists in the transformation $\Theta(T):(S_n)\longmapsto(\Theta(T_n)=\Theta_n)_n$ given by

$$\Theta_n = S_n - \frac{\Delta S_n}{\Delta D_n}\cdot D_n\,, \qquad n=0,1,\ldots$$

Of course, it must be assumed that $\Delta D_n \neq 0$. If this is not the case we shall only apply the procedure Θ to the subsequence of (T_n) such that this condition holds.

Obviously also, if T is quasi-linear then so is $\Theta(T)$.

The kernel of the transformation T is the set of sequences such that $\forall n, S_n + D_n = S$. For the transformation $\Theta(T)$ we have the following result which shows the improvement obtained by the procedure Θ

Theorem 2.38

Let us assume that $\forall n, D_n \neq 0$ *and* $\Delta D_n \neq 0$. *A necessary and sufficient condition that* $\forall n, \Theta_n = S$ *is that* $\exists c \in \mathbb{C}$ *such that,* $\forall n$

$$S_n = S + cD_n.$$

Of course if the transformation T is written as

$$T_n = S_{n+p} + D'_n\,, \qquad n = 0, 1, \ldots$$

where p is a fixed non-negative integer, we can apply the procedure Θ and obtain

$$\Theta'_n = S_{n+p} - \frac{\Delta S_{n+p}}{\Delta D'_n} \cdot D'_n\,, \qquad n = 0, 1, \ldots$$

For the transformation T, obviously (T_n) tends to S if and only if (D_n) tends to zero and (T_n) converges to S faster than (S_n) if and only if $\lim_{n\to\infty}(S_n - S)/D_n = -1$. For $\Theta(T)$, we have the

Theorem 2.39

(Θ_n) tends to S if and only if $\lim_{n\to\infty} \frac{\Delta S_n}{\Delta D_n} \cdot D_n = 0$, a condition satisfied if $\exists \alpha < 1 < \beta$ such that $\forall n > N, D_{n+1}/D_n \notin [\alpha, \beta]$.

(Θ_n) converges to S faster than (S_n) if and only if $\lim_{n\to\infty} \frac{\Delta S_n}{\Delta D_n} \cdot \frac{D_n}{S_n - S} = 1$. This condition is satisfied if $\exists b \neq 1$ such that $\lim_{n\to\infty}(S_{n+1} - S)/(S_n - S) = \lim_{n\to\infty} D_{n+1}/D_n = b$. It is also satisfied if $\exists a \neq 0$ such that $\lim_{n\to\infty}(S_n - S)/D_n = \lim_{n\to\infty} \Delta S_n/\Delta D_n = a$.

The advantage of the second acceleration condition over the first one is that it can also apply to logarithmic sequences while the first condition cannot, see Brezinski [77].

Let us mention that if we consider the sequence transformation T : $(S_n) \longmapsto (T_n = S_{n+1})$ then we can write

$$T_n = S_n + \Delta S_n.$$

Applying the procedure Θ yields

$$\Theta_n = S_n - \frac{\Delta S_n}{\Delta^2 S_n} \cdot \Delta S_n$$

which is Aitken's Δ^2 process.

Let us now apply the procedure Θ to the E-algorithm. Since this algorithm has a main rule and an auxiliary rule we can obtain various algorithms by applying the procedure Θ either to the main rule, or to the auxiliary rule or to both. Since, also, both rules can be written as

above either with $p = 0$ or $p = 1$ then many possibilities can be studied. We shall only consider here one of these possibilities in order to show the improvement obtained over the E-algorithm.

Let us apply the procedure Θ to the main rule of the E-algorithm in which $p = 0$; we obtain the following algorithm

$$T_0^{(n)} = S_n\,, \quad g_{0,i}^{(n)} = g_i(n)\,, \qquad n = 0,1,\ldots;\ i \geq 1$$

$$T_k^{(n)} = T_{k-1}^{(n)} - \frac{\Delta T_{k-1}^{(n)}}{\Delta D_k^{(n)}} \cdot D_k^{(n)}\,, \qquad k = 1,2,\ldots;\ n = 0,1,\ldots$$

with

$$D_k^{(n)} = -\frac{\Delta T_{k-1}^{(n)}}{\Delta g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}$$

and

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{\Delta g_{k-1,i}^{(n)}}{\Delta g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}\,, \qquad i > k$$

and where Δ operates on the upper indexes.

Of course theorem 2.39 applies to this algorithm. But, in this case we have

$$\frac{D_k^{(n+1)}}{D_k^{(n)}} = \frac{\Delta T_{k-1}^{(n+1)}}{\Delta T_{k-1}^{(n)}} \cdot \frac{g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} - 1}{g_{k-1,k}^{(n+2)}/g_{k-1,k}^{(n+1)} - 1}$$

and we have the more precise following result in the non-logarithmic case

Theorem 2.40

If $\lim_{n\to\infty} T_{k-1}^{(n)} = S$, *if* $\exists \alpha < 1 < \beta$ *such that* $\forall n > N, \Delta T_{k-1}^{(n+1)}/\Delta T_{k-1}^{(n)} \notin [\alpha,\beta]$ *and if* $\exists b_k \neq 1$, $\lim_{n\to\infty} g_{k-1,k}^{(n+1)}/g_{k-1,k}^{(n)} = b_k$ *then,* $\lim_{n\to\infty} T_k^{(n)} = S$.

Moreover if $b_k \neq 0$ *and if* $\exists c_k \neq 0$ *and 1 such that* $\lim_{n\to\infty} \left(T_{k-1}^{(n+1)} - S\right)/\left(T_{k-1}^{(n)} - S\right) = c_k$ *then* $\left(T_k^{(n)}\right)$ *converges to* S *faster than* $\left(T_{k-1}^{(n)}\right)$.

If this result is compared with theorem 2.9, we see that the acceleration condition $b_k = c_k$ for the E-algorithm is now no more needed. We only need that both limits exist but they can be different.

Let us apply the E-algorithm with $g_i(n) = \Delta S_{n+i-1}$ (that is Shanks' transformation) and the preceding algorithm to the sequence

$$S_n = (0.95)^{n+1}/(n+1), \qquad \text{for } n = 0, 1, \ldots.$$

We obtain the following results (the first number corresponds to the E-algorithm and the second one to the T-algorithm)

n	(S_n)	$\left(E_1^{(n)}, T_1^{(n)}\right)$	$\left(E_2^{(n)}, T_2^{(n)}\right)$	$\left(E_3^{(n)}, T_3^{(n)}\right)$
0	0.95000000			
1	0.45125000			
2	0.28579167	0.20365202		
		0.05877243		
3	0.20362656	0.12257430		
		0.01932949		
4	0.15475619	0.08302371	0.07015522	
		0.00412200	−0.01399516	
5	0.12251532	0.06000750	0.04522404	
		−0.00242632	−0.01086415	
6	0.09976247	0.04519982	0.03113841	0.02778396
		−0.00523243	−0.00854903	0.00473557

For $(S_n = (-0.95)^{n+1}/(n+1))$ we have

$$\begin{array}{llllll} S_3 = 0.20362656 & E_1^{(1)} = 0.00832445 & T_1^{(0)} = -0.00471336 \\ S_6 = -0.09976247 & E_2^{(2)} = -0.00006298 & T_2^{(1)} = -0.00000083 \end{array}$$

For the logarithmic sequence $(S_n = (n+1)/(n+2))$ we obtain

$$\begin{array}{llllll} S_3 = 0.80000000 & E_1^{(1)} = 0.87500000 & T_1^{(0)} = 0.94444444 \\ S_6 = 0.87500000 & E_2^{(2)} = 0.94444444 & T_2^{(1)} = 0.99900000 \\ S_9 = 0.90909091 & E_3^{(3)} = 0.96875000 & T_3^{(2)} = 1.00000256 \\ S_{12} = 0.92857143 & E_4^{(4)} = 0.98000000 & T_4^{(3)} = 0.99999998 \end{array}$$

Let us mention that the rules of the E-algorithm can be written as

$$E_k^{(n)} = E_{k-1}^{(n)} - \frac{\Delta E_{k-1}^{(n)}}{\Delta g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{\Delta g_{k-1,i}^{(n)}}{\Delta g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}, \qquad i > k.$$

Thus they can be considered as obtained by the application of the procedure Θ to (with the same letters)

$$\begin{aligned} E_k^{(n)} &= E_{k-1}^{(n)} + g_{k-1,k}^{(n)} \\ g_{k,i}^{(n)} &= g_{k-1,i}^{(n)} + g_{k-1,k}^{(n)}, \qquad i > k. \end{aligned}$$

This procedure can be used to obtain non-linear sequence transformations from linear ones as done by Weniger [458].

2.10 The iterated Δ^2 process

Instead of using, for example, the ε-algorithm, Aitken's Δ^2 process can be applied iteratively to the sequence (S_n). That is, it is first applied to (S_n), then again to the sequence thus obtained and so on.

We have the following algorithm

$$x_0^{(n)} = S_n, \qquad n = 0, 1, \ldots$$

$$x_{k+1}^{(n)} = x_k^{(n)} - \frac{\left(x_k^{(n+1)} - x_k^{(n)}\right)^2}{x_k^{(n+2)} - 2x_k^{(n+1)} + x_k^{(n)}}, \qquad k, n = 0, 1, \ldots$$

The motivation for studying the iterated Δ^2 process instead of studying the iteration of another algorithm can be considered as almost purely subjective: Aitken's process is very simple and efficient on linearly convergent sequences. As proved by Delahaye [137] it is optimal, in three different meanings, for such sequences (see section 1.11).

Let us consider the following sequence

$$\begin{aligned} S_0 &= 1 \\ S_{n+1} &= 1 + \frac{a}{S_n}, \qquad n = 0, 1, \ldots \end{aligned}$$

(S_n) is the sequence of the successive convergents of the continued fraction

$$S = 1 + \frac{a}{\left\lceil 1 \right\rfloor} + \frac{a}{\left\lceil 1 \right\rfloor} + \frac{a}{\left\lceil 1 \right\rfloor} + \cdots$$

If $a \neq -1/4 + c$ where c is a real nonpositive number, (S_n) converges to the zero of greatest modulus of $x^2 - x - a = 0$. If the iterated Δ^2 process is applied to this sequence (S_n) then very interesting results are obtained. First of all, it produces better results than the ε-algorithm. But it also shows that the linearly converging sequence (S_n) (when $a \neq -1/4+c$ with $c < 0$) can be transformed into the sequence $\left(x_k^{(n)}\right)_k$ which converges super-quadratically that is $\lim_{k\to\infty} \left(x_{k+1}^{(n)} - S\right) / \left(x_k^{(n)} - S\right)^2 = 0$. It must be noticed that $\left(\varepsilon_{2k}^{(n)}\right)_k$ converges only super-linearly, that is $\lim_{k\to\infty} \left(\varepsilon_{2k+1}^{(n)} - S\right) / \left(\varepsilon_{2k}^{(n)} - S\right) = 0$.

When $a = -1/4$, (S_n) is a logarithmic sequence but $\left(x_k^{(n)}\right)_k$ converges linearly while $\left(\varepsilon_{2k}^{(n)}\right)_k$ is still logarithmic. Thus $\left(x_k^{(n)}\right)_k$ can be accelerated again by Aitken's Δ^2 process.

This is the only known example of the transformation of a linear sequence into a super-quadratic sequence and of the transformation of a logarithmic sequence into a linear one. The details of these results can be found in Brezinski and Lembarki [94] and in section 6.1.4.

Other versions of the iterated Δ^2 process and the iterated Θ_2 transformation are studied by Weniger [458].

As showed by Wynn [483] on numerical examples the repeated application of an extrapolation algorithm can lead to a considerable improvement of the results. Other examples can be found in Brezinski [56], in Weniger [458] and in Bhowmick, Bhattacharya and Roy [29] who also showed that repeated applications can improve the numerical stability of an algorithm.

Let us consider the sequence of the partial sums of the series

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

for $x = 1$.

If we apply the ε-algorithm again to the sequence $\varepsilon_0^{(0)}$, $\varepsilon_0^{(1)}$, $\varepsilon_2^{(0)}$, $\varepsilon_2^{(1)}$, $\varepsilon_4^{(0)}$, $\varepsilon_4^{(1)}$, ... several times we obtain (number of exact digits)

n	(S_n)	1st appl.	2nd appl.	3rd appl.
0	0.35			
1	0.55			
2	0.69	2.00	1.14	0.93
3	0.80	2.41	2.49	1.50
4	0.89	3.57	4.09	2.53
5	0.96	4.08	4.63	4.34
6	1.02	5.12	5.35	6.35
7	1.07	5.68	6.70	6.88
8	1.12	6.66	8.40	7.79
9	1.16	7.26	8.88	9.22
10	1.20	8.20	9.57	10.38
11	1.24	8.82	10.98	11.69
12	1.27	9.73	12.75	12.72
13	1.30	10.38	13.15	15.49
14	1.33	11.27	13.79	15.80

Such a procedure is called the repeated application of the ε-algorithm.

Applying iteratively the ε-algorithm to the diagonal sequence $\varepsilon_0^{(0)}$, $\varepsilon_2^{(0)}$, $\varepsilon_4^{(0)}$, ... we obtain for the same example

n	(S_n)	1st appl.	2nd appl.	3rd appl.
0	0.35			
1	0.55	2.00	4.30	8.42
2	0.69	3.57	8.74	15.80
3	0.80	5.12	13.09	
4	0.89	6.66	15.80	
5	0.96	8.20		
6	1.02	9.73		
7	1.07	11.27		

The obtention of these results needs the same first 15 terms of the series.

This procedure is called the associated application of the ε-algorithm.

Although some theoretical results were obtained by Kateb [257], the repeated application of extrapolation methods remains an almost unexplorated question. It is also possible to think about the composition (in the sense of composing applications) of several sequence transformation

that is

$$(S_n) \overset{T_1}{\longmapsto} \left(T_1^{(n)}\right) \overset{T_2}{\longmapsto} \left(T_2^{(n)}\right) \longmapsto \cdots$$

where $T_1, T_2, \ldots$ are any extrapolation method. This question has not yet been studied.

The subroutine IDELTA performs the iterated Δ^2 process.

2.11 Miscellaneous algorithms

In this section we shall present some algorithms which are not, for the moment, much developed but which however present some interest either because of their theory or for their applications. The selection of these algorithms was mostly a matter of taste or, more simply, knowledge and we do not pretend to be comprehensive.

Since Aitken's Δ^2 process is one of the most popular and powerful sequence transformation and since many more sophisticated algorithms have it as their first step, we shall begin by its various generalizations and follow the chronological order. The first generalization of Aitken's process is the T_{+m} transformation due to Gray and Clark [198]. Let m be a strictly positive integer. We consider the sequence

$$T_{+m}^{(n)} = S_n - \frac{\Delta S_n}{\Delta S_{n+m} - \Delta S_n} \cdot (S_{n+m} - S_n) \ , \qquad n = 0, 1, \ldots$$

For $m = 1$, Aitken's Δ^2 process is recovered. Acceleration results were given by the previous authors and by Streit [420]. When using this transformation the main point is to find the best possible value of the integer m. Of course it is easy to see that $\forall n, T_{+m}^{(n)} = S$ if and only if $\forall n, S_{n+m} - S = a(S_n - S)$ with $a \neq 1$ or, in other terms if and only if $\forall n, S_{n+m} = aS_n + (1-a)S$. If $\forall n, S_{n+m} \simeq aS_n + (1-a)S$ then the T_{+m} transformation will give good approximations of S. For knowing whether or not the sequence (S_n) has this form, it is possible to compute the correlation coefficient of the sequences (S_n) and (S_{n+m}) for various values of m and to choose the value of m which gives the coefficient closest to 1 in absolute value. This technique was proposed by Jones [249]. This correlation coefficient can be computed from the

first $n+m$ terms of the sequence by

$$\varrho = \frac{n\sum_{i=0}^{n-1} S_i S_{i+m} - \sum_{i=0}^{n-1} S_i \sum_{i=0}^{n-1} S_{i+m}}{\left[n\sum_{i=0}^{n-1} S_i^2 - \left(\sum_{i=0}^{n-1} S_i\right)^2\right]^{1/2} \cdot \left[n\sum_{i=0}^{n-1} S_{i+m}^2 - \left(\sum_{i=0}^{n-1} S_{i+m}\right)^2\right]^{1/2}}.$$

Let us mention that Jones [248] already used the same idea to determine the index for starting the application of Aitken's process and that this idea is also the basis for the selection procedure proposed in section 3.6. Let us take the same example as Jones [249] since the results found are not exactly the same as his. He considered

$$S = \sum_{i=1}^{\infty} \frac{\cos ix}{i^2} = \frac{\pi^2}{6} - \frac{\pi x}{2} + \frac{x^2}{4} - \cdots, \qquad 0 \le x \le 2\pi.$$

For $x = 1.05$, we have $S = 0.2712229\ldots$. The values of ϱ were computed from $S_0, \ldots, S_{5+m}$ by the above formula with $n = 5$. We obtain

$$\begin{aligned} m &= 1 \quad 0.819 \\ m &= 2 \quad -0.462 \\ m &= 3 \quad -0.984 \\ m &= 4 \quad -0.650 \end{aligned}$$

n	S_n	$T_{+1}^{(n)}$	$T_{+2}^{(n)}$	$T_{+3}^{(n)}$	$T_{+4}^{(n)}$
10	0.266864	0.278414	0.273838	0.271125	0.268887
20	0.270187	0.269757	0.270232	0.271243	0.296420
30	0.272224	0.273121	0.268132	0.271230	0.271649
40	0.270972	0.271876	0.271471	0.271220	0.270952
50	0.271000	0.270948	0.271054	0.271226	0.269844

Thus the values obtained agree with those of the correlation coefficient which predicts that T_{+3} must be the best transform.

A generalization of the T_{+m} transformation to double sequences was proposed by Streit [419].

A further generalization was proposed by Delahaye [135] for accelerating the convergence of linear periodic sequences that is such that

$$\lim_{n\to\infty} \frac{S_{nk+i+1} - S}{S_{nk+i} - S} = \beta_i, \qquad \text{for } i = 0, \ldots, k-1$$

with $\beta_i \neq 0$ and 1 and $|\beta_0 \cdot \ldots \cdot \beta_{k-1}| < 1$ (a condition insuring the convergence of (S_n)). k is called the period and a numerical method for determining k was given by Delahaye [133]. When k is known we consider the transformation defined by

$$T_n = S_{n+2k} - \frac{(S_{n+k} - S_n)^2}{S_{n+2k} - 2S_{n+k} + S_n}, \qquad n = 0, 1, \ldots$$

which accelerates the convergence of linear periodic sequences of period k. Its kernel is the set of sequences such that

$$S_n = S + a \prod_{i=0}^{n} \beta_i, \qquad n = 0, 1, \ldots$$

with $a \neq 0$ and $\beta_{i+k} = \beta_i$.

If k is not known a sequence $(k(n))$ converging to k can be built (see Delahaye [133]) and we can consider the transformation given by

$$T_n = S_{n+2k(n)} - \frac{\left(S_{n+k(n)} - S_n\right)^2}{S_{n+2k(n)} - 2S_{n+k(n)} + S_n}, \qquad n = 0, 1, \ldots$$

which still accelerates the convergence of (S_n).

A further generalization consists in taking

$$T_n = S_n - \frac{(S_{n+p+r} - S_n) \cdot (S_{n+p+q} - S_{n+q})}{S_{n+2p+r+q} - S_{n+p+r+q} - S_{n+p+q} + S_{n+q}}, \qquad n = 0, 1, \ldots$$

For all p, q and r such a transformation accelerates linearly converging sequences. As showed by Benchiboun [24] such processes can be useful for accelerating sequences which are neither linear nor logarithmic.

Another generalization of the Δ^2 process was given by Iguchi [234]. It is

$$T_k^{(n)} = S_{n+2} + \omega_k^{(n)} (S_{n+2} - S_n), \qquad n = 0, 1, \ldots$$

with $\omega_k^{(n)} = \sum_{i=1}^{k} a_n^{2i}$ and $a_n = \Delta S_{n+1} / \Delta S_n$.

We have

$$\lim_{k \to \infty} \omega_k^{(n)} = a_n^2 \Big/ \left(1 - a_n^2\right) = \omega_\infty^{(n)}$$

and it follows that

$$T_{\infty}^{(n)} = S_{n+2} + \omega_{\infty}^{(n)} (S_{n+2} - S_n) = \varepsilon_2^{(n)}.$$

This algorithm was proved to be useful for accelerating the convergence of the power method for computing the dominant eigenvalue of a matrix, see Iguchi [235], but it deserves further theoretical studies.

Let us now study some modifications of Aitken's Δ^2 process for accelerating logarithmically convergent series with terms having an asymptotic expansion of a known form. The first of these processes is due to Drummond [148] who considered a series $S = a_1 + a_2 + \cdots$ with

$$a_n \sim A n^{\theta - 1}$$

where θ is a known negative constant. He proposed to replace the partial sums $S_n = a_1 + a_2 + \cdots + a_n$ by $S_n^* = a_1^* + a_2^* + \cdots + a_n^*$ where

$$a_n^* = a_n + \frac{1-\theta}{\theta} \cdot \left[\frac{a_n a_{n+1}}{\Delta a_n} - \frac{a_{n-1} a_n}{\Delta a_{n-1}} \right].$$

Thus

$$S_n^* = S_n + \frac{1-\theta}{\theta} \cdot \frac{a_n a_{n+1}}{\Delta a_n} = S_n - \frac{\theta - 1}{\theta} \cdot \frac{\Delta S_{n-1} \Delta S_n}{\Delta^2 S_{n-1}}$$

which shows that a factor $(\theta - 1)/\theta$ has been introduced in Aitken's formula. It can be proved that

$$a_n^* \sim -\frac{A(2-\theta)}{12} \cdot (n-1)^{\theta - 3}$$

and thus the process can be iterated after replacing θ by $\theta - 2$, which leads to the recursive algorithm obtained by Bjørstad, Dahlquist and Grosse [30]

$$S_0^{(n)} = S_n \,, \qquad n = 0, 1, \ldots$$

$$S_{k+1}^{(n)} = S_k^{(n)} - \frac{2k + 1 - \theta}{2k - \theta} \cdot \frac{\Delta S_k^{(n)} \Delta S_k^{(n-1)}}{\Delta^2 S_k^{(n-1)}} \,, \qquad n, k = 0, 1, \ldots$$

who proved that this algorithm accelerates the convergence of all the sequences of the set L of sequences of the form

$$S_n \sim S + n^{\theta} \cdot \left(c_0 + c_1 n^{-1} + c_2 n^{-2} + \cdots \right) \qquad (n \to \infty)$$

with $\theta < 0$ and $c_0 \neq 0$, more precisely that

$$S_k^{(n)} - S = O\left(n^{\theta-2k}\right) \qquad (n \to \infty).$$

Instead of using this modification of Aitken's process it is possible to use the following modification of the ϱ-algorithm (see section 2.5) due to Drummond [148] and Osada [347]

$$\begin{aligned} \varrho_{-1}^{(n)} &= 0\,, \quad \varrho_0^{(n)} = S_n\,, & n &= 0,1,\ldots;\ i \geq 1 \\ \varrho_{k+1}^{(n)} &= \varrho_{k-1}^{(n+1)} + \frac{k-\theta}{\varrho_k^{(n+1)} - \varrho_k^{(n)}}\,, & k, n &= 0,1,\ldots \end{aligned}$$

As proved by Osada [347], for sequences of $\mathbb{L}$

$$\varrho_{2k}^{(n)} - S = O\left((n+k)^{\theta-2k}\right) \qquad (n \to \infty).$$

The main drawback of these processes is that they need the knowledge of θ. Drummond [148] proposed to estimate θ by

$$\theta_n = 1 + \frac{1}{\Delta\left(\Delta S_n / \Delta^2 S_{n-1}\right)}\,, \qquad n = 0,1,\ldots$$

Bjørstad, Dahlquist and Grosse [30] proved that for sequences of $\mathbb{L}$

$$\theta_n \sim \theta + n^{-2} \cdot \left(t_0 + t_1 n^{-1} + t_2 n^{-2} + \cdots\right) \qquad (n \to \infty)$$

with $t_0 \neq 0$.

Thus the sequence (θ_n) can be accelerated by the previous modification of the ϱ-algorithm where θ is replaced by -2, or by the modification of Aitken's process or by the E-algorithm. Numerical results and another process called the automatic generalized ϱ-algorithm can be found in Osada [347].

Processes for accelerating sequences (depending on a parameter x) of the form

$$S_n(x) = S(x) + x^n \cdot \left[a_0(x)(n+\gamma)^{-k} + a_1(x)(n+\gamma)^{-k-1} + \cdots\right]$$

are given by Smith [412].

Various generalizations of Aitken's process were proposed by Kowalewski [267] for accelerating the convergence of subsets of monotone logarithmic sequences. We set

$$\lambda_n = 1 - (S_{n+1} - S)/(S_n - S) .$$

If (S_n) is logarithmic, then (λ_n) converges to zero. If $\forall n, \lambda_n > 0$ and if there exists $\lambda > 1$ such that

$$\lim_{n\to\infty} (n+1) \cdot (\Delta S_n/\Delta S_{n+1} - 1) = \lambda$$

then the sequence

$$T_n = S_n - \frac{\Delta S_n}{b_n \cdot \Delta S_{n+1}/\Delta S_n - 1} , \qquad n = 0, 1, \ldots$$

with $b_n = \dfrac{n+p+1}{n+p}$ converges faster than $(S_n), \forall p$.

If there exists $\varrho \in]0, 1]$ such that

$$\lim_{n\to\infty} \lambda_n (1 - \Delta S_{n+1}/\Delta S_n)^{-1} = \varrho$$

then the sequence

$$T_n = S_n - \frac{\Delta S_n}{(\Delta S_{n+1}/\Delta S_n)^{\varrho} - 1} , \qquad n = 0, 1, \ldots$$

converges faster than (S_n). Another possibility is to consider

$$T_n = S_n - \frac{(\Delta S_n)^2}{\varrho \Delta^2 S_n} , \qquad n = 0, 1, \ldots$$

The following modification of the ε-algorithm was introduced by Vanden Broeck and Schwartz [447]

$$\begin{aligned} & \varepsilon_{-1}^{(n)} = 0 , \quad \varepsilon_0^{(n)} = S_n \\ \varepsilon_{2k+1}^{(n)} &= \alpha \varepsilon_{2k-1}^{(n+1)} + \frac{1}{\Delta \varepsilon_{2k}^{(n)}} \\ \varepsilon_{2k+2}^{(n)} &= \varepsilon_{2k}^{(n+1)} + \frac{1}{\Delta \varepsilon_{2k+1}^{(n)}} . \end{aligned}$$

When $\alpha = 1$ we recover the ε-algorithm of Wynn while for $\alpha = 0$ we obtain the iterated Δ^2 process (see section 2.10). The following results on this algorithm were proved by Barber and Hamer [13].

Theorem 2.41

Let F be the generating function of (S_n) that is

$$F(x) = \sum_{n=0}^{\infty} S_n x^n.$$

If $F(x) = S(1-x)^{-1}\left[1 + A(1-x)^{\lambda}\right]$ and if we take $\alpha = -1$ in the modified ε-algorithm, then $\forall n, \varepsilon_4^{(n)} = S$.

Theorem 2.42

Let (S_n) be a sequence converging to S such that

$$S_n - S = an^{-\lambda} + o\left(n^{-\lambda}\right) \qquad (n \to \infty)$$

with $\lambda > 0$. If the modified ε-algorithm is applied to (S_n) with $\alpha = -1$ then

$$\varepsilon_4^{(n)} - S = o\left(n^{-\lambda}\right) \qquad (n \to \infty).$$

Moreover if

$$S_n - S = an^{-\lambda}\left(1 + bn^{-\gamma} + o\left(n^{-\lambda}\right)\right) \qquad (n \to \infty)$$

then

$$\varepsilon_4^{(n)} - S = an^{-\lambda}\varrho_n \qquad (n \to \infty)$$

with $\varrho_n = o(n^{-1})$ if $\gamma > 1$ and $\varrho_n = O\left(n^{-\gamma}\right)$ if $\gamma < 1$.

The modified ε-algorithm with $\alpha = -1$ can be generalized by replacing in its rule α by $\alpha_n = -\left(1 - (-1)^n\right)/2$. An application with an α depending on k was given by Yue-Kuen Kwok and Barthez [487].

Generalizations of Aitken's process were also given by Prévost [362, 366]. They are

$$\Delta_k^{(n)} = \sum_{i=0}^{k} \binom{k}{i} \cdot \frac{(\Delta S_{n+k-2})^i \cdot (\Delta S_{n+k-1})^{k-i}}{(\Delta^2 S_{n+k-2})^k} \cdot S_{n+i}.$$

Aitken's Δ^2 process is recovered for $k = 2$. For all k, the kernel of this transformation contains Aitken's, the convergence of linear sequences is accelerated but no other theoretical result is known for these transformations.

Now we set

$$\ln^{(0)}(x) = x, \qquad L^{(0)}(x) = x,$$
$$\ln^{(k+1)}(x) = \ln\left(\ln^{(k)}(x)\right), \quad L^{(k+1)}(x) = L^{(k)}\left(x \ln^{(k)}(x)\right), \quad k = 0, 1, \ldots$$

Osada [346] defined the transformations $T^{(k)}$ by

$$T_n^{(k)} = S_{n-1} + \frac{L^{(k)}\left((n-1)\Delta S_{n-1}\Delta S_{n-2}\right)}{L^{(k)}\left((n-2)\Delta S_{n-2}\right) - L^{(k)}\left((n-1)\Delta S_{n-1}\right)}, \quad k, n = 0, 1, \ldots$$

The case $k = 0$ was considered by Kowalewski [267]. These transformations are able to accelerate the convergence of some subsets of logarithmic sequences as we shall see below.

As explained in section 1.11, Aitken's Δ^2 process is a rational transformation. Other rational transformations were studied by Osada [349] who considered the transformations defined by

$$T_n = S_{n-1} - \frac{\Delta S_{n-2} \cdot h_n\left(S_{n-k}, \ldots, S_n\right)}{h_n\left(S_{n-k}, \ldots, S_n\right) - h_{n-1}\left(S_{n-k-1}, \ldots, S_{n-1}\right)}, \quad n = 0, 1, \ldots$$

where h_n is a rational function of its $k+1$ variables. We assume, as before, this transformation to be defined which means that its denominator does not vanish. We set

$$f_n\left(x_1, \ldots, x_{k+2}\right) = h_n\left(x_1, \ldots, x_{k+1}\right) / \left(x_k - x_{k+2}\right).$$

The kernel of this transformation, which clearly generalizes Aitken's Δ^2 process, is the set of sequences such that $\forall n \geq k$

$$f_n\left(S_{n-k}, \ldots, S_n, S\right) = \alpha$$

where α is a nonzero real number.

We set $\lambda_n = 1 - (S_{n+1} - S)/(S_n - S)$. Osada [349] proved the following fundamental result

Theorem 2.43

Suppose that there exists a nonzero real number α and a sequence (e_n) tending to zero such that $\forall n \geq k$

$$f_n\left(S_{n-k}, \ldots, S_n, S\right) = \alpha + e_{n-1},$$

suppose that there exist m and M, $0 \notin [m, M]$ such that $\forall n \geq N \geq k$, $m < (S_{n+1} - S)/(S_n - S) < M$, then a necessary and sufficient condition that (T_n) converges to S faster than (S_n) is that

$$\lim_{n \to \infty} \Delta e_n / \lambda_n = 0.$$

This condition is automatically satisfied if $(S_{n+1} - S)/(S_n - S)$ does not tend to 1 when n tends to infinity.

From this theorem, Osada [349] was able to prove that its previous transformation, which is recovered if $h_n(x_1, x_2) = L^{(k)}(n-1)(x_2 - x_1)$, accelerates the convergence of some subsets of LOGSF.

We consider the following sets

$$A^{(k)} = \left\{(S_n) \in \text{LOGSF} \,\middle|\, \exists \alpha > 0 \text{ such that } \lim_{n\to\infty} L^{(k)} n\lambda_n = \alpha\right\}$$
$$B^{(k)} = \left\{(S_n) \in A^{(k)} \,\middle|\, \lim_{n\to\infty} L^{(k)} n\Delta e_n = 0\right\}.$$

We have the

Theorem 2.44

Let $(S_n) \in A^{(k)}$ for some $k \geq 0$. A necessary and sufficient condition that

$$\lim_{n\to\infty} \left(T_n^{(k)} - S\right) \Big/ (S_n - S) = 0$$

is that $(S_n) \in B^{(k)}$.

We have

$$\varepsilon_2^{(n)} = \Delta\left(S_n/\Delta S_n\right)/\,\Delta(1/\Delta S_n)$$
$$\Theta_2^{(n)} = \Delta^2\left(S_n/\Delta S_n\right)\Big/\,\Delta^2(1/\Delta S_n).$$

This remark gave Drummond [147] the idea of generalizing as

$$T_k^{(n)} = \Delta^k\left(S_n/\Delta S_n\right)\Big/\,\Delta^k(1/\Delta S_n).$$

He gave some numerical results showing the interest of such transformations but no theoretical results. Bhowmick, Bhattacharya and Roy [29] found that for $k > 2$ no improvement occurs for series with positive terms. Let us mention that the Θ_2-transformation $(S_n) \longmapsto \left(\Theta_2^{(n)}\right)$ is equivalent to the W-transformation of Lubkin [301] and that it was proved to be based on an approximation of the main term of the Euler-Maclaurin's formula by Sato [386].

Finally another generalization of Aitken's process was given by Meinardus [318] and found to be quite effective for sequences generated by fixed point iterations, namely

$$y_0^{(n)} = S_n$$
$$y_{k+1}^{(n)} = y_k^{(n)} + \frac{\Delta y_k^{(n)}}{1 - a_k^{(n)}}, \qquad \text{with } a_k^{(n)} = (\Delta S_{n+1}/\Delta S_n)^k .$$

For $k = 0$, Aitken's formula is recovered. This transformation can be considered as a variant of Overholt's process (see section 2.8).

Let us consider the partial sums of a power series

$$S_n = \sum_{k=1}^{n} c_k (-x)^{k-1} , \qquad n = 1, 2, \ldots$$

For accelerating (S_n), Longman [297] considered the sequence $(\bar{S}_n)$ given by

$$\bar{S}_n = \sum_{k=1}^{2n} c_k \alpha_k (-x)^{k-1} , \qquad n = 1, 2, \ldots$$

with

$$\alpha_k = \begin{cases} 1 & 1 \le k \le n \\ (1+x)^{-n} \displaystyle\sum_{i=0}^{2n-k} \binom{n}{i} x^i & n+1 \le k \le 2n. \end{cases}$$

It was proved by Longman [298] that it holds

$$\bar{S}_n = \left(\frac{x}{1+x}\right)^n \sum_{i=0}^{n} \binom{n}{i} x^{-i} S_{n+i}.$$

The sequence $(\bar{S}_n)$ can be recursively computed by an algorithm due to Lepora and Gabutti [282] which is as follows

$$W_{n,0} = S_n , \qquad n = 1, 2, \ldots$$
$$W_{n,k+1} = (1+x)^{-1} (W_{n+2,k} + x\, W_{n+1,k}) , \qquad n, k = 0, 1, \ldots$$

and we obtain $W_{0,n} = \bar{S}_n$.

It was proved by Longman [297] that if

$$c_k = \int_0^1 t^{k-1} g(t) dt$$

then $(\bar{S}_n)$ converges to $S = \int_0^1 \frac{g(t)}{1+xt}dt$ faster than (S_n). Numerical examples can be found also in these papers.

A technique based on shifted Jacobi polynomials was considered by Prévost [367]. Let $P_n^{(\alpha,\beta)}$ be the usual Jacobi polynomials defined by

$$\int_{-1}^{1} P_n^{(\alpha,\beta)}(x)P_m^{(\alpha,\beta)}(x)\,(1-x)^\alpha\,(1+x)^\beta\,dx = 0 \quad \text{if } n \neq m$$

with the normalization

$$P_n^{(\alpha,\beta)}(1) = \binom{n+\alpha}{n}.$$

The shifted Jacobi polynomials $P_n^{*(\alpha,\beta)}$ are defined by

$$P_n^{*(\alpha,\beta)}(x) = P_n^{(\alpha,\beta)}(2x-1).$$

They satisfy

$$\int_{-1}^{1} P_n^{*(\alpha,\beta)}(x)P_m^{*(\alpha,\beta)}(x)\,(1-x)^\alpha\,x^\beta\,dx = 0 \quad \text{if } n \neq m$$

$$P_n^{*(\alpha,\beta)}(0) = P_n^{(\alpha,\beta)}(-1) = (-1)^n\binom{n+\beta}{n}$$

$$P_n^{*(\alpha,\beta)}(1) = \binom{n+\alpha}{n}$$

and we have

$$P_n^{*(\alpha,\beta)}(x) = \sum_{i=0}^{n}\binom{n+\alpha}{n-i}\binom{n+\beta}{i}(x-1)^i x^{n-i}.$$

Prévost considered the sequence

$$T_n = \sum_{i=0}^{n}\binom{n+\alpha}{n-i}\binom{n+\beta}{i}\Delta^i S_{n-i} \Big/ P_n^{*(\alpha,\beta)}(1)$$

and he proved that if $(S_n) \in TM$, $\alpha > -1/2$, $\alpha > \beta$ then

$$\lim_{n\to\infty} T_n = \lim_{n\to\infty} S_n.$$

For example we consider

$$S_n = \sum_{i=1}^{n} \left[1 - \left(1 - i^{-3}\right)^i\right], \qquad n = 1, 2, \ldots$$

It is a logarithmic and totally monotonic sequence. From the 17 first terms of this series we obtain respectively

ε-algorithm	1.61074762160818
ϱ-algorithm	1.62231122658372
Θ-algorithm	1.62231122520473
Prévost algorithm	1.62233234119527
$(\alpha = 1, \beta = 0)$	

while $S_{17} = 1.56521360171339$.

If

$$S_n = S + \int_0^1 x^n \omega(x) dx$$

where $\omega(x) = O((1-x)^\varrho)$ when x tends to 1 and is square integrable, then Prévost [365] proved that the previous sequence (T_n), constructed with $\alpha = \varrho$ and $\beta = 0$, converges to S faster that (S_n).

If ω has the form

$$\omega(x) = \int_0^1 \frac{1}{1 - (ax + b)t} \, d\alpha(t)$$

where α is a function of bounded variation and $b \leq 1$, $a + b \leq 1$ then, instead of considering $T_n = c\left(P_n^{*(\alpha,\beta)}(x)\right) \Big/ P_n^{*(\alpha,\beta)}(1)$ (with $S_n = c(x^n)$), Prévost used the transformation

$$T'_n = c\left(P_n^*(x)\right) / P_n^*(1)$$

where P_n^* is the shifted Legendre polynomial on $[0, 1]$. If $a = -1$ and $b = 0$ then (T'_n) is still logarithmic but it converges to S faster than (S_n). If $a = -1$ and $b = 1$ then we can consider

$$T''_n = c\left(P_n(2x + 1)\right) / P_n(3)$$

and we have

$$\limsup_{n \to \infty} |T''_n - S|^{1/n} = 3 - \sqrt{8} \simeq 0.1716.$$

There are some methods for accelerating the convergence that are based on direct or inverse rational interpolation. They can be considered as extensions of the ϱ-algorithm (section 2.5) and are described by Wuytack [468], Larkin [273] and Mühlbach and Reimers [331].

Other methods consist in replacing the series to be computed (or its tail) by an integral and then evaluating this integral by a numerical method. For such a technique see Gustafson [205], Beckman, Fornberg and Tengvald [17] and Gustafson [206].

Finally for power series, acceleration methods can be based on approximation techniques. This is, in particular, the case for Padé approximations which can be obtained via the ε-algorithm. Other approximations lead to various acceleration methods. See, for example, Della Dora [141], Niethammer [334], Gabutti and Lyness [169], Prévost [366], Gabutti [167], Loi [293], Gabutti [168], Walz [453], Wimp [466]. It is not our purpose here to enter into the details of such transformations.

This page intentionally left blank

Chapter 3

SPECIAL DEVICES

The aim of this chapter is to show how to use some special devices in connection with extrapolation algorithms. As explained in section 1.4, the problem of accelerating the convergence of a sequence is equivalent to the problem of finding a perfect estimation of its error. As we shall see below such a perfect estimation of the error can be obtained either from a good estimation of the error or from the classical convergence tests for sequences and series. We shall also see how to construct an asymptotic expansion of the error in some cases, which will allow us to use efficiently the E-algorithm. Another possibility is to extract a subsequence from the sequence to be transformed and to accelerate its convergence. One can also use simultaneously several sequence transformations and then, at each step, select one answer among all the answers obtained from the various algorithms. Under some assumptions such a selection procedure, selects the best possible answer. The answers can also be combined together, thus leading to composite sequence transformations. We shall also see how to control the error in extrapolation processes. Finally when acceleration is impossible, special devices to obtain a smaller error can be used. Extrapolation in the least squares sense will also be discussed.

3.1 Error estimates and acceleration

In section 1.4, we saw that the problem of accelerating the convergence of a sequence (S_n) which converges to S is equivalent to the problem of finding a perfect estimation (D_n) of the error that is such that

$$\lim_{n\to\infty} D_n/(S - S_n) = 1.$$

If such a perfect estimation is known then the sequence

$$T_n = S_n + D_n\,, \qquad n = 0, 1, \ldots$$

converges to S faster than (S_n).

Sometimes only a good estimation (D_n) of the error is known, that is such that $\exists a \neq 0, 1$ and finite

$$\lim_{n\to\infty} D_n/(S - S_n) = a^{-1}.$$

In that case the convergence is not accelerated by the previous transformation since we have

$$\lim_{n\to\infty} (T_n - S)/(S_n - S) = 1 - a^{-1} \neq 1, 0.$$

$T : (S_n) \longmapsto (T_n)$ is said to be a synchronous process for (S_n), a notion introduced by Germain-Bonne and Kowalewski [184]. Let us now see how to construct from (T_n) (or, equivalently, from (D_n)) a new sequence transformation accelerating the convergence of (S_n).

Three cases can happen.

First, if the value of a is known then the sequence

$$U_n = S_n + aD_n\,, \qquad n = 0, 1, \ldots$$

converges faster than (S_n). This is the easiest case but it is not often seen in practice.

Secondly, if the sequence (S_n) is not logarithmic, that is if there exists α and β with $\alpha < 1 < \beta$ such that $\forall n \geq N, (S_{n+1} - S)/(S_n - S) \notin [\alpha, \beta]$ then it can be proved that

$$\lim_{n\to\infty} \Delta D_n/\Delta S_n = -a^{-1}.$$

Thus, in this case, the sequence (U_n) given by

$$U_n = S_n - \frac{\Delta S_n}{\Delta D_n} \cdot D_n\,, \qquad n = 0, 1, \ldots$$

converges to S faster than (S_n). This is exactly the application of the procedure Θ (see section 2.9) to the transformation T and it is also identical to the second standard process of Germain-Bonne [182].

The last case is the most difficult one since it covers all the possible situations. It is only assumed that $\forall n, S_n \neq S$. If a sequence (a_n) converging to a is known, then the sequence (U_n) given by

$$U_n = S_n + a_n D_n\,, \qquad n = 0, 1, \ldots$$

converges to S faster than (S_n). The construction of such a sequence (a_n) can be done by the so-called ACCES-algorithm (where ACCES stands for ACcélération de la Convergence d'un Ensemble Synchronisable) due to Litovsky [290]. This algorithm is based on the fact that the sign R_n of $S_n - S$ can be asymptotically obtained if a synchronous process for (S_n) is known (see the DAQES-algorithm below). We assume that the computation of D_n needs only $S_0, \ldots, S_n$. The ACCES-algorithm is as follows:

- Set $S_{-1} = T_{-1} = 0$.
- For $n = 0, 1, \ldots$
  - compute $T_n = S_n + D_n$.
  - compute R_n by the DAQES-algorithm (see below).
  - compute $p(n)$ by
    - i) If $R_n D_n > 0$ then $p(n) = \max(\{-1\} \cup \{j \in [0, n[\text{ such that } R_n(T_n - S_j) \leq 0\})$.
    - ii) If $R_n D_n < 0$ then $p(n) = \max(\{-1\} \cup \{j \in [0, n[\text{ such that } R_n(2S_n - T_n - S_j) \leq 0\})$.
    - iii) If $R_n D_n = 0$ then $p(n) = -1$.
  - compute

$$\begin{array}{ll} b_n = \left(T_n - T_{p(n)}\right) / \left(S_n - S_{p(n)}\right) & \text{if } S_n \neq S_{p(n)} \\ b_n = 1 & \text{otherwise} \end{array}$$

  - compute

$$\begin{array}{ll} U_n = S_n + a_n D_n \text{ with } a_n = (1 - b_n)^{-1} & \text{if } b_n \neq 1 \\ U_n = S_n & \text{if } b_n = 1 \end{array}$$

It can be proved that (U_n) converges to S faster than (S_n) and that there exists an index N such that $\forall n \geq N$, the condition $S_n \neq S_{p(n)}$ always holds. This algorithm shows that the problems of finding a synchronous transformation or an accelerative one are equivalent.

Let us now describe the DAQES-algorithm for determining asymptotically the sign R_n of $S_n - S$. We shall use the notation $< \alpha, \beta >$ to denote $[\alpha, \beta]$ if $\alpha \leq \beta$ and $[\beta, \alpha]$ if $\beta < \alpha$. We set

$$T_c(n) = cT_n + (1 - c)S_n.$$

Let (a_k) be an auxiliary strictly decreasing sequence which converges to zero. For example $a_k = 2^{-k}$. We shall denote respectively by $A(k, n)$ and $B(k, n)$ the following assertions

$$A(k,n) : \bigcap_{j=k}^{n} < S_j,\ T_{-1/a_k}(j) > \neq \emptyset$$

$$B(k,n) : \bigcap_{j=k}^{n} < S_j,\ T_{1/a_k}(j) > \neq \emptyset$$

where $\emptyset$ denotes the empty set.
Then, the DAQES-algorithm is as follows:

- Set $k = 0$.
- For $n = 0, 1, \ldots$

  i) If $A(k, n)$ is true then $R_n =$ sign D_n.

  ii) If $A(k, n)$ is false and $B(k, n)$ is true then $R_n =$ $-$ sign D_n.

  iii) If $A(k, n)$ and $B(k, n)$ are false then $R_n = R_{n-1}$ and $k = n+1$.

It can be proved that the sequence (R_n) given by this algorithm is asymptotically equal to the sign of $S_n - S$ that is $\exists N, \forall n \geq N, R_n(S_n - S) > 0$.

The subroutines ACCES and DAQES perform the ACCES and DAQES algorithms.

The ACCES-algorithm was tried on the sequence

$$S_n = (n+1)^{-1} \cdot \sin a(n+1), \quad n = 0, 1, \ldots$$

which tends to zero. If $a(n+1)$ is not a multiple of π, for all n, then the ratio S_{n+1}/S_n has no limit. S_n has infinitely many sign changes and the distance between two consecutive sign changes varies.

We took

$$T_n = S_n + b\left(S_n^2 - S_n\right), \quad n = 0, 1, \ldots$$

Obviously (T_n) tends to zero and (T_n/S_n) tends to $1-b$. Taking $a = 5.0$ and $b = 0.5$ we got the following results

n	S_n	R_n	U_n	U_n/S_n
0	$-0.959 \cdot 10^{+00}$	1.0	$-0.325 \cdot 10^{-18}$	$0.339 \cdot 10^{-18}$
1	$-0.272 \cdot 10^{+00}$	−1.0	$-0.117 \cdot 10^{+00}$	$0.430 \cdot 10^{+00}$
2	$0.217 \cdot 10^{+00}$	−1.0	$0.559 \cdot 10^{-01}$	$0.258 \cdot 10^{+00}$
3	$0.228 \cdot 10^{+00}$	−1.0	$0.595 \cdot 10^{-01}$	$0.261 \cdot 10^{+00}$
4	$-0.265 \cdot 10^{-01}$	−1.0	$-0.555 \cdot 10^{-02}$	$0.209 \cdot 10^{+00}$
5	$-0.165 \cdot 10^{+00}$	−1.0	$-0.312 \cdot 10^{-01}$	$0.189 \cdot 10^{+00}$
6	$-0.612 \cdot 10^{-01}$	−1.0	$-0.822 \cdot 10^{-02}$	$0.134 \cdot 10^{+00}$
7	$0.931 \cdot 10^{-01}$	1.0	$-0.313 \cdot 10^{-01}$	$-0.336 \cdot 10^{+00}$
8	$0.945 \cdot 10^{-01}$	1.0	$-0.319 \cdot 10^{-01}$	$-0.337 \cdot 10^{+00}$
9	$-0.262 \cdot 10^{-01}$	−1.0	$-0.148 \cdot 10^{-02}$	$0.563 \cdot 10^{-01}$
10	$-0.909 \cdot 10^{-01}$	−1.0	$-0.119 \cdot 10^{-01}$	$0.131 \cdot 10^{+00}$
50	$-0.993 \cdot 10^{-02}$	−1.0	$-0.187 \cdot 10^{-03}$	$0.189 \cdot 10^{-01}$
100	$0.708 \cdot 10^{-02}$	1.0	$-0.824 \cdot 10^{-04}$	$-0.116 \cdot 10^{-01}$
150	$0.564 \cdot 10^{-02}$	1.0	$-0.489 \cdot 10^{-04}$	$-0.869 \cdot 10^{-02}$
200	$-0.152 \cdot 10^{-02}$	−1.0	$-0.760 \cdot 10^{-05}$	$0.501 \cdot 10^{-02}$
250	$-0.398 \cdot 10^{-02}$	−1.0	$-0.248 \cdot 10^{-04}$	$0.623 \cdot 10^{-02}$
300	$-0.585 \cdot 10^{-03}$	−1.0	$-0.193 \cdot 10^{-05}$	$0.330 \cdot 10^{-02}$

We see that the sign R_n of $S_n - S$ is well determined by the DAQES-algorithm for $n \geq 4$. We also see that the ratio (U_n/S_n) tends to zero as predicted by the theory.

Thus the ACCES-algorithm accelerates the convergence of any sequence (S_n) for which a synchronous transformation is known or, in other words, for which a good (or a perfect) estimation of the error is known that is such that

$$S - S_n = aD_n + o(D_n), \quad n = 0, 1, \ldots$$

aD_n appears as the first term in an asymptotic expansion of the error $S - S_n$ with respect to some asymptotic sequence. In the next section we

shall see how to obtain good and perfect estimations of the error from the usual convergence tests for sequences and series.

But, before going to that case, let us explain how to accelerate the convergence of a sequence for which a good estimation of the absolute value of the error is known. The details and the proofs can be found in Matos [312].

Let (S_n) be a sequence converging to S and such that a sequence (D_n) satisfying

$$\lim_{n\to\infty} |S_n - S|/D_n = \alpha \neq 0$$

is known.

We consider the following algorithm:

- Choose ε and M such that $\varepsilon < \alpha < M$.
- For $n = 0, 1, \ldots$

  i) If $\left|\dfrac{\Delta S_n}{\Delta D_n}\right| > M$ then set $T_n = S_n + \dfrac{\Delta S_n}{D_n + D_{n+1}} \cdot D_n$

  ii) If $\left|\dfrac{\Delta S_n}{D_n + D_{n+1}}\right| < \varepsilon$ then set $T_n = S_n - \dfrac{\Delta S_n}{\Delta D_n} \cdot D_n$

We have the

Theorem 3.1

If $|S_n - S| = \alpha D_n + r_n$ *where* $\lim_{n\to\infty} D_{n+1}/D_n = 1$ *and* $r_n \sim \beta D'_n\ (n \to \infty)$ *with* $\lim_{n\to\infty} D'_n/D_n = \lim_{n\to\infty} \Delta D'_n/\Delta D_n = 0$ *then the sequence* (T_n) *defined by the previous algorithm converges to* S *faster than* (S_n).

We recall that the notation $u_n \sim v_n$ means that $\lim_{n\to\infty} u_n/v_n = 1$.

In the case where (D_n) converges linearly, we shall make use of a different algorithm. We set

$$T_n^{(1)} = S_n - \frac{\Delta S_n}{\Delta D_n} \cdot D_n\,; \quad T_n^{(2)} = S_n + \frac{\Delta S_n}{D_n + D_{n+1}} \cdot D_n$$

$$\text{and } T_n^{(3)} = \frac{T_n^{(1)} \Delta T_n^{(2)} - T_n^{(2)} \Delta T_n^{(1)}}{\Delta T_n^{(2)} - \Delta T_n^{(1)}}.$$

Let us define the functions A and B by

$$A(r) = \frac{\varrho+1}{\varrho} + m\left(r + \varrho \cdot \frac{\varrho+1}{\varrho-1}\right) \text{ with } m = -2 \cdot \frac{\varrho+1}{\varrho^2} \cdot \frac{(\varrho-1)(\varrho+1)}{(\varrho+1)^2 + (\varrho-1)^2}$$

$$B(r)=\frac{\varrho-1}{2\varrho}+M\left(r+\varrho\cdot\frac{\varrho+1}{\varrho-1}\right) \text{ with } M=-\frac{\varrho-1}{\varrho^2}\cdot\frac{(\varrho-1)(\varrho+1)}{(\varrho+1)^2+(\varrho-1)^2}$$

and the sequence (T_n) by

$$\begin{aligned} T_n &= (\varrho+r_n)B(r_n)T_n^{(1)}+(r_n-\varrho)A(r_n)T_n^{(2)} \\ &\quad +[1-(\varrho+r_n)B(r_n)-(r_n-\varrho)A(r_n)]\ T_n^{(3)} \end{aligned}$$

with $r_n=\Delta S_{n+1}/\Delta S_n$.

We have the

Theorem 3.2

Let (S_n) be a sequence converging to S for which a sequence (D_n) such that $\lim_{n\to\infty}|S_n-S|/D_n=\alpha\neq 0$ and $\lim_{n\to\infty}D_{n+1}/D_n=\varrho$ with $0<\varrho<1$ is known. Then the sequence (T_n) given by the previous algorithm converges to S faster than (S_n). In this algorithm if the (usually unknown) parameter ϱ is replaced by $\varrho_n=D_{n+1}/D_n$, then the same acceleration result still holds.

$T_n^{(3)}$ as defined above is a so-called composite sequence transformation. Such transformations will be studied in details in section 3.7.

3.2 Convergence tests and acceleration

As explained in the previous section, we shall now see how the classical convergence tests for sequences and series can produce good and perfect estimations of the error and, thus, convergence acceleration methods.

Let (x_n) be an auxiliary increasing sequence with a known limit x. Let us assume that (S_n) also increases. We set $R_n=S-S_n, r_n=x-x_n, A_n=\Delta x_n/\Delta S_n$ and we assume that there exists $A\neq 0$ such that (A_n) converges to A. Then (S_n) converges and $\forall n$, we have

$$\frac{r_n}{A}\leq R_n\leq\frac{r_n}{A_n} \text{ if } (A_n) \text{ is increasing}$$

$$\frac{r_n}{A}\geq R_n\geq\frac{r_n}{A_n} \text{ if } (A_n) \text{ is decreasing.}$$

Thus, in both cases, (r_n/A) and (r_n/A_n) are perfect estimations of the error of (S_n) and the sequences (T_n) and (U_n) given by

$$\begin{aligned} T_n &= S_n+\frac{(x-x_n)}{A}, \quad n=0,1,\ldots \\ U_n &= S_n-\frac{\Delta S_n}{\Delta x_n}\cdot(x_n-x), \quad n=0,1,\ldots \end{aligned}$$

both converge to S faster than (S_n).

The preceding convergence test reduces to d'Alembert's for $x_n = \Delta S_n$, to Cauchy's for $\Delta x_n = \lambda^n$ where $\lambda \in]0,1[$ and to Raabe-Duhamel's for $x_n = n\Delta S_n$. In the first case U_n is Aitken's Δ^2 process. The choices $x_n = \Delta S_n$ also corresponds to Levin's t-transform while $x_n = (n+1)\Delta S_n$ and $x_n = \Delta S_n \Delta S_{n-1}/\Delta^2 S_{n-1}$ give respectively Levin's u and v-transforms.

The use of such a convergence test for obtaining convergence acceleration methods was introduced by Brezinski [77] and extended by Matos [311] whose results will be now presented.

They are based on the following more general convergence test given by Lyusternik and Yanpol'skii [305] for monotonous sequences.

We set $A_k(m) = \Delta x_k/\Delta S_{k+m}$. Then if $\liminf_{n\to\infty} A_n(m) > 0$ and $\forall n, r_n < +\infty$ or if $\limsup_{n\to\infty} A_n(m) < 0$ and $\forall n, r_n > -\infty$ then the sequence (S_n) is convergent. In fact, in the first case we have $\forall n$

$$\frac{r_n}{\sup_{k\geq n} A_k(m)} \leq R_{n+m} \leq \frac{r_n}{\inf_{k\geq n} A_k(m)}$$

while, in the second case, the inequalities are reversed. Thus if, for m fixed, $(A_n(m))_n$ converges to a non-zero limit then we have

$$\lim_{n\to\infty} \frac{R_{n+m}}{r_n/A_n(m)} = 1$$

which shows that $(r_{n-m}/A_{n-m}(m))_n$ is a perfect estimation of the error of (S_n) and we have the

Theorem 3.3

Let (S_n) be a monotonous sequence and (x_n) a sequence converging to a known finite limit x. If, for m fixed, $\lim_{n\to\infty} A_n(m)$ exists and is different from zero, then the sequence (T_n) given by

$$T_n = S_n + (x - x_n)/A_{n-m}(m), \qquad n = 0, 1, \ldots$$

converges to S faster than (S_n).

We shall now study various choices for (x_n) and determine for each choice under which assumptions on (S_n) the conditions of the preceding theorem are satisfied.

We shall take $m = 0$ and, set $A_n = A_n(0)$.

The choice $x_n = \Delta S_n$ corresponds to d'Alembert's test and leads to

Aitken's Δ^2 process which accelerates sequences converging linearly. The choice $x_n = (1-\varrho^n)/(1-\varrho)$ where $\varrho \in]0,1[$ corresponds to Cauchy's test and leads to the first standard process of Germain-Bonne [182]. This process can be iterated under some additional assumptions and we obtain the

Theorem 3.4

Let (S_n) be a monotonous sequence such that, $\forall n$

$$\Delta S_n = a_1 \varrho_1^n + a_2 \varrho_2^n + \cdots$$

with $\forall i, a_i \neq 0$ and $1 > \varrho_1 > \varrho_2 > \cdots > 0$.
We consider the following algorithm

$$T_0^{(n)} = S_n\,, \quad n = 0, 1, \ldots$$

$$T_{k+1}^{(n)} = T_k^{(n)} - \frac{\Delta T_k^{(n)}}{(1-\varrho_{k+1})}\,, \quad k, n = 0, 1, \ldots$$

Then, $\forall k$, $\lim_{n\to\infty} \left(T_{k+1}^{(n)} - S\right) / \left(T_k^{(n)} - S\right) = 0$.

Let us now consider the choice $x_n = \Delta S_n \Delta S_{n+1} / \Delta^2 S_n$ which corresponds to Levin's v-transform. We have the

Theorem 3.5

Let (S_n) be a monotonous sequence such that $\exists \lambda \in]0,1[$, $\lim_{n\to\infty} \Delta S_{n+1} / \Delta S_n = \lambda$ or $\Delta S_{n+1}/\Delta S_n - 1 = \lambda_n$ with $\lim_{n\to\infty} \lambda_n = 0$, $\lim_{n\to\infty} \Delta S_n/\lambda_n = 0$ and $\lim_{n\to\infty} (\lambda_{n+1}^{-1} - \lambda_n^{-1}) \neq 1$. Then the sequence (T_n) given by

$$T_n = S_n - \frac{(\Delta S_n)^2 \Delta^2 S_{n+1}}{\Delta S_{n+2} \Delta^2 S_n - \Delta S_n \Delta^2 S_{n+1}}\,, \quad n = 0, 1, \ldots$$

converges to S faster than (S_n).

Let us now consider the choice $x_n = a_n \Delta S_n$ where (a_n) is an auxiliary sequence. It corresponds to Kummer's test and we obtain the

Theorem 3.6

Let (S_n) be a monotonous sequence and (a_n) an auxiliary sequence such that

$$\lim_{n\to\infty} a_n \Delta S_n = 0$$

$$\lim_{n\to\infty}\left(a_{n+1}\cdot\frac{\Delta S_{n+1}}{\Delta S_n}-a_n\right)\neq 0.$$

Then the sequence (T_n) *given by*

$$T_n=S_n-\frac{a_n(\Delta S_n)^2}{a_{n+1}\Delta S_{n+1}-a_n\Delta S_n},\quad n=0,1,\ldots$$

converges to S *faster than* (S_n).

Let us now examine some possible choices for the auxiliary sequence (a_n). If (S_n) converges linearly and if (a_n) converges to a limit different from zero then the conditions of the preceding theorem are satisfied. The problem is not so simple for logarithmic sequences since it is a remanent set and thus no universal choice for (a_n) could exist.

Let LOGSF be the set of sequences such that $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=\lim_{n\to\infty}\Delta S_{n+1}/\Delta S_n=1$. This set is also remanent and it cannot be accelerated. This is the reason why we shall now study some of its subsets for which a choice of (a_n) satisfying the assumptions of theorem 3.6 (and thus giving an acceleration method) can be found.
Choosing $a_n=n$, we have the

Theorem 3.7

Let (S_n) *be a monotonous sequence of* LOGSF *such that* $\exists\beta<-1$,

$$\lim_{n\to\infty}(n+1)\cdot\left(\frac{\Delta S_{n+1}}{\Delta S_n}-1\right)=\beta.$$

Then the sequence (T_n) *given by*

$$T_n=S_n-\frac{n(\Delta S_n)^2}{(n+1)\Delta S_{n+1}-n\Delta S_n},\quad n=0,1,\ldots$$

converges to S *faster than* (S_n).

This procedure can be iterated under some supplementary assumptions and we obtain the

Theorem 3.8

Let (S_n) *be a monotonous sequence of* LOGSF *such that*

$$\frac{\Delta S_{n+1}}{\Delta S_n}=1+\frac{a}{n}\cdot(1+\nu_n),\quad n=1,2,\ldots$$

with $a < -1$ *and* $v_n \sim Kn^{\alpha}(\ln n)^{\beta}$ $(n \to \infty)$ *where* $\alpha \leq 0$ *and* $\beta < 0$ *if* $\alpha = 0$.
We consider the following algorithm

$$T_0^{(n)} = S_n\,, \quad n = 0, 1, \ldots$$

$$T_{k+1}^{(n)} = T_k^{(n)} - \frac{n\left(\Delta T_k^{(n)}\right)^2}{(n+1)\Delta T_k^{(n+1)} - n\Delta T_k^{(n)}}\,, \quad k, n = 0, 1, \ldots$$

Then, $\forall k$, $\lim_{n\to\infty} \left(T_{k+1}^{(n)} - S\right) \Big/ \left(T_k^{(n)} - S\right) = 0$.

The choice $a_n = n$ corresponds to Raabe-Duhamel's and Gauss' tests. The assumptions of theorem 3.8 are satisfied if

$\Delta S_n = c_1 n^{-\alpha_1} + c_2 n^{-\alpha_2} + \cdots$ with $1 < \alpha_1 < \alpha_2 < \cdots$
or if
$\Delta S_n = c_1 n^{-\alpha_1}(\ln n)^{-\beta_1} + c_2 n^{-\alpha_2}(\ln n)^{-\beta_2} + \cdots$
with $\beta_1 < \beta_2 < \cdots$ and $1 < \alpha_1 = \alpha_2 = \cdots$ or with $1 < \alpha_1 < \alpha_2 < \cdots$

In these cases the convergence is accelerated even if the values of the α_i's and β_i's are unknown, thus showing the interest of such transformations.

Let us give an application of theorems 3.7 and 3.8. It concerns the Riemann ξ function

$$\xi(t) = \sum_{i=1}^{\infty} i^{-t}$$

which can be expressed as a continued fraction

$$\xi(t) = \frac{1}{\left\lceil 1 \right\rceil} + \frac{a_2}{\left\lceil b_2 \right\rceil} + \frac{a_3}{\left\lceil b_3 \right\rceil} + \cdots$$

with $a_n = -\left(1 - n^{-1}\right)^t$ and $b_n = 1 - a_n$. Let (C_n) be the convergents of this continued fraction. Kooman [264] using the results given in Kooman [263] (see also Kooman and Tijdeman [265]) about second order difference equations with rational coefficients proved that

$$\frac{C_{n+1} - \xi(t)}{C_n - \xi(t)} = 1 + \frac{1-t}{n} + O\left(\frac{1}{n^2}\right)$$

and

$$\frac{\Delta C_n}{\Delta C_{n-1}} = 1 - \frac{t}{n} + O\left(\frac{1}{n^2}\right).$$

Thus the conditions of theorem 3.7 and 3.8 are satisfied for $t > 1$ and we obtain the following numerical results (number of exact digits) for $t = 3$

$n\backslash k$	0	1	2	3	4	5
2	1.19	1.06	1.74	2.43	3.22	4.07
3	1.48	1.70	2.44	3.22	4.07	4.98
4	1.69	2.12	2.92	3.77	4.68	5.64
5	1.87	2.42	3.29	4.20	5.16	
6	2.01	2.67	3.60	4.56		
7	2.13	2.88	3.85			
8	2.24	3.06				
9	2.34					

With $t = 7$, we find

$n\backslash k$	0	1	2	3	4	5
2	3.27	3.31	4.48	5.64	6.79	7.96
3	4.10	4.46	5.62	6.78	7.95	9.14
4	4.74	5.29	6.46	7.64	8.83	10.0
5	5.25	5.94	7.13	8.33	9.55	
6	5.68	6.47	7.69	8.92		
7	6.05	6.93	8.18			
8	6.37	7.33				
9	6.66					

Due to rounding errors the maximum number of exact digits obtained was 12. After, precision deteriorates.

Let us now choose $a_n = n \cdot \ln n$ which corresponds to Bertrand's test with $p = 1$. We have the

Theorem 3.9

Let (S_n) *be a monotonous sequence of* LOGSF *such that* $\exists \beta < -1$,

$$\lim_{n\to\infty} (n+1) \cdot \ln(n+1) \cdot \left[\Delta S_{n+1}/\Delta S_n - 1 + (n+1)^{-1}\right] = \beta.$$

Then the sequence (T_n) *given by*

$$T_n = S_n - \frac{(\Delta S_n)^2}{b_n \cdot \Delta S_{n+1} - \Delta S_n}, \quad n = 0, 1, \ldots$$

with

$$b_n = \frac{n+1}{n} \cdot \frac{\ln(n+1)}{\ln n}$$

converges to S faster than (S_n).

With some additional assumptions, this procedure can be iterated, thus leading to the

Theorem 3.10

Let (S_n) be a monotonous sequence of LOGSF *such that*

$$\frac{\Delta S_{n+1}}{\Delta S_n} = 1 - \frac{1}{n+1} + \frac{\beta}{(n+1)\cdot\ln(n+1)} \cdot (1+\nu_n), \quad n = 0,1,\ldots$$

with $\beta < -1$ and $\nu_n \sim c(\ln n)^{-\gamma}$ $(n \to \infty)$ where $\gamma > 0$ and $c \neq 0$.
We consider the following algorithm

$$T_0^{(n)} = S_n, \quad n = 0,1,\ldots$$

$$T_{k+1}^{(n)} = T_k^{(n)} - \frac{\Delta T_k^{(n)}}{b_n \cdot \Delta T_k^{(n+1)}/\Delta T_k^{(n)} - 1}, \quad k,n = 0,1,\ldots$$

with

$$b_n = \frac{n+1}{n} \cdot \frac{\ln(n+1)}{\ln n}.$$

Then, $\forall k$, $\lim_{n\to\infty} \left(T_{k+1}^{(n)} - S\right) \Big/ \left(T_k^{(n)} - S\right) = 0$.

Finally we consider the choice $a_n = n \cdot \ln n \cdot \ln_2 n \cdot \ldots \cdot \ln_p n$ where $\ln_{i+1} n = \ln(\ln_i n)$ with $\ln_1 n = \ln n$. It corresponds to Bertrand's test and we have the

Theorem 3.11

Let (S_n) be a monotonous sequence of LOGSF *such that* $\exists \beta < -1$,

$$\lim_{n\to\infty} L_p(n+1) \cdot \left[\frac{\Delta S_{n+1}}{\Delta S_n} - 1 + \frac{1}{n+1} + \cdots + \frac{1}{L_{p-1}(n+1)}\right] = \beta$$

where $L_p(n) = n \cdot \ln n \cdot \ldots \cdot \ln_p n$.
Then the sequence (T_n) given by

$$T_n = S_n - \frac{L_p(n) \cdot \Delta S_n}{L_p(n+1) \cdot \Delta S_{n+1}/\Delta S_n - L_p(n)}, \quad n = 0,1,\ldots$$

converges to S faster than (S_n).

Thus, as seen from the examples given above, the classical convergence tests for sequences and series are powerful tools for obtaining perfect estimations of the error and constructing convergence acceleration methods. In some cases, integral tests can also be useful. For example if $S_n = \sum_{i=1}^{n} f(i)$ where $f(x) \geq 0$ and decreasing for all $x \geq 0$, it is well known that $\forall n$

$$D_{n+1} \leq R_n \leq D_n = \int_n^{\infty} f(x)\, dx.$$

If $f(x) = 1/x^2$ then $(D_n = 1/n)$ is a perfect estimation of the error of (S_n).

For series with positive terms, lower and upper bounds for R_n whose ratio tends to one are needed in order to obtain perfect estimations of the error. Such bounds are known for some series, see Fabry [153]. For example we have

- For $\sum_{n=1}^{\infty} 1/n^{1+a}$ with $a > 0$

$$\frac{1}{a(n+1)^a} < R_n < \frac{1}{an^a}.$$

- For $\sum_{n=1}^{\infty} 1/(n+b)^{1+a}$ with $b > 0$

$$\frac{(n+k-2a-1)}{a(n+1)^{1+a}} < R_n < \frac{(n+b-1)}{a(n+b)^{1+a}}$$

where k is any real number strictly greater than one.

- For $\sum_{n=1}^{\infty} u_n$ with $\frac{u_{n+1}}{u_n} = \frac{n^p + a_1 n^{p-1} + a_2 n^{p-2} + \cdots}{n^p + b_1 n^{p-1} + b_2 n^{p-2} + \cdots}$ and $a = b_1 - a_1 - 1 > 0$

$$\frac{(n+k-2a-1)u_n}{a} < R_n < \frac{(n+h-1)u_n}{a}$$

where h and k are real numbers such that $h > b_1 + (a_2 - b_2)/(a+1) > k$.

- For $\sum_{n=1}^{\infty} u_n$ with $u_n = \dfrac{(1-b)\cdot(2-b)\cdot\ldots\cdot(n-b)}{(n+1)\cdot(p+a)\cdot\ldots\cdot(p+a+n)}$ and $b \geq 0, p+a>0$

$$\frac{(n-p-a-2b)u_n}{(p+a+b)} < R_n < \frac{(n+p+a+1)u_n}{(p+a+b)}.$$

3.3 Construction of asymptotic expansions

As seen from theorem 2.10, the E-algorithm achieves its full power when an asymptotic expansion of the error is known. However, this case is not often gained in practice. We shall now see how to obtain such an asymptotic expansion from the asymptotic expansion of ΔS_n or from that of $1/(S_n - S)$. For the details and the proofs, the interested reader is referred to Matos [309, 313]. This section is very technical and can be omitted at a first lecture.

Let us assume that an asymptotic expansion of ΔS_n is known, that is

$$\Delta S_n = a_1 g_1(n) + a_2 g_2(n) + \cdots$$

with $\forall i, \lim_{n\to\infty} g_{i+1}(n)/g_i(n) = 0$, $g_i(n)\cdot g_i(n+1) > 0$ $\forall n$, $\lim_{n\to\infty} g_i(n+1)/g_i(n) = b_i, 0 < b_i < 1$. We shall now try to obtain an asymptotic expansion of the error $S_n - S$. We set

$$R_i(n) = \sum_{m=0}^{\infty} g_i(n+m+1)$$

and summing up the asymptotic expansions of $\Delta S_n, \Delta S_{n+1}, \ldots$ we get

$$S - S_n = a_1 R_1(n-1) + a_2 R_2(n-1) + \cdots$$

If the R_i's are explicitly known, then the E-algorithm can be applied and it leads to convergence acceleration at each column (theorem 2.10). However, usually, this is not the case and we shall now see how to estimate these R_i's. We shall set

$$G_i(n,p) = \sum_{j=0}^{p} g_i(n+j)$$

and we have

$$R_i(n-1) = G_i(n,p) + R_i(n+p).$$

Thus

$$S-S_n = a_1G_1(n,p)+a_2G_2(n,p)+\cdots+a_kG_k(n,p)+a_1R_1(n+p) + a_2R_2(n+p)+\cdots+a_kR_k(n+p)+A_n$$

where A_n designates the remainder. It is easy to see that $\forall i$, $\lim_{n\to\infty} G_i(n+1,p)/G_i(n,p) = b_i$ and $\lim_{n\to\infty} G_{i+1}(n,p)/G_i(n,p) = 0$.

Thus in order to be under the conditions of acceleration for the E-algorithm given by theorem 2.10, it is necessary to estimate the R_i's and then to classify all the terms of the development of $S - S_n$ to obtain an asymptotic expansion of the error. The procedure for doing this work is described in the following theorem

Theorem 3.12

Let (S_n) be such that, $\forall n$

$$\Delta S_n = a_1g_1(n) + a_2g_2(n) + \cdots + a_{k-1}g_{k-1}(n) + r_k(n)$$

with

i) $\lim_{n\to\infty} g_{i+1}(n)/g_i(n) = 0$ *for* $i = 1,\ldots,k-1$ *and where* $(g_k(n))$ *is such that* $\exists N, \forall n \geq N, |r_k(n)| \leq c|g_k(n)|$ *with* $c > 0, r_k(n)\cdot r_k(n+1) > 0$ *and* $\lim_{n\to\infty} g_k(n+1)/g_k(n) = \varrho_k$ *with* $0 < \varrho_k < 1$.

ii) $\forall i = 1,\ldots,k-1, \exists m_i \geq 1$ *such that* $\lim_{n\to\infty} g_i(n+1)/g_i(n) = \varrho_i$ *with* $0 < \varrho_i < 1$ *and for* $j = 0,\ldots,m_i$, $\lim_{n\to\infty} \Delta\alpha_n^{(j)}/\Delta\alpha_{n-1}^{(j)} = c_j, 0 < c_j \leq 1$ *with* $\alpha_n^{(j)} = g^{(j)}(n+1)/g^{(j)}(n) - \varrho_j$ *and* $\lim_{n\to\infty} \alpha_n^{(j)} = 0$ *and where the* $g^{(j)}$*'s are recursively defined by* $g^{(0)}(n) = g_i(n)$ *and* $g^{(p)}(n) = g^{(p-1)}(n)\Delta h_{n-1}^{(p-1)}$ *with* $h_{n-1}^{(p-1)} = g^{(p-1)}(n-1)/\Delta g^{(p-1)}(n-1)$ *(Δ operating on the index n).*

Let us now define the sequences $\left(F_j^{(i)}(n,p)\right)_n$ *for* $p \geq -1$ *fixed,* $j = 1,\ldots,k-1$ *and* $i = 1,\ldots,m_j+2$ *by*

$$F_j^{(1)}(n,p) = -G_j(n,p) + \frac{(g_j(n+p+1))^2}{\Delta g_j(n+p+1)}$$

and

$$F_j^{(i)}(n,p) = \frac{\left(g_j^{(i-1)}(n+p+i)\right)^2}{\Delta g_j^{(i-1)}(n+p+i)} \text{ for } i \geq 2.$$

We assume that $\forall i<k, \exists p_i \leq m_i+1$ *such that* $\lim_{n\to\infty} F_i^{(p_i+1)}(n,p)/g_k(n) = B_i \neq 0$ *and* $\lim_{n\to\infty} g_k(n)/\, F_i^{(p_i)}(n,p) = 0$. *We set, for* $i = 1,\ldots,k-1$

$$D_i^{(p_i)}(n) = F_i^{(1)}(n,p) + \sum_{j=2}^{p_i}(-1)^j F_i^{(j)}(n,p).$$

Then

$$S - S_n = a_1 D_1^{(p_1)}(n) + a_2 D_2^{(p_2)}(n) + \cdots + a_{k-1} D_{k-1}^{(p_{k-1})}(n) + r_k'(n)$$

with $\lim_{n\to\infty} D_i^{(p_i)}(n) \Big/ D_{i-1}^{(p_{i-1})}(n) = 0$ *for* $i = 2,\ldots,k-1$,
$\lim_{n\to\infty} r_k'(n)/D_{k-1}^{(p_{k-1})}(n) = 0$ *and* $\exists N, \forall n \geq N, |r_k'(n)| \leq c|g_k(n)|$ *with* $c>0$.

Of course, the procedure given is this theorem is quite complicated and it is not so easy to check if its assumptions are satisfied or not. However it must be understood that some sequences are really difficult to accelerate and require such a treatment. The preceding theorem ensures that if the E-algorithm is applied to (S_n) with $g_i(n) = D_i^{(p_i)}(n)$ for $i = 1,\ldots,k-1$ then $\lim_{n\to\infty} \left(E_i^{(n)} - S\right) \Big/ \left(E_{i-1}^{(n)} - S\right) = 0$ for $i = 1,\ldots,k-1$. When $k = 2$ we have the

Theorem 3.13

Let (S_n) *be such that* $\Delta S_n = a g_1(n) + r_2(n), \forall n$. *If the conditions of theorem 3.12 are satisfied for* $k = 2$ *then* $\forall i \leq p_1$, *the sequence* $\left(T_n^{(i)}\right)_n$ *given by*

$$T_n^{(i)} = S_n - \frac{\Delta S_n}{\Delta D_1^{(i)}(n)} \cdot D_1^{(i)}(n), \quad n = 0,1,\ldots$$

converges to S *faster than* $\left(T_n^{(i-1)}\right)_n$.

Other procedures and numerical results are given by Matos [309]. In particular a generalization of the procedure of theorem 3.12 for logarithmic sequences can be found.

The case where an asymptotic expansion of $\Delta S_n/\Delta S_{n-1}$ is known was considered by Overholt [352]. We assume that $\forall n$

$$\varrho_n = \frac{\Delta S_n}{\Delta S_{n-1}} = a\left(1 + \frac{r_1}{n} + \frac{r_2}{n^2} + \cdots\right).$$

We shall say that the transformation $T : (S_n) \longmapsto (T_n)$ is of P-order p for (S_n) if, $\forall n$

$$\frac{\Delta T_n}{\Delta S_n} = \frac{c_p}{n^p} + \frac{c_{p+1}}{n^{p+1}} + \cdots \quad \text{with} \quad c_p \neq 0, p \geq 0,$$

and we shall write $T(p)(S_n)$. Then we have the

Theorem 3.14

i) If $T(p)(S_n)$ *and* $U(q)(T_n)$ *then*

$$U(p+q)(S_n).$$

ii) If $T(p)(S_n)$ *then* $\forall n$,

$$\frac{\Delta T_n}{\Delta T_{n-1}} = b\left(1 + \frac{r_1 - p}{n} + \frac{r_2'}{n^2} + \cdots\right).$$

iii) If $T(p)(S_n)$ *then*

$$(T_n - S)/(S_n - S) = O\left(\frac{1}{n^p}\right).$$

We shall now see how to construct such transformations.
Two cases will be considered

- 2-point formulæ of the form $T_n = S_n + \Delta S_{n-1}\Phi_2(\{\text{pars}\})$ where $\{\text{pars}\}$ is a set (possibly empty) of parameters such as a, n, r_1, r_2.
- 3-point formulæ of the form $T_n = S_n + \Delta S_{n-1}\Phi_3(\varrho_{n-1}, \{\text{pars}\})$.

These two cases can be treated in a common formalism by writing

$$T_n = S_n + q_{n-1}\Delta S_{n-1}$$

with

$$\begin{aligned} q_{n-1} &= q(\{a, n-1, r_1\}) \text{ for 2-point formulæ and} \\ q_{n-1} &= q(\varrho_{n-1}, \{a, n-1, r_1\}) \text{ for 3-point formulæ.} \end{aligned}$$

Thus in both cases the knowledge of a and r_1 is required for constructing the transformation. As we shall see below, r_2 could also be needed. We assume that $a \neq 1$ (non-logarithmic case).

Let us first consider the case of 2-point formulæ and assume that q_n is a rational function of its arguments having an expansion of the form

$$q_n = a\left(a_0 + \frac{a_1}{n} + \frac{a_2}{n^2} + \cdots\right).$$

We have the

Theorem 3.15
If $a \neq 1$ and

i) if $a_0 = 1/(1-a)$ then T has the P-order 1.

ii) if $a_0 = 1/(1-a)$ and $a_1 = a_0^2 r_1$ then T has the P-order 2.

iii) if $a_0 = 1/(1-a), a_1 = a_0^2 r_1$ and $a_2 = a_0^2\left((a_0-1)r_1^2 - a_0 r_1 + r_2\right)$ then T has the P-order 3.

Thus the knowledge of a is always needed. That of r_1 is needed for constructing methods of P-order 2 and r_2 is required for obtaining methods of P-order 3.

Let us now consider 3-point formulæ and rewrite them as

$$T_n = S_{n-1} + p_{n-1}\Delta S_{n-1}$$

with

$$p_{n-1} = 1 + q_{n-1} = \frac{1}{1 - \varrho_n - \sigma_n}.$$

We assume that σ_n is chosen such that it has an expansion of the form

$$\sigma_n = a\left(b_0 + \frac{b_1}{n} + \frac{b_2}{n^2} + \cdots\right).$$

We have the

Theorem 3.16
If $a \neq 1$ and

i) if $b_0 = b_1 = 0$ then T has the P-order 2.

ii) if $b_0 = b_1 = 0$ and $b_2 = -a_0 r_1$ with $a_0 = 1/(1-a)$ then T has the P-order 3.

iii) if $b_0 = b_1 = 0$, $b_2 = -a_0 r_1$ and $b_3 = -a_0\left((a_0-1)r_1^2 + (1-2a_0)r_1 + 2r_2\right)$ then T has the P-order 3.

For example, if

$$\sigma_n = \frac{\varrho_n - a}{n(\varrho_n - 1)} \text{ or } \sigma_n = \frac{\varrho_n - a}{n(\varrho_n - 1) - \varrho_n - 1}$$

then the method has the P-order 3. For the choice

$$\sigma_n = \frac{(\varrho_n - a)^2}{(\varrho_n - 1 - (a+1)/n)\, a x_1}$$

we have a method of P-order 4.

When $a = 1$ (logarithmic sequence) the choice $p_n = -n/(1 + r_1)$ leads to a method of P-order 1 and $p_n = r_1(r_1 - 1)^{-1}(1 - \varrho_n)^{-1}$ to a method of P-order 2.

To illustrate these processes, let us take again the example of the continued fraction for the Riemann ξ function.

For $t = 3$, we obtain (number of exact digits)

n	C_n	P-order 1	P-order 2
2	1.19		
5	1.87	2.14	1.78
8	2.24	2.89	2.31
11	2.50	3.36	2.64
14	2.70	3.71	2.88
17	2.87	3.98	3.06
20	3.00	4.21	3.22

With $t = 7$, we have

n	C_n	P-order 1	P-order 2
2	3.27		
5	5.25	4.41	4.75
8	6.37	6.15	6.29
11	7.15	7.26	7.27
14	7.75	8.07	7.97
17	8.24	8.70	8.53
20	8.65	9.23	8.99

Let us now assume that an asymptotic expansion of the inverse of the error is known. Then the following procedure was proposed by Matos [313]

Theorem 3.17

Let (S_n) *be such that,* $\forall n$

$$S_n - S = (a_1 g_1(n) + a_2 g_2(n) + \cdots)^{-1}$$

with $\forall i$, $\lim_{n\to\infty} g_{i+1}(n)/g_i(n) = 0$, $\lim_{n\to\infty} g_i(n+1)/g_i(n) = b_i \neq 1$ *and* $\forall j \neq i, b_j \neq b_i$, *and* $\exists k > 1$ *such that* $\lim_{n\to\infty} g_2(n)/(g_1(n)g_{k+2}(n)) = c \neq 0$.
If the E-algorithm is applied to

$$E_0^{(n)} = S_n + \Delta S_n \cdot \frac{g_1^{(n+1)}}{\Delta g_1^{(n)}}, \quad n = 0, 1, \ldots$$

$$g_{0,i}^{(n)} = \Delta S_n \cdot \frac{g_{i+1}(n)}{\Delta g_1(n)} + S_{n+1} \cdot \frac{\Delta g_{i+1}(n)}{\Delta g_1(n)}, \qquad i = 1, \ldots, k \text{ and } n = 0, 1, \ldots$$

then $\left(E_0^{(n)}\right)$ *converges to* S *faster than* (S_n) *and for* $i = 0, \ldots, k-1$, $\left(E_{i+1}^{(n)}\right)_n$ *converges to* S *faster than* $\left(E_i^{(n)}\right)_n$.

When the sequences $(g_i(n))$ are logarithmic (that is $b_i = 1$) other procedures described in Matos [310, 313] can be used. They have applications in convergence acceleration of some continued fractions.

Of course it was impossible, in this section, to be complete and to present all the known procedures leading to an asymptotic expansion of the error. Our aim was to show that such a construction was possible (but difficult) in some cases, to give a flavour of some procedures and to refer the interested reader to the literature on the subject.

3.4 Construction of extrapolation processes

In the preceding section we saw that if a quite complete information is known (namely an asymptotic expansion related in some way to the error $S_n - S$) then acceleration is possible in some cases.

In this section we shall take a complementary point of view. We shall assume that only a partial information on the error is known and try to construct appropriate extrapolation processes. The details can be found in Brezinski and Redivo Zaglia [97].

Let (S_n) be a sequence converging to S such that for all n

$$S_n - S = a_n g_n$$

where (a_n) and (g_n) are sequences which are known or not and which can depend on the sequence (S_n) itself. Since (S_n) converges to S, $(a_n g_n)$ must tend to zero. In the sequel we shall assume that (g_n) converges to zero, an assumption which does not restrict the generality, but we shall make no assumption on convergence of (a_n). This formalism covers many interesting particular cases

$S_n - S = O(g_n)$	which corresponds to $\lvert a_n \rvert \leq M, \forall n$.
$S_n - S = o(g_n)$	which corresponds to $\lim_{n\to\infty} a_n = 0$.
$S_n - S = a_1 g_1(n) + a_2 g_2(n) + \cdots$	where the g_i's form an asymptotic sequence. Thus $S_n - S = a_n g_n$ with $g_n = g_1(n)$ and $a_n = a_1 + a_2 g_2(n)/g_1(n) + \cdots = a_1 + \varepsilon_n$ with $\lim_{n\to\infty} \varepsilon_n = 0$.
$S_n = c_0 f_0 + \cdots + c_n f_n$	which corresponds to $g_n = -c_{n+1}$ or $g_n = -f_{n+1}$ or $g_n = -c_{n+1} f_{n+1}$.

The first situation was studied by Szabo [422] who used an extrapolation algorithm identical to one of those given below.

We shall consider two cases: (a_n) unknown and (g_n) known, and (a_n) and (g_n) unknown.

Let us begin by the case where (a_n) is unknown and (g_n) known. We shall study three different extrapolation processes according to the information we have on (a_n).

We assume that an operator P is known such that $\forall n$, $P(a_n) = 0$ and such that $P(au_n + b) = a\,P(u_n) + b$, $\forall a, b$ and (u_n).

Thus

$$P(a_n) = 0 = P\left((S_n - S)/g_n\right) = P(S_n/g_n) - S\,P(1/g_n)$$

and it follows that $\forall n$

$$S = P(S_n/g_n)/P(1/g_n).$$

This approach is due to Weniger [458] and it was already described in section 1.3. Of course it leads to the exact result for all n but needs a quite complete information on (a_n).

Let us now require less information. We assume that an approximation (b_n) of (a_n) is known and we consider the sequence (T_n) given

by

$$T_n = S_n - b_n g_n, \qquad n = 0, 1, \ldots$$

We have the

Theorem 3.18

A necessary and sufficient condition that (T_n) *converges to* S *faster than* (S_n) *is that* $\lim_{n\to\infty} b_n/a_n = 1$.

In order to illustrate this theorem let us consider the sequence $S_n = n \cdot \sin(1/n), n = 1, 2, \ldots$ which converges to $S = 1$. We have

$$S_n - S = \frac{1}{n^2} \cdot \left(-\frac{1}{6} + \frac{1}{5!n^2} - \cdots\right)$$

and we shall take $g_n = 1/n^2$ and $b_n = -1/6 + e_n$ where (e_n) is an arbitrary sequence converging to zero. The assumption of the preceding theorem is satisfied and we obtain the following results (number of exact digits)

n	S_n	T_n $e_n = 0.9^n$	T_n $e_n = 1/n$	T_n $e_n = 1/n^2$
1	0.80	0.05	0.00	0.00
10	2.78	2.46	3.00	4.00
20	3.38	3.52	3.90	5.21
30	3.73	4.33	4.43	5.91
40	3.98	5.03	4.81	6.41
50	4.18	5.69	5.10	6.80
60	4.33	6.30	5.33	7.12

In that case we have

$$\frac{T_n - S}{S_n - S} = 1 - b_n \cdot \frac{g_n}{S_n - S}.$$

Thus an idea is to take $b_n = \Delta S_n / \Delta g_n$ and we obtain the process

$$T_n = S_n - \frac{\Delta S_n}{\Delta g_n} \cdot g_n, \qquad n = 0, 1, \ldots$$

which corresponds to an extrapolation at zero by a function $f(x) = ax + b$ satisfying $S_n = f(g_n)$ and $S_{n+1} = f(g_{n+1})$. (T_n) is also identical to

the sequence $\left(E_1^{(n)}\right)$ obtained by the E-algorithm with $g_1(n) = g_n$ (see section 2.1) or to the Θ-procedure (see section 2.9).

For this transformation we have several results according to the asymptotic behaviour of (g_n).

Theorem 3.19

If $\exists \alpha < 1 < \beta, \exists N$ *such that* $\forall n \geq N, g_{n+1}/g_n \notin [\alpha, \beta]$ *and if* $\exists 0 < m \leq M$ *such that* $\forall n \geq N$, $m \leq |g_{n+1}/g_n| \leq M$ *then a necessary and sufficient condition that* (T_n) *converges to* S *faster than* (S_n) *and* (S_{n+1}) *is that* $\lim_{n\to\infty} a_{n+1}/a_n = 1$.

The conditions of this theorem are, in particular, satisfied in the following situations

i) $\lim_{n\to\infty} a_n = a \neq 0, \pm\infty$

ii) $\lim_{n\to\infty} \Delta a_n = 0$ and $\exists m, \exists N$ such that $\forall n \geq N, m \leq |a_n|$

iii) $\lim_{n\to\infty} |a_n| = \infty$ and $\exists M, \exists N$ such that $\forall n \geq N, |\Delta a_n| \leq M$.

Let us consider the sequence

$$S_n = \sum_{i=1}^{n} \frac{i(i+1)}{2} \cdot x^{i-1}, \qquad n = 1, 2, \ldots, \quad |x| < 1.$$

We have

$$S_n = \frac{1}{(1-x)^3} - x^n \left[\frac{1}{(1-x)^3} + \frac{n(n+3) + xn(n+1)}{2(1-x)^2} \right].$$

Taking $g_n = x^n$ it is easy to see that the assumptions of theorem 3.19 are satisfied and we obtain for $x = 0.5$

n	S_n	T_n
1	0.06	0.30
4	0.46	0.96
9	1.49	2.23
14	2.68	3.59
19	3.96	4.98
24	5.28	6.40
29	6.64	7.83

Theorem 3.20

Let us assume that $\lim_{n\to\infty} g_{n+1}/g_n = 0$.

i) If $\exists M, \exists N$ *such that* $\forall n \geq N, |a_{n+1}/a_n| \leq M$ *then* (T_n) *converges to* S *faster than* (S_n).

ii) If $\exists m, \exists M, \exists N$ *such that* $\forall n \geq N, m \leq |a_n|$ *and* $|\Delta a_n| \leq M$ *then* (T_n) *converges to* S *faster than* (S_n).

iii) A necessary and sufficient condition that (T_n) *converges to* S *faster than* (S_{n+1}) *is that* $\lim_{n\to\infty} a_{n+1}/a_n = 1$.

Let us consider the sequence

$$S_n = 1 + \frac{t}{1!} + \cdots + \frac{t^n}{n!}$$

which converges to $S = e^t$. If $t > 0$ then $g_n = t^{n+1}/(n+1)!$ and $a_n = -e^{t\theta_n}$ with $0 < \theta_n < 1$ and $\lim_{n\to\infty} \theta_n = 0$. The condition *iii)* of the preceding theorem is satisfied and we obtain for $t = 1.5$

n	S_n	T_n
1	0.11	0.25
4	1.18	2.80
7	3.03	5.30
10	5.39	8.09
13	8.11	11.14
16	11.11	14.38
19	14.38	15.40

Of course the condition *iii)* is more difficult to check than the two others but it leads to better acceleration properties.

Theorem 3.21

Let us assume that $\lim_{n\to\infty} g_{n+1}/g_n = 1$.

i) A necessary and sufficient condition that (T_n) *converges to* S *faster than* (S_n) *is that* $\lim_{n\to\infty} \dfrac{1 - a_{n+1}/a_n}{1 - g_n/g_{n+1}} = 0$.

ii) A necessary and sufficient condition that (T_n) *converges to* S *faster then* (S_{n+1}) *is that* $\lim_{n\to\infty} \dfrac{1 - a_n/a_{n+1}}{1 - g_{n+1}/g_n} = 0$.

Let us take again $S_n = n \cdot \sin(1/n)$ and $g_n = n^{-2}$. We have $\dfrac{1 - a_n/a_{n+1}}{1 - g_{n+1}/g_n} \sim \dfrac{6}{5!} n^{-2}$ which shows that the condition *ii)* is satisfied and we obtain

n	S_n	T_n
1	0.80	2.69
4	1.98	4.68
9	2.69	5.99
14	3.07	6.72
19	3.34	7.24
24	3.54	7.64
29	3.70	7.96

Let us now come to the case where (a_n) and (g_n) are both unknown. We shall begin by the particular case where $g_n = S_{n-1} - S$ and we assume that an approximation (b_n) of (a_n) is known.

We consider the process given by

$$T_n = \frac{S_n - b_n S_{n-1}}{1 - b_n}, \qquad n = 1, 2, \ldots$$

We have the

Theorem 3.22

A necessary and sufficient condition that (T_n) converges to S faster than (S_{n-1}) is that $\lim_{n\to\infty} (a_n - 1)/(b_n - 1) = 1$.

The condition of this theorem is, in particular, satisfied in the following situations

i) $\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n = a \neq 1$

ii) $\lim_{n\to\infty} (a_n - b_n) = 0$ and $\exists \alpha < 1 < \beta, \exists N$ such that $\forall n \geq N, b_n \notin [\alpha, \beta]$.

Let us consider the sequence (S_n) given by

$$\frac{S_n - S}{S_{n-1} - S} = \frac{1}{2} \sin(an + b/n), \qquad n = 1, 2, \ldots$$

We have $g_n = S_{n-1} - S$ and $a_n = \frac{1}{2}\sin(an + b/n)$. Choosing $b_n = \frac{1}{2}\sin an$ we have

$$a_n - b_n = \cos\frac{2an + b/n}{2} \cdot \sin\frac{b}{2n}.$$

Thus the conditions *ii)* hold and we obtain with $S_0 = 0$, $S = 1$, $a = 2$ and $b = 1$

n	S_n	T_n
1	0.00	0.15
6	3.86	5.09
10	6.03	7.51
14	8.42	9.81
18	10.87	13.60
22	12.71	14.35

Let us now consider the general case where (a_n) and (g_n) are arbitrary unknown sequences. Let (h_n) be an approximation of (g_n). We consider the transformation given by

$$T_n = S_n - \frac{\Delta S_n}{\Delta h_n} \cdot h_n, \qquad n = 0, 1, \ldots$$

We have the following results

Theorem 3.23

Let us assume that $\exists \alpha < 1 < \beta, \exists N$ *such that* $\forall n \geq N, h_{n+1}/h_n \notin [\alpha, \beta]$. *If* $\lim_{n\to\infty} g_n/h_n = a \neq 0, \pm\infty$ *then a necessary and sufficient condition that* (T_n) *converges to* S *faster than* (S_n) *and* (S_{n+1}) *is that* $\lim_{n\to\infty} a_{n+1}/a_n = 1$.

These conditions are satisfied by the sequence (S_n) given by $S_n = S + a_n g_n$ for $n = 0, 1, \ldots$ with $g_n = \lambda^{n+1}(0.5 + 1/\ln(n+2))$, $a_n = 2 + (n+1)^{-1}$ and $|\lambda| < 1$. With $h_n = \lambda^{n+1}, \lambda = 0.95$ and $S = 1$ we obtain

n	S_n	T_n
15	0.11	0.77
30	0.49	1.59
45	0.84	2.20
60	1.18	2.73
75	1.53	3.21
90	1.87	3.66
105	2.20	4.09

Theorem 3.24

Let us assume that $\exists \alpha < 1 < \beta, \exists N$ *such that* $\forall n \geq N, h_{n+1}/h_n \notin [\alpha, \beta]$. *A necessary and sufficient condition that* (T_n) *converges to* S *faster than* (S_n) *and* (S_{n+1}) *is that* $\lim_{n\to\infty} \frac{a_{n+1}}{a_n} \cdot \frac{g_{n+1}}{g_n} \cdot \frac{h_n}{h_{n+1}} = 1$.

Let us consider the sequence (S_n) given by $S_n = S + a_n g_n$ for $n = 0, 1, \ldots$ with $g_n = \lambda^{n+1}/(n+1)$, $a_n = 2 + (n+1)^{-1}$ and $|\lambda| < 1$.

The conditions of the preceding theorem are satisfied by the choice $h_n = \lambda^{n+1}$ and we obtain with $\lambda = 0.95$ and $S = 1$

n	S_n	T_n
10	0.97	0.64
20	1.48	1.47
30	1.87	2.04
40	2.22	2.52
50	2.54	2.93
60	2.84	3.31

Finally we have the

Theorem 3.25

Let us assume that $\lim_{n\to\infty} h_{n+1}/h_n = 1$.

i) A necessary and sufficient condition that (T_n) *converges to* S *faster than* (S_n) *is that*

$$\lim_{n\to\infty} \left(1 - \frac{a_{n+1}}{a_n} \cdot \frac{g_{n+1}}{g_n} \cdot \frac{h_n}{h_{n+1}}\right) \cdot \left(1 - \frac{h_n}{h_{n+1}}\right)^{-1} = 0.$$

ii) A necessary and sufficient condition that (T_n) *converges to* S *faster than* (S_{n+1}) *is that*

$$\lim_{n\to\infty}\left(\frac{a_n}{a_{n+1}}\cdot\frac{g_n}{g_{n+1}}\cdot\frac{h_{n+1}}{h_n}-1\right)\cdot\left(\frac{h_{n+1}}{h_n}-1\right)^{-1}=0.$$

We consider the sequence given by $S_n = S + a/(n+b)$ for $n = 0, 1, \ldots$ and $b \geq 1$. If $h_n = (n+1)^{-1}$ the conditions of the preceding theorem are satisfied and we obtain with $S = 1, a = 2$ and $b = 4$

n	S_n	T_n
10	0.85	1.48
20	1.08	1.96
30	1.23	2.27
40	1.34	2.50
50	1.43	2.68
60	1.51	2.83

Sometimes the processes described above can be iterated.
Some of the transformations studied in this section have the form

$$T_n = \frac{S_{n+1} - b_n S_n}{1 - b_n}, \qquad n = 0, 1, \ldots$$

If (S_n) is a monotone sequence it can be interesting to know conditions on (b_n) such that (T_n) is also monotone either in the same or in the opposite direction as (S_n). Such conditions were studied by Opfer [342]. In particular when the monotonicity is reversed we obtain intervals containing the limit S of the sequence (S_n). For example we have the

Theorem 3.26

If (S_n) *is strictly monotone, if* (b_n) *is strictly monotone in the same direction as* (S_n) *and if* $\forall n, b_n \neq 1$ *and* $\lim_{n\to\infty} b_n = b \neq 1$, *then a necessary and sufficient condition that* (T_n) *be strictly monotone in the opposite direction as* (S_n) *is that* $\forall n, 0 < b_n < 1$ *and* $\Delta S_{n+1}/\Delta S_n < b_n(1 - b_{n+1})/(1 - b_n)$. *A sufficient condition is that* $\forall n, \Delta S_{n+1}/\Delta S_n < b_n < 1$.

In this case S belongs to the intervals with endpoints S_n and T_n. This method for controlling the error can be compared with that of section 3.8.

3.5 Extraction procedures

Logarithmic sequences, that is such that $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=1$, are difficult to accelerate. In section 1.10 we saw that they form a remanent set and thus a universal algorithm for their acceleration cannot exist. This is also true for some subsets of logarithmic sequences.

On the other hand, linear sequences, that is such that $\lim_{n\to\infty}(S_{n+1}-S)/(S_n-S)=a\neq 1$, are easy to accelerate, for example, by Aitken's Δ^2 process.

Thus the idea came to extract, from a given logarithmic sequence, a subsequence converging linearly and then to accelerate it. However such a procedure is not always possible since, otherwise, it would be equivalent to the possibility of accelerating all the logarithmic sequences thus violating the result on remanence (theorem 1.12). Such an extraction procedure is only possible for some subsets of logarithmic sequences as we shall see below. But before explaining this point, we have to look if the acceleration of a subsequence implies the acceleration of the initial sequence.

Let $\left(S_{a(n)}\right)$ be a subsequence of (S_n). Let $T:\left(S_{a(n)}\right)\longmapsto(T_n)$ be a sequence transformation such that the computation of T_n uses only $S_{a(j)}$ for $j\leq n$ and such that

$$\lim_{n\to\infty}(T_n-S)\Big/\left(S_{a(n)}-S\right)=0.$$

We would like to know if the acceleration of the subsequence $(S_{a(n)})$ implies the acceleration of the initial sequence (S_n). For that let us define the sequence (t_n) by

$$t_n=\begin{cases}S_n & \text{if } n<a(0)\\ T_i & \text{if } a(i)\leq n<a(i+1)\,.\end{cases}$$

We see that t_n only depends on S_j for $j\leq n$. We have the following result

Theorem 3.27

If (S_n) is a monotonous sequence of real numbers, if $a:\mathbb{N}\longmapsto\mathbb{N}$ is strictly increasing and if $\exists\alpha>0$ such that

$$\liminf_{n\to\infty}\left(S_{a(n+1)}-S\right)\Big/\left(S_{a(n)}-S\right)=\alpha$$

then

$$\lim_{n\to\infty}(t_n - S)/(S_n - S) = 0.$$

We shall now describe some extraction procedures and give the corresponding theoretical results.

Let $(g(n))$ be a given monotone logarithmic sequence converging to zero, and let us consider the set S_g of sequences such that $\exists\lambda \neq 0$, $\lim_{n\to\infty}(S_n - S)/g(n) = \lambda$. We have the

Theorem 3.28

Let $(a(n))$ be defined by $a(0) = 0$ and $\forall n \geq 1$

$$a(n) = \min\{i > a(n-1) : |g(i)| < \varrho^n |g(0)|\}$$

where $\varrho \in]0,1[$.
If $(S_n) \in \mathrm{S}_g$ then $(S_{a(n)})$ converges linearly. Moreover

$$\lim_{n\to\infty} g(a(n+1))/g(a(n)) = \lim_{n\to\infty} \left(S_{a(n+1)} - S\right) / \left(S_{a(n)} - S\right) = \varrho.$$

Thus the set S_g can be accelerated by Aitken's Δ^2 process and the sequence (t_n) constructed as above converges to S faster then (S_n). As proved by Keagy and Ford [258] the set S_g in the previous theorem cannot be extended by replacing the condition by a less restrictive one.

Under some additional assumptions, this procedure can be iterated as explained by Brezinski, Delahaye and Germain-Bonne [93].

The result of theorem 3.28 only holds if $(g(n))$ is a monotone logarithmic sequence. From any logarithmic sequence converging to zero it is possible to extract a monotone sequence as explained by Brezinski [81]. The extraction procedure is based on the remark that since the ratio $g(n+1)/g(n)$ tends to 1 then there exists an index N such that $\forall n \geq N, g(n+1)/g(n) > 0$ which means that $g(n)$ and $g(n+1)$ have the same sign. If $g(n) > 0$ a decreasing subsequence can be extracted while, if $g(n) < 0$, an increasing subsequence can be extracted. The extraction algorithm is as follows.

1. Set $N = 0$.

2. Set $n = 0$ and $b(0) = N$.
   If $g(b(0)) > 0$ set $k = 0$. Otherwise set $k = 1$.

3. Set $j = 1$. If $k = 0$ go to 4, if $k = 1$ go to 5.

4. If $g(b(n) + j) < 0$ set $N = b(n) + j$, $k = 1$ and go to 2.
   If $g(b(n)+j) > 0$ and if $g(b(n)+j) < g(b(n))$ set $b(n+1) = b(n)+j$, replace n by $n + 1$ and go to 3.
   If $g(b(n) + j) > 0$ and if $g(b(n) + j) \geq g(b(n))$ replace j by $j + 1$ and go to 4.

5. If $g(b(n) + j) > 0$ set $N = b(n) + j$, $k = 0$ and go to 2.
   If $g(b(n)+j) < 0$ and if $g(b(n)) < g(b(n)+j)$ set $b(n+1) = b(n)+j$, replace n by $n + 1$ and go to 3.
   If $g(b(n) + j) < 0$ and if $g(b(n)) \geq g(b(n) + j)$ replace j by $j + 1$ and go to 5.

Point 4 corresponds to the case where $\forall n \geq N, g(n) > 0$ while point 5 treats the other case. It must be noticed that, in practice, it is impossible to know whether or not the index N such that $\forall n \geq N, g(n+1)/g(n) > 0$ has been attained.

Let us now consider the following extraction procedure

step 0. Let $\lambda \in]0, 1[, a(0) = 0, n = 0$.
step 1. Replace n by $n + 1$.
Set $a(n) = a(n - 1) + 1$.
Go to step 3.
step 2. Replace $a(n)$ by $a(n) + 1$.
Go to step 3.
step 3. If $\left|S_{a(n)+1} - S_{a(n)}\right| \leq \lambda^n |S_1 - S_0|$ go to step 1, otherwise go to step 2.

We have the

Theorem 3.29

If (S_n) is a monotone logarithmic sequence such that $\lim_{n\to\infty} \Delta S_{n+1}/\Delta S_n = 1$, the subsequence extracted by the preceding procedure satisfies

$$\lim_{n\to\infty} \left(S_{a(n+1)+1} - S_{a(n+1)}\right) \Big/ \left(S_{a(n)+1} - S_{a(n)}\right) = \lambda.$$

It was not possible to prove that this implies

$$\lim_{n\to\infty} \left(S_{a(n+1)} - S\right) \Big/ \left(S_{a(n)} - S\right) = \lambda$$

in the general case although it is true for all the examples considered by Smith and Ford [413].

Let us give some numerical examples to illustrate this procedure.

We set

$$r_n = \left(S_{a(n+1)+1} - S_{a(n+1)}\right) \Big/ \left(S_{a(n)+1} - S_{a(n)}\right) .$$

(T_n) is the sequence obtained by applying Aitken's Δ^2 process to the subsequence $\left(S_{a(n)}\right)$. Since $t_n = T_i$ for $a(i) \leq n < a(i+1)$ we shall compare T_i with $S_{a(n+1)-1}$.

- Example 1

$$S_n = \sum_{i=0}^{n} \frac{1}{(i+1)^2}, \quad S = 1.6449340668, \quad \lambda = 0.3$$

n	$a(n)$	r_n	$S_{a(n+1)-1}$	T_n
0	0	0.082		
1	5	0.290		
2	11	0.319	1.5980	1.578
3	22	0.315	1.6196	1.6336
4	39	0.299	1.6313	1.644966
5	73	0.304	1.6375	1.644835
6	134	0.303	1.6409	1.6445422
7	245	0.302	1.6424	1.6449386

- Example 2

$$S_0 = 1, \quad S_{n+1} = S_n\,(1 - S_n/2)\,, \quad S = 0\,, \quad \lambda = 0.3$$

n	$a(n)$	r_n	$S_{a(n+1)-1}$	T_n
0	0	0.067		
1	4	0.401		
2	8	0.286	0.0916	0.150
3	18	0.304	0.0494	−0.224
4	36	0.305	0.0270	0.00655
5	69	0.306	0.0149	0.000244
6	129	0.301	0.00812	0.000118

Other extraction procedures can be found in Germain-Bonne and Litovsky [185].

3.6 Automatic selection

We are now faced with a great number of methods and procedures for accelerating the convergence. Each of them works in specific cases that is if the sequence to be accelerated satisfies some assumptions. However, in practical situations it is often difficult (and in some cases, impossible) to check if these assumptions are satisfied or not and thus the user is in a quite uncomfortable situation. Thus the idea came to use simultaneously several sequence transformations $T_i : (S_n) \longmapsto \left(T_i^{(n)}\right)$, $i = 1, \ldots, k$. Then at each step n one can either choose one of the answers according to some test or to combine all these answers together. The first procedure, due to Delahaye [136], is called automatic selection and it will be studied in this section. The second procedure, due to Brezinski [70], leads to the so-called composite sequence transformations which form the subject of the next section.

Let us explain in details the idea of automatic selection. We use simultaneously several sequence transformations T_i. For each value of the index n, we thus obtain the answers $T_1^{(n)}, T_2^{(n)}, \ldots, T_k^{(n)}$. For each transformation $i = 1, \ldots, k$ and each index n, we denote by $R_i^{(n)}$ an assertion, depending on i and n, which can be true or false. Then we define the following count coefficients

$$ {}^0r_i^{(n)} = \begin{cases} 0 & \text{if } R_i^{(n)} \text{ is false} \\ 1 & \text{if } R_i^{(n)} \text{ is true} \end{cases} $$

$$ {}^1r_i^{(n)} = \text{card}\left\{q \in \{0, \ldots, n\} \text{ such that } R_i^{(q)} \text{ is true}\right\} $$

$$ {}^2r_i^{(n)} = \begin{cases} 0 & \text{if } R_i^{(n)} \text{ is false} \\ \max\left\{q \in \{0, \ldots, n\} \text{ such that } R_i^{(n)}, \ldots, R_i^{(n-q+1)} \text{ are true}\right\} \\ & \text{if } R_i^{(n)} \text{ is true.} \end{cases} $$

Let $r_i^{(n)}$ be any one of the previous count coefficients. Then at each step n, let $i(n)$ be the smallest index such that

$$ r_{i(n)}^{(n)} = \max_{i \leq j \leq k} r_j^{(n)}. $$

We set $T_n = T_{i(n)}^{(n)}$. Thus among all the answers $T_1^{(n)}, T_2^{(n)}, \ldots, T_k^{(n)}$, the answer $T_{i(n)}^{(n)}$ has been automatically selected.

For $R_i^{(n)}$, we can take one of the following assertions

$$^1R_i^{(n)} : T_i^{(n)} = T_i^{(n-1)}$$

or

$$^2R_i^{(n)} : \left|T_i^{(n)} - T_i^{(n-1)}\right| = \min_{1\leq j\leq k} \left|T_j^{(n)} - T_j^{(n-1)}\right|.$$

Before giving theoretical results on the preceding automatic selection procedures (based on the 3 count coefficients and the 2 assertions given) we need some definitions.

We say that the transformation T_i is exact for the set S of sequences if $\forall(S_n) \in \mathrm{S}, \exists N, \forall n \geq N, T_i^{(n)} = \lim_{p\to\infty} S_p$.

We say that the transformation T_i is semi-regular for the set S of convergent sequences if $\forall(S_n) \in \mathrm{S}$ such that $\exists N$ with $\forall n \geq N, T_i^{(n)} = T_i^{(n+1)}$ then $\exists M, \forall n \geq M, T_i^{(n)} = \lim_{p\to\infty} S_p$.

We say that T_i is accelerative for S if $\forall(S_n) \in \mathrm{S}$, $\lim_{n\to\infty} (T_i^{(n)} - S)/(S_n - S) = \lim_{n\to\infty} (T_i^{(n+1)} - T_i^{(n)})/(S_{n+1} - S_n) = 0$.

We say that T_i is fair for S if either T_i is accelerative for S or $\forall(S_n) \in \mathrm{S}, \exists \varepsilon > 0, \exists N$ such that $\forall n \geq N, \left|(T_i^{(n+1)} - T_i^{(n)})/(S_{n+1} - S_n)\right| \geq \varepsilon$.

We shall also denote by $T^{p,q}$ the transformation $(S_n) \longmapsto (T_n)$, obtained by using the count coefficients $^pr_i^{(n)}$ for $p = 0, 1$ or 2 and the assertions $^qR_i^{(n)}$ for $q = 1$ or 2.

We have the following results

Theorem 3.30

Let $\mathrm{S}_1, \ldots, \mathrm{S}_k$, *be* k *sets of convergent sequences and let* $\mathrm{S} = \bigcup_{i=1}^{k} \mathrm{S}_i$.

If $\forall i, T_i$ *is exact for* S_i *and semi-regular for* S *then the transformations* $T^{1,1}, T^{2,1}, T^{1,2}$ *and* $T^{2,2}$ *are exact for* S.

Thus this result shows that the transformations $T^{p,q}$ for $p, q = 1$ or 2 are, under some additional assumptions of semi-regularity, exact for the union of the kernels of $T_1, \ldots, T_k$ and, thus, that the best possible selection has been made.

Concerning acceleration we have the following result

Theorem 3.31

If $\forall i, T_i$ *is accelerative for* S_i *and fair for* S, *then the transformations* $T^{0,2}, T^{1,2}$ *and* $T^{2,2}$ *accelerate* S.

This result shows that the transformations $T^{p,2}$ for $p = 0, 1$ or 2 select among the transformations $T_1, \ldots, T_k$ a transformation accelerating the convergence and thus the selection procedure is optimal in that respect.

These results prove that an automatic selection procedure can lead to a new transformation combining, under some assumptions, the best properties of all the transformations used in its construction. It must be understood, however, that such a new transformation is also limited by theorem 1.12 on remanence and that, even by using such a selection procedure, a remanent set of sequences could never be accelerated. One can also imagine a selection procedure between selection procedures. Another selection procedure will be presented in section 3.8.

Let us give a numerical example. We used the transformation $T^{1,2}$ defined above since it is the only one to satisfy simultaneously the conditions of theorems 3.30 and 3.31 but we slightly modify the selection process in order to obtain a classification of all the transformations at each step n. For that we order them such that

$$\left|\Delta T_{q_1}^{(n)}\right| \leq \left|\Delta T_{q_2}^{(n)}\right| \leq \cdots \leq \left|\Delta T_{q_k}^{(n)}\right|$$

and then we increase by k the count coefficient of the transformation q_1, by $k-1$ that of the transformation $q_2, \ldots$, by 1 that of the transformation q_k that is

$$r_{q_i}^{(n)} = r_{q_i}^{(n-1)} + k + 1 - i, \quad i = 1, \ldots, k$$

$$\text{with} \quad r_i^{(0)} = 0 \qquad \text{for } i = 1, \ldots, k.$$

Then the transformations are classified from the best one to the worst one according to

$$r_{p_1}^{(n)} \geq r_{p_2}^{(n)} \geq \cdots \geq r_{p_k}^{(n)}$$

the index p_1 corresponding to $i(n)$ as obtained above.

The transformations used were

T_1 : ε-algorithm
T_2 : ϱ-algorithm with $x_n = n$
T_3 : Θ-algorithm
T_4 : Richardson process with $x_n = \Delta S_n$
T_5 : First generalization of the ε-algorithm with $x_n = 1 + (n+1)^{-1}$
T_6 : Second generalization of the ε-algorithm with $x_n = 1 + (n+1)^{-1}$
T_7 : Iterated Δ^2 process
T_8 : Overholt process
T_9 : Levin's t-transform

We consider the sequence

$$S_n = \frac{1}{n} \cdot \sin an\,, \qquad n = 1, 2, \ldots$$

which converges to zero. The ratio S_{n+1}/S_n has no limit if a is a non rational multiple of π. Moreover, as pointed out by Saff [385], in that case, $\forall r$ there exists an infinite subsequence (n_k) such that the ratio S_{n_k+1}/S_{n_k} tends to r. For $a = 3$ we obtained the following results (number of exact digits and $p_1, \ldots, p_k$)

n	S_n	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
4	0.87	3.62 1	0.86 9	5.62 8	1.85 4	1.34 7	1.34 3	3.62 2	1.93 5	2.61 6
5	0.89	6.50 1	0.87 9	5.46 7	2.78 3	1.34 8	1.63 5	7.02 2	2.97 4	3.67 6
6	0.90	6.50 1	0.88 9	5.34 6	1.60 5	1.65 8	1.86 7	6.96 2	1.96 3	4.58 4
7	0.92	9.39 2	0.88 9	7.79 4	2.16 7	1.49 8	2.15 6	9.16 1	2.56 5	5.52 3
8	0.95	9.39 1	0.90 9	7.96 3	1.33 7	1.90 8	2.41 6	9.21 2	1.92 5	5.73 4
9	0.97	12.28 1	0.90 9	8.09 3	1.89 7	1.61 8	2.69 6	11.17 2	2.69 5	6.06 4
10	1.01	12.28 1	0.92 9	10.97 3	1.34 7	2.11 8	2.97 5	10.91 2	2.45 6	6.33 4

n	S_n	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
11	1.04	15.18	0.93	11.74	1.87	1.72	3.25	12.18	2.94	6.58
		1	9	3	7	8	5	2	6	4
12	1.08	15.18	0.95	11.33	1.55	2.28	3.54	12.18	2.78	6.78
		1	9	3	7	8	5	2	6	4
13	1.13	17.73	0.96	11.42	2.15	1.82	3.82	13.62	3.50	6.95
		1	9	3	7	8	5	2	6	4

These results were obtained by using the last result delivered by the corresponding subroutines. The line giving the order obtained by the procedure means that, for $n = 8$ for example, the best transformation was T_1 (that is the ε-algorithm), the second best was T_7 (that is the iterated Δ^2 process) and the last one T_2 (that is the ϱ-algorithm).

As seen from these results the selection procedure found the correct order of the transformations for $n \geq 10$, but the first four best transformations were obtained for $n \geq 8$.

In section 3.7 we shall see how to use the preceding classification procedure with composite transformations.

The subroutine SELECT performs an automatic selection between several transformations.

To end this section let us present a new selection procedure based on statistical ideas, but for which no theoretical results have been proved so far. It is based on the multiple correlation coefficient whose properties are fully described in Brezinski [89]. Let us first recall some definitions. If x and y are random variables we shall denote by $E(x)$ and $E(y)$ (or by $\bar{x}$ and $\bar{y}$) their expectations (mean values) and we shall set

$$cov(xy) = E\left((x - \bar{x}) \cdot (y - \bar{y})\right) = E(xy) - \bar{x}\bar{y}$$

$$var\, x = cov(xx) = E\left((x - \bar{x})^2\right) = E(x^2) - \bar{x}^2.$$

Let $y, x_1, \ldots, x_k$ be random variables. We define the multiple correlation coefficient of y and $x_1, \ldots, x_k$ by

$$\varrho_k^2 = \frac{(a_k, C_k^{-1} a_k)}{var\, y}$$

where

$$a_k = (cov(yx_1), \ldots, cov(yx_k))^T$$

$$C_k = \begin{pmatrix} var\, x_1 & cov(x_1x_2) & \cdots & cov(x_1x_k) \\ cov(x_2x_1) & var\, x_2 & \cdots & cov(x_2x_k) \\ \vdots & \vdots & & \vdots \\ cov(x_kx_1) & cov(x_kx_2) & \cdots & var\, x_k \end{pmatrix}.$$

We have $0 \leq \varrho_k \leq 1$ and

$$(a_k, C_k^{-1}a_k) = -\frac{\begin{vmatrix} 0 & cov(yx_1) & \cdots & cov(yx_k) \\ cov(x_1y) & cov(x_1x_1) & \cdots & cov(x_1x_k) \\ \vdots & \vdots & & \vdots \\ cov(x_ky) & cov(x_kx_1) & \cdots & cov(x_kx_k) \end{vmatrix}}{|C_k|}.$$

As its name indicates, ϱ_k measures the correlation between y and $x_1, \ldots, x_k$ and it can be proved that if $y = a + b_1x_1 + \cdots + b_kx_k$ where $a, b_1, \ldots, b_k$ are constants, then $\varrho_k = 1$.
ϱ_k can be recursively computed as follows

- Set $y_1 = x_1$, $x_1^* = y_1/\sqrt{var\, y_1}$.
- For $i = 2, \ldots, k$ compute $y_i = x_i - \sum_{j=1}^{i-1} cov(x_ix_j^*)x_j^*$, $x_i^* = y_i/\sqrt{var\, y_i}$
- Then $\varrho_k^2 = \sum_{i=1}^{k} |cov(yx_i^*)|^2 / var\, y$.

The relation $y = a + b_1x_1 + \cdots + b_kx_k$ is very similar to the form of the sequences belonging to the kernel of the E-algorithm since it is the set of sequences such that $\forall n, S_n = S + a_1g_1(n) + \cdots + a_kg_k(n)$. If a sequence has this form then $\forall n, E_k^{(n)} = S$. Thus the idea came to use the multiple correlation coefficient to know whether or not a given sequence (S_n) is close to the form $S_n = S + a_1g_1(n) + \cdots + a_kg_k(n)$. For that we consider the S_n's and the $g_i(n)$'s as realizations of random variables. Since there are infinitely many such realizations, the multiple correlation coefficient cannot be computed exactly but only estimated by taking for an arbitrary sequence (u_n).

$$E((u_n)) = \frac{1}{m+1} \cdot \sum_{i=0}^{m} u_{n+i}.$$

Of course we must take $m > k$ since otherwise ϱ_k would be zero. The estimation of ϱ_k thus obtained, depends on n and m but if (S_n) belongs to the kernel of $E_k : (S_n) \longmapsto \left(E_k^{(n)}\right)_n$ then $\forall n$ and $\forall m > k$ we shall have $\varrho_k = 1$.

An automatic selection procedure for the E-algorithm can be built upon the multiple correlation coefficient as follows

1. Let k and $m > k$ be fixed integers.
2. Make several choices for the k auxiliary sequences in the E-algorithm: $(g_1^i, \ldots, g_k^i)$ for $i = 1, \ldots, p$.
3. For a given n, compute the multiple correlation coefficients ϱ_k^1, $\ldots$, ϱ_k^p corresponding to the various sets of auxiliary sequences as explained above.
4. Let $i(n)$ be the smallest index such that $\varrho_k^{i(n)} = \max\limits_{1 \le j \le p} \varrho_k^j$.
   Use the E-algorithm with $g_{0,i}^{(j)} = g_i^{i(n)}(j)$ and compute $E_k^{(n)}$.
   Set $T_k^{(n)} = E_k^{(n)}$.
   Add 1 to n and go to 3.

This selection procedures deserves further studies. Since

$$\varrho_{k+1}^2 = \varrho_k^2 + \left|cov(yx_{k+1}^*)\right|^2 \Big/ var\ y$$

the procedure can also be used to find the best possible value of k.

In section 2.7 we saw that the three Levin's transforms, u, t and v correspond to three different choices of the auxiliary sequence $(g(n))$. Thus let us apply this selection procedure to them with $k = 1$ since it is a particular case of the E-algorithm.

We consider the partial sums of the series $\ln(1+x)$ for $x = -0.9$ which converges to -2.302585092994046. We obtain the following numbers of exact decimal digits (the u-transform was used with $b = 0$) for $T_n^{(0)}$

n	u	t	v
5	1.87	2.15	2.93
10	3.88	4.10	4.60
15	5.49	6.09	6.38
20	8.25	7.98	8.48

The correlation coefficients, computed with $m = 5$ and $m = 8$ respectively, are

n	u	t	v
5	−0.9932	−0.99938	−0.99946
	−0.9887	−0.99894	−0.99915
10	−0.9981	−0.99983	−0.99990
	−0.9964	−0.99969	−0.99982
15	−0.9990	−0.999936	−0.999970
	−0.9982	−0.999877	−0.999946
20	−0.99947	−0.999970	−0.999989
	−0.99889	−0.999941	−0.999979

Thus we see that the correlation coefficient provides a good selection test between the transformations used.

An application of this procedure for finding the best value of m for the T_{+m} transformation was already described in section 2.11.

Selection procedures can be easily implemented on parallel computers.

3.7 Composite sequence transformations

As in the previous section, let us assume that k sequence transformations are used simultaneously, $T_i : (S_n) \longmapsto \left(T_i^{(n)}\right)$ for $i = 1, \ldots, k$. At each step n, instead of choosing one result among $T_1^{(n)}, T_2^{(n)}, \ldots, T_k^{(n)}$, we shall combine them. More precisely we shall define the transformation $T : (S_n) \longmapsto (T_n)$ by

$$T_n = \sum_{i=1}^{k} a_i^{(n)} T_i^{(n)}, \quad n = 0, 1, \ldots$$

T is called a composite sequence transformation of rank k. They were introduced by Brezinski [70] who showed their interest by some examples. If the coefficients $a_i^{(n)}$ are chosen such that $\forall n, \exists i(n)$ with $a_{i(n)}^{(n)} = 1$ and $a_j^{(n)} = 0$ for $j \neq i(n)$ then we recover the automatic selection procedure as defined in the previous section.

Let $\mathcal{K}_i$ be the kernel of the transformation T_i that is the set of sequences for which $\forall n, T_i^{(n)} = S$. Of course, if $(S_n) \in \bigcap_{i=1}^{k} \mathcal{K}_i$ then $\forall i$ and

$\forall n, T_i^{(n)}$ will be equal to S. Thus it will be interesting for T to share the same property which leads to, $\forall n$

$$T_n = S = S \cdot \sum_{i=1}^{k} a_i^{(n)}.$$

Thus we shall have $\forall n, \sum_{i=1}^{k} a_i^{(n)} = 1$, a property assumed to hold in the sequel.

Let us first study composite sequence transformations of rank 2. To shorten the notations, let us write a_n instead of $a_2^{(n)}$ and $a_1^{(n)} = 1 - a_n$. Thus we have

$$T_n = (1 - a_n)T_1^{(n)} + a_n T_2^{(n)}, \qquad n = 0, 1, \ldots$$

It will be interesting that the kernel of T contains the kernels of T_1 and T_2. This was, in fact, a property satisfied by the transformations built by automatic selection (see theorem 3.30 of the previous section). Thus we must have $\forall n, a_n = 0$ if $(S_n) \in \mathcal{K}_1$ and $\forall n, a_n = 1$ if $(S_n) \in \mathcal{K}_2$. This leads us to the choice

$$\begin{aligned} a_n &= -\frac{\Delta T_1^{(n)}}{\Delta T_2^{(n)} - \Delta T_1^{(n)}} \qquad \text{if } \Delta T_2^{(n)} \neq \Delta T_1^{(n)} \\ a_n &= 0 \qquad \text{otherwise} \end{aligned}$$

and we obtain in the first case

$$T_n = T_1^{(n)} - \frac{\Delta T_1^{(n)}}{\Delta T_2^{(n)} - \Delta T_1^{(n)}} \cdot \left(T_2^{(n)} - T_1^{(n)}\right)$$

and $T_n = T_1^{(n)}$ in the second one.
T_n can be written as

$$T_n = \frac{\begin{vmatrix} T_1^{(n)} & T_2^{(n)} \\ \Delta T_1^{(n)} & \Delta T_2^{(n)} \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ \Delta T_1^{(n)} & \Delta T_2^{(n)} \end{vmatrix}}.$$

T_n can also be obtained by applying the procedure Θ to $T_2^{(n)}$ if we write it as $T_2^{(n)} = T_1^{(n)} + D_n$ with $D_n = T_2^{(n)} - T_1^{(n)}$. Thus, results on this composite sequence transformation T immediately follow from the results given in section 2.9 on the procedure Θ and we have the

Theorem 3.32

Let us assume that $\forall n, T_1^{(n)} \neq T_2^{(n)}$. *A necessary and sufficient condition that* $\forall n, T_n = S$ *is that* $\exists a_1$ *and* a_2 *with* $a_1 + a_2 \neq 0$ *such that* $\forall n, a_1 \left(T_1^{(n)} - S\right) + a_2 \left(T_2^{(n)} - S\right) = 0$.

When $T_1^{(n)} = S_n$ and $T_2^{(n)} = S_{n+1}$, T is Aitken's Δ^2 process and the condition of theorem 3.32 is exactly the condition defining the kernel of this process.

For the convergence we have the

Theorem 3.33

If $\lim_{n\to\infty} T_1^{(n)} = \lim_{n\to\infty} T_2^{(n)}$ *and if* $\exists \alpha < 1 < \beta$ *such that* $\forall n \geq N$, $\Delta T_2^{(n)} / \Delta T_1^{(n)} \notin [\alpha, \beta]$, *then* $\lim_{n\to\infty} T_n = S$.

We have now several type of acceleration results depending whether or not one of the transformations T_i accelerates the convergence.

Theorem 3.34

If $\lim_{n\to\infty} \left(T_2^{(n)} - S\right) / \left(T_1^{(n)} - S\right) = a \neq 1$ *and if* $\exists \alpha < 1 < \beta$ *such that* $\forall n \geq N$, $\left(T_1^{(n+1)} - S\right) / \left(T_1^{(n)} - S\right) \notin [\alpha, \beta]$, *then* $\lim_{n\to\infty} (T_n - S) / \left(T_1^{(n)} - S\right) = 0$. *Moreover if* $a \neq 0$, $\lim_{n\to\infty} (T_n - S) / \left(T_2^{(n)} - S\right) = 0$.

This result was improved by Sadok [381] who considered the case where $\left(T_1^{(n)}\right)$ is logarithmic. He gave the

Theorem 3.35

If $\left(T_2^{(n)} - S\right) / \left(T_1^{(n)} - S\right) = a + a_n$ *with* $a \neq 1$ *and* $\lim_{n\to\infty} a_n = 0$, *if* $\left(T_1^{(n+1)} - S\right) / \left(T_1^{(n)} - S\right) = 1 + b_n$ *with* $\lim_{n\to\infty} b_n = 0$, *and if* $\lim_{n\to\infty} \Delta a_n / b_n = 0$ *then* $T_n - S = o\left(T_1^{(n)} - S\right)$. *Moreover if* $a \neq 0, T_n - S = o\left(T_2^{(n)} - S\right)$.

Of course similar results hold by exchanging T_1 and T_2.

Theorem 3.36

Let (S_n) be a non-logarithmic sequence such that $\lim_{n\to\infty} \left(T_1^{(n)} - S\right) / (S_n - S) = a$ *and* $\lim_{n\to\infty} \left(T_2^{(n)} - S\right) / (S_n - S) = b \neq a$. *Then* $\lim_{n\to\infty} (T_n - S)/(S_n - S) = 0$.

This result still holds if a or b is equal to zero. Finally we have the following result in which the condition $b \neq a$ is replaced by another one.

Theorem 3.37

If $\lim_{n\to\infty} \left(T_1^{(n)} - S\right) / (S_n - S) = \lim_{n\to\infty} \left(T_2^{(n)} - S\right) / (S_n - S) = a$ *and if* $\exists \alpha < 1 < \beta$ *such that* $\forall n \geq N, \Delta T_2^{(n)} / \Delta T_1^{(n)} \notin [\alpha, \beta]$ *then* $\lim_{n\to\infty} (T_n - S)/(S_n - S) = a$.

This result also holds when $a = 0$.

These results were extended by Draux [145]. Let r_i be the largest positive real number such that

$$\lim_{n\to\infty} \frac{T_i^{(n)} - S}{|S_n - S|^{r_i}} = C_i$$

with $|C_i| < +\infty$ for $i = 1$ and 2. He proved the

Theorem 3.38

If $r_1 = r_2$, if C_1 and C_2 are different from zero and if $C_1 \neq C_2$ then $\lim_{n\to\infty} (T_n - S) / |S_n - S|^{r_1} = 0$.

We also have the

Theorem 3.39

If $\lim_{n\to\infty} (S_{n+1} - S) / (S_n - S) = b$ *and if one of the following assumptions is satisfied*

a) $b \in]-1, 1[$

b) $b = \pm 1$ *and* $(S_{n+1} - S) / (S_n - S) = b(1 + \alpha_n)$ *with* $\lim_{n\to\infty} \alpha_n = 0$, $(T_i^{(n)} - S) / |S_n - S|^{r_i} = C_i + \beta_i^{(n)}$ *with* $\lim_{n\to\infty} \beta_i^{(n)} = 0$, $|C_i| < +\infty$ *and* $\lim_{n\to\infty} \Delta\beta_i^{(n)} / \alpha_n = 0$ *for* $i = 1$ *and* 2

then

i) if $r_1 > r_2$ *and* $C_2 \neq 0$ *we have* $\lim_{n\to\infty} a_n = 0$.

ii) if $r_1 = r_2$ *and* $C_2 \neq 0$ *we have* $\lim_{n\to\infty} \Delta T_1^{(n)} / \Delta T_2^{(n)} = C_1/C_2$. *Moreover if* $C_1 = 0$ *we have* $\lim_{n\to\infty} a_n = 0$.

iii) if $r_1 > r_2$, $C_2 = 0$, $\lim_{n\to\infty} |S_n - S|^{r_1} / \left(T_2^{(n)} - S\right) = 0$ *and* $\lim_{n\to\infty} \left(T_2^{(n+1)} - S\right) / \left(T_2^{(n)} - S\right) = c \in [-1, 1[$ *we have* $\lim_{n\to\infty} a_n = 0$.

If $b = 1$ the additional assumption of *b*) are not needed.

This theorem shows that the composite transformation T chooses automatically the best transformation among T_1 and T_2 except in the case *ii)*. However in that case the assumptions of theorem 3.38 are satisfied and convergence acceleration is obtained. In that case, if $C_2 = 1$ it can be interesting to take a transformation T_1 slowing down the convergence and such that $r_1 = 1$ and $C_1 > 1$ as shown by Draux [145].

We shall now define composite sequence transformations of an arbitrary rank k. The results given below improve those of Brezinski [70].

Let us consider the previous determinantal formula for T_n and just increase the determinants as

$$ {}_iT_k^{(n)} = \frac{\begin{vmatrix} T_i^{(n)} & \cdots & T_{i+k}^{(n)} \\ \Delta T_i^{(n)} & \cdots & \Delta T_{i+k}^{(n)} \\ \vdots & & \vdots \\ \Delta T_i^{(n+k-1)} & \cdots & \Delta T_{i+k}^{(n+k-1)} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \Delta T_i^{(n)} & \cdots & \Delta T_{i+k}^{(n)} \\ \vdots & & \vdots \\ \Delta T_i^{(n+k-1)} & \cdots & \Delta T_{i+k}^{(n+k-1)} \end{vmatrix}} $$

where Δ operates on the upper indexes and $i \geq 1$. k must be strictly less than the number of transformations used.

If $T_i^{(n)} = S_{n+i}$ we recover Shanks' transformation: ${}_iT_k^{(j)} = e_k(S_n) = \varepsilon_{2k}^{(n)}$ where $i + j = n$.

The numbers ${}_iT_k^{(n)}$ can be recursively computed by the E-algorithm

for a fixed value of n. If we set

$$\begin{aligned} E_0^{(i)} &= T_{i+1}^{(n)}, & i &= 0,1,\dots \\ \text{and } g_{0,j}^{(i)} &= \Delta T_{i+1}^{(n+j-1)}, & i &= 0,1,\dots, \; j = 1,2,\dots \end{aligned}$$

then we shall obtain $E_k^{(i)} = {}_{i+1}T_k^{(n)}$ for all k and i.

Thus we obtain the finite triangular array from $T_1, \dots, T_m$

$$\begin{array}{ccccccc} {}_1T_0^{(n)} & & & & & & \\ {}_2T_0^{(n)} & {}_1T_1^{(n)} & & & & & \\ {}_3T_0^{(n)} & {}_2T_1^{(n)} & {}_1T_2^{(n)} & & & & \\ {}_4T_0^{(n)} & {}_3T_1^{(n)} & {}_2T_2^{(n)} & {}_1T_3^{(n)} & & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & & \\ {}_mT_0^{(n)} & {}_{m-1}T_1^{(n)} & {}_{m-2}T_2^{(n)} & {}_{m-3}T_2^{(n)} & \cdots & {}_1T_{m-1}^{(n)} \end{array}$$

If the index i is fixed, the numbers ${}_iT_k^{(n)}$ can again be obtained from the E-algorithm. If we set

$$\begin{aligned} E_0^{(n)} &= T_i^{(n)}, & n &= 0,1,\dots \\ \text{and } g_{0,j}^{(n)} &= T_{i+j}^{(n)} - T_{i+j-1}^{(n)}, & n &= 0,1,\dots, \; j = 1,2,\dots \end{aligned}$$

then we shall obtain $E_k^{(n)} = {}_iT_k^{(n)}$ for all k and n. Thus we obtain the infinite array

$$\begin{array}{cccccc} {}_iT_0^{(0)} & & & & & \\ {}_iT_0^{(1)} & {}_iT_1^{(0)} & & & & \\ {}_iT_0^{(2)} & {}_iT_1^{(1)} & {}_iT_2^{(0)} & & & \\ {}_iT_0^{(3)} & {}_iT_1^{(2)} & {}_iT_2^{(1)} & {}_iT_3^{(0)} & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \end{array}$$

Since the determinantal formula for ${}_iT_k^{(n)}$ is similar to that of the E-algorithm, a similar result holds for the kernel. It is the

Theorem 3.40

A necessary and sufficient condition that $\forall n$, ${}_iT_k^{(n)} = S$ *is that there exist* $a_0, a_1, \dots, a_k$ *with* $a_0 + \cdots + a_k \neq 0$ *such that* $\forall n$

$$a_0 \left(T_i^{(n)} - S\right) + \cdots + a_k \left(T_{i+k}^{(n)} - S\right) = 0.$$

A necessary and sufficient condition that $\forall i$, ${}_iT_k^{(n)} = S$ *is that there exist* $b_0, b_1, \ldots, b_k$ *with* $b_0 + \cdots + b_k \neq 0$ *such that* $\forall i$

$$b_0 \left(T_i^{(n)} - S\right) + \cdots + b_k \left(T_i^{(n+k)} - S\right) = 0.$$

Of course $a_0, \ldots, a_k$ can depend on i and k and $b_0, \ldots, b_k$ on n and k. If the a_j's are independent of i and if the first condition holds for all i, then the result is also true for all i. If the b_j's are independent of n and if the second condition holds for all n, then the result is also true for all n.

These results generalize the relation defining the kernel of Shanks' transformation (theorem 2.18) which is recovered for $T_i^{(n)} = S_{n+i}$. It also shows that the kernel of the transformation ${}_iT_k : (S_n) \longmapsto \left({}_iT_k^{(n)}\right)_n$ contains the kernels of the transformations $T_i, \ldots, T_{i+k}$. The convergence and acceleration theorems for the E-algorithm (theorems 2.8, 2.10 and 2.11) immediately give results for ${}_iT_k$ but their conditions are usually too difficult to check to be of any practical interest.

In applications we are dealing with a finite number of transformations $T_1, \ldots, T_m$. When applying the E-algorithm to the computation of ${}_iT_k^{(n)}$ for a fixed value of n, $i + k$ must be strictly less than m. Such a restriction still exists when computing ${}_iT_k^{(n)}$ by the E-algorithm for a fixed value of i, but n can be arbitrarly large. Thus our second implementation of composite transformations (that is for i fixed) seems to be more interesting than the first one since the number of terms is not limited and sequences are really constructed instead of only a finite triangular part of the array.

Let us give a numerical example and take the same transformations as in section 3.6 and the same sequence. When $i = 1$ and when the seven best transformations are used in the last order given in section 3.6 the E-algorithm gives (number of exact digits) for $n \geq 9$

n	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$
9	12.28						
10	12.28	12.84					
11	15.18	13.56	13.56				
12	15.18	17.81	13.85	14.27			
13	17.73	16.58	16.88	15.35	14.19		
14	18.52	17.81	16.22	16.73	14.46	13.68	
15	17.56	17.59	17.68	16.44	16.46	13.78	14.51

We made a systematical use of such a procedure, starting again from the beginning the use of the E-algorithm for implementing the composite transformation each time the order of the transformations obtained by the automatic selection changes.

As it can be seen from the previous results, the use of a composite transformation can improve the result of the best transformation. Then the precision downgrades due to the introduction of transformations with fewer exact digits and to the rounding errors in the E-algorithm.

Let us now consider the sequence $S_{n+1} = e^{-S_n}$ with $S_0 = 1$. The four best transformations as found by the selection procedure described in the preceding section are Richardson's process, Overholt's process, the iterated Δ^2 process and the ε-algorithm. Using them, in this order, for constructing the corresponding composite transformation with $i = 1$, we obtain

n	$k=0$	$k=1$	$k=2$	$k=3$
4	4.51			
5	4.90	4.86		
6	7.04	6.67	5.44	
7	9.08	8.74	7.93	6.80
8	11.26	11.59	9.78	9.92
9	14.75	13.55	13.16	9.87
10	15.71	15.71	15.01	14.71
11	15.41	15.41	15.41	15.41
12	99.00	99.00	99.00	99.00

When 99.00 is printed it means that full accuracy was achieved. The column $k = 0$ contains the results given by the best transformation among the four ones involved in the process.

Let us mention that, instead of using a rank k composite sequence transformation, it is possible to iterate rank 2 composite transformations as follows

- Construct the composite transformation (T'_n) from $\left(T_1^{(n)}\right)$ and $\left(T_2^{(n)}\right)$.
- Construct the composite transformation (T''_n) from (T'_n) and $\left(T_3^{(n)}\right)$.
- Construct the composite transformation (T'''_n) from (T''_n) and $\left(T_4^{(n)}\right)$ and so on.

Such a procedure remains to be studied and compared with rank k composite transformations.
The subroutine COMPOS performs the above composite sequence transformation.

Composite transformations can be easily implemented on parallel computers.

3.8 Error control

From the user's point of view it is not enough to know that the sequence (T_n) he obtained converges faster than the initial sequence (S_n). It would be better to have an estimation of the error or, still better, to know a sequence of intervals containing the limit S of the sequence (S_n) and whose endpoints also converge to S faster than (S_n). This section is devoted to such questions and their answers. We shall set, as usual

$$T_n = S_n + D_n\,, \qquad n = 0, 1, \ldots$$

and we shall define the sequence (e_n) by

$$T_n - S = e_n D_n\,, \qquad n = 0, 1, \ldots$$

Let b be a real nonzero number. We define $(T_n(b))$ as

$$T_n(b) = S_n + (1 - b)D_n\,, \qquad n = 0, 1, \ldots$$

and the interval $I_n(b)$ by

$$I_n(b) = [\min(T_n(b), T_n(-b)), \max(T_n(b), T_n(-b))]\,.$$

We have

$$[T_n(b) - S] \cdot [T_n(-b) - S] = (e_n^2 - b^2)D_n^2$$

and thus we obviously have the

Theorem 3.41

A necessary and sufficient condition that $\forall b \neq 0, \exists N$ *such that* $\forall n \geq N, S \in I_n(b)$ *is that* $\exists N$ *such that* $\forall n \geq N, |e_n| \leq |b|$.

S is the limit of (S_n) and the index N depends on b. If (T_n) converges to S faster than (S_n) then (e_n) converges to zero and we obtain the

Theorem 3.42

If (T_n) converges to S faster than (S_n) then $\forall b \neq 0, \exists N$ such that $\forall n \geq N, S \in I_n(b)$.

Thus it is quite easy to construct a sequence of intervals containing asymptotically. If $|a| < |b|$ then $I_n(a)$ is strictly contained in $I_n(b)$ and N, which is a function of b, tends to infinity when $|b|$ tends to zero.

As seen from the numerical examples given by Brezinski [67], the intervals $I_n(b)$ can be quite large and thus the error control is not quite satisfactory. This drawback is due to the fact that $(T_n(b))$ and $(T_n(-b))$ do not converge to S faster than (S_n). This can be avoided by using a sequence (b_n) instead of a constant value b. In this case, theorem 3.41 still holds if b is replaced by (b_n). If (T_n) converges to S faster than (S_n) this condition is no more automatically satisfied and a supplementary assumption has also to be introduced in order that the endpoints of $I_n(b_n)$ converge to S faster than (S_n), thus leading to the

Theorem 3.43

If (T_n) converges to S faster than (S_n) then a necessary and sufficient condition that $(T_n(b_n))$ and $(T_n(-b_n))$ converge to S faster than (S_n) is that (b_n) tends to zero.

A necessary and sufficient condition that $\exists N$ such that $\forall n \geq N, S \in I_n(b_n)$ is that $\exists N$ such that $\forall n \geq N, |e_n| \leq |b_n|$.

This second condition shows that (b_n) cannot be arbitrarily chosen and this remark leads to a second drawback since we want, of course, both conditions to be satisfied simultaneously. This is possible in particular cases but, most of the time, additional assumptions have to be introduced and their verification needs a sharp knowledge of the asymptotic behaviour of (S_n) and (T_n).

The best possible choice for b_n is $b_n = e_n$ since it leads to $T_n(e_n) = S$. Of course this choice is impossible in practice since e_n is unknown. However a choice such that (b_n/e_n) tends to 1 is expected to provide a good alternative. Since $e_n = (T_n - S)/D_n$, one can think of taking $b_n = \Delta T_n/\Delta D_n$ but this choice presents some other drawback. For that reason we shall take

$$b_n = \frac{\Delta T_n}{\Delta D_n} \cdot \frac{1}{1 + D_{n+1}/D_n}, \quad n = 0, 1, \ldots$$

and we have the

Theorem 3.44

If $\lim_{n\to\infty} D_{n+1}/D_n = \lim_{n\to\infty} e_{n+1}/e_n \neq \pm 1$ *then* $\lim_{n\to\infty} b_n/e_n = 1$.

But this is not sufficient for our purpose since we need that the second condition of theorem 3.43 be satisfied to have $S \in I_n(b_n)$, and additional conditions have to be introduced. Thus, instead of considering the endpoints $T_n(b_n)$ and $T_n(-b_n)$ we shall use the endpoints $T_n\,(b_n(1+\varepsilon))$ and $T_n\,(b_n(1-\varepsilon))$. Let $J_n(\varepsilon)$ be the interval with these endpoints. We have

$$\begin{aligned} T_n\,(b_n(1+\varepsilon)) &= T_n + (1+\varepsilon)(-b_n D_n) \\ T_n\,(b_n(1-\varepsilon)) &= T_n + (1-\varepsilon)(-b_n D_n) \end{aligned}$$

which shows that theorem 3.41 applies after replacing S_n by T_n, D_n by $-b_n D_n$, b by ε and e_n by e'_n such that $T_n - S = (1 - e'_n)b_n D_n$. But $T_n - S = e_n D_n$ and thus $e'_n = 1 - e_n/b_n$. Finally we obtain the

Theorem 3.45

A necessary and sufficient condition that $\forall \varepsilon \neq 0, \exists N$ *such that* $\forall n \geq N, S \in J_n(\varepsilon)$ *is that* $\exists N$ *such that* $\forall n \geq N, |1 - e_n/b_n| \leq |\varepsilon|$.

With the preceding choice of (b_n) and under the assumptions of theorem 3.44, this condition is satisfied. Thus $S \in J_n(\varepsilon)$ for all $n \geq N$ and the sequences $(T_n\,(b_n(1+\varepsilon)))$ and $(T_n\,(b_n(1-\varepsilon)))$ converge to S faster than (S_n) and also faster than (T_n).

Let us illustrate this procedure by a numerical example. We shall take for T_n Aitken's Δ^2 process and the sequence $S_{n+1} = \exp(-S_n)$ with $S_0 = 1$ which converges to 0.5671432904097838.

We obtain the following results

$$\varepsilon = 0.5$$

$n = 0$	0.5615158786644487	0.5753226908852316
$n = 5$	0.5671182846229792	0.5671677134657973
$n = 10$	0.5671432057743461	0.5671433751618701
$n = 15$	0.5671432901180999	0.5671432907014442
$n = 20$	0.5671432904087798	0.5671432904107880
$n = 25$	0.5671432904097804	0.5671432904097873

$$\varepsilon = 0.01$$

$n = 0$	0.5682812166526323	0.5685573528970480
$n = 4$	0.5671433316609800	0.5671463696971711
$n = 5$	0.5671425047559601	0.5671434933328164
$n = 10$	0.5671432887742328	0.5671432921619833
$n = 15$	0.5671432904039386	0.5671432904156055
$n = 20$	0.5671432904097639	0.5671432904098039
$n = 25$	0.5671432904097837	0.5671432904097840

We see that when ε is smaller, so are the intervals but, on the other hand, for $\varepsilon = 0.01$ S belongs to the intervals only for $n \geq 5$ while for $\varepsilon = 0.5$ this is true from $n = 0$.

It must be noticed that this procedure uses in fact two sequence transformations namely $(S_n) \longmapsto (T_n) \longmapsto (T_n(b_n))$ and that (T_n) converges to S faster than (S_n) and $(T_n(b_n))$ faster than (T_n). We say that these two transformations accelerate the convergence of (S_n) in cascade. Any two arbitrary transformations accelerating the convergence of (S_n) in cascade lead to the same result. Let us set $T_n(b)$ and $I_n(b)$ as before. We assume that a second sequence transformation $V : (S_n)$ or $(T_n) \longmapsto (V_n)$ is known and we set

$$V_n(b) = V_n - b(V_n - T_n) \qquad n = 0, 1, \ldots$$

$$J_n(b) = [\min(V_n(b), V_n(-b)), \max(V_n(b), V_n(-b))] \qquad n = 0, 1, \ldots$$

We have the

Theorem 3.46

If $\lim_{n\to\infty} (T_n - S)/(S_n - S) = \lim_{n\to\infty} (V_n - S)/(T_n - S) = 0$ *then* $\forall b \neq 0, \exists N$ *such that* $\forall n \geq N, S \in J_n(b) \subseteq I_n(b)$. *Moreover* $(V_n(b))$ *and* $(V_n(-b))$ *converge to* S *faster that* (S_n).

It must be remarked that with only one transformation, (T_n) converges faster than (S_n) but not $(T_n(b))$. With two transformations accelerating (S_n) in cascade, (V_n) converges faster than (T_n) but not faster than $(V_n(b))$. Thus, in order to control the error, more than acceleration is needed if we want the endpoints of the intervals to converge faster than the initial sequence. This is not a very surprising fact since, as explained in sections 1.4 and 3.1, convergence acceleration is equivalent to estimating the error perfectly.

It is possible to built an automatic selection procedure (see section 3.6) on the error control. The idea is as follows: let us use simultaneously several pairs of sequence transformations $(T_i, V_i) : (S_n) \longmapsto \left(T_i^{(n)}, V_i^{(n)}\right)$ for $i = 1, \ldots, k$. Thus at each step n and for each pair of transformations, we have an interval

$$J_i^{(n)}(b) = \left[\min\left(V_i^{(n)}(b), V_i^{(n)}(-b)\right), \max\left(V_i^{(n)}(b), V_i^{(n)}(-b)\right)\right]$$

with

$$V_i^{(n)}(b) = V_i^{(n)} - b\left(V_i^{(n)} - T_i^{(n)}\right).$$

Let $L_i^{(n)}(b)$ be the length of the interval $J_i^{(n)}(b)$, that is

$$L_i^{(n)}(b) = 2 \cdot |b| \cdot \left|V_i^{(n)} - T_i^{(n)}\right|.$$

Let $i(n)$ be the smallest index such that

$$r_{i(n)}^{(n)} = \max_{1 \le i \le k} r_i^{(n)}$$

where $r_i^{(n)} = \text{card}\left\{q \in \{0, \ldots, n\} \text{ such that } L_i^{(q)}(b) = \min_{1 \le j \le k} L_j^{(q)}(b)\right\}$.

We set

$$T_n = T_{i(n)}^{(n)} \text{ and } V_n = V_{i(n)}^{(n)}.$$

$r_i^{(n)}$ is the number of times that the interval $J_i^{(n)}(b)$ was the smallest one from the beginning. Thus, at the step n, we select the pair of transformations for which this number is maximum. Under some assumptions this selection procedure exhibits interesting properties.

Let L_i be the intersection of the kernels of T_i and V_i and let L be the union of the L_i's. We assume that for all i, L_i is not empty.

We say that a pair of transformations (T_i, V_i) possesses the property A for a set S of sequences if $\forall (S_n) \in \mathsf{S}$ such that $\exists N, \forall n \ge N, T_n = V_n$ then it implies that $\forall n \ge N, T_n = V_n = S$.

We say that a pair of transformations (T_i, V_i) possesses the property B for a set S of sequences in $\forall (S_n) \in \mathsf{S}$ either $\lim_{n \to \infty} \left(T_i^{(n)} - S\right) / (S_n - S) = \lim_{n \to \infty} \left(V_i^{(n)} - S\right) / \left(T_i^{(n)} - S\right) = 0$ or $\exists \varepsilon > 0, \exists N$ such that $\forall n \ge N$, $\left|T_i^{(n)} - S\right| / |S_n - S| \ge \varepsilon$.

We say that a pair of transformations (T_i, V_i) possesses the property C for a set S of sequences if $\forall (S_n) \in \mathbf{S}, \exists \alpha < 1 < \beta, \exists N$ such that $\forall n \geq N, \left(V_i^{(n)} - S\right) / \left(T_i^{(n)} - S\right) \notin [\alpha, \beta]$.

We shall denote by $\mathbf{S}_i$ the set of convergent sequences such that $\left(T_i^{(n)}\right)_n$ converges faster than (S_n) and $\left(V_i^{(n)}\right)_n$ faster than $\left(T_i^{(n)}\right)_n$.

Let $\mathbf{S}$ be the union of $\mathbf{S}_1, \ldots \mathbf{S}_k$.

If (T_n) and (V_n) are the sequences obtained by the above selection procedure, let us set $\forall n$

$$V_n(b) = V_n - b(V_n - T_n)$$

and $J_n(b) = [\min(V_n(b), V_n(-b)), \max(V_n(b), V_n(-b))]$. We have the

Theorem 3.47

- If $\forall i$, the pair (T_i, V_i) possesses the property A for L, *then* $\forall (S_n) \in$ L, $\exists N$ *such that* $\forall n \geq N, T_n = V_n = S$.

- If $\forall i$, the pair (T_i, V_i) possesses the properties B and C for S, *then* $\forall (S_n) \in \mathbf{S}$, $\lim_{n \to \infty} (T_n - S)/(S_n - S) = \lim_{n \to \infty} (V_n - S)/(T_n - S) = 0$.

$(V_n(b))$ and $(V_n(-b))$ converge faster than (S_n) and $\forall b \neq 0, \exists N$ such that $\forall n \geq N, S \in J_n(b)$.

To end this section let us point out a fundamental practical point. Under some assumptions, the theorems given above say that for all n greater than N, S belongs to some interval. However such an N is not known without adding supplementary assumptions. Such an N has been attained if the interval at the step $n+1$ is contained in the interval obtained at the step n, whatever $n \geq N$ be. This is a good test for having attained this N. Let us give some results obtained by Bellalij [21] in which N is known.

Theorem 3.48

If $T_n = S_{n+1}$ (that is if $D_n = \Delta S_n$), if (S_n) is strictly monotone and if $\exists N$ such that $\forall n \geq N, 0 < \Delta S_{n+2}/\Delta S_{n+1} < \Delta S_{n+1}/\Delta S_n < 1$ then $\forall n \geq N, S \in I_n(b_n)$ with $b_n = -\Delta S_{n+1}/\Delta^2 S_n$.

The assumption on the monotonicity of $(\Delta S_{n+1}/\Delta S_n)$ can be suppressed by using two sequences (b_n) and (c_n) instead of (b_n) alone.
We set, $\forall n$

$$I_n(b, c) = [\min(T_n(b), T_n(c)), \max(T_n(b), T_n(c))],$$

and we have the

Theorem 3.49

If (S_n) is strictly monotone, if $T_n = S_{n+1}$, if $\forall n, c_n < b_n < 0$ and if $\exists N$ such that $\forall n \geq N$,

$$\frac{b_n}{b_{n+1} - 1} \leq \frac{\Delta S_{n+1}}{\Delta S_n} \leq \frac{c_n}{c_{n+1} - 1}$$

then $\forall n \geq N, I_{n+1}(b_{n+1}, c_{n+1})$ is contained in $I_n(b_n, c_n)$.
Moreover if $\lim_{n\to\infty} b_n \Delta S_n = \lim_{n\to\infty} c_n \Delta S_n = 0$ then $\forall n \geq N, S \in I_n(b_n, c_n)$.

To illustrate this result let us consider the sequence

$$S_n = \frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{n^2}, \qquad n = 1, 2, \ldots$$

which converges to $\pi^2/6 = 1.6449334066848226\ldots$.

The assumptions of the preceding theorem are satisfied with $b_n = -n$ and $c_n = -(n+1)$ and we obtain

$n = 0$	1.5	1.75
$n = 5$	1.634246031746032	1.654654195011338
$n = 10$	1.641365527309791	1.648309971754236
$n = 15$	1.643170062856753	1.646630270469210
$n = 20$	1.643885363063712	1.645951478766192
$n = 25$	1.644239730568865	1.645611472681348
$n = 45$	1.644706114934087	1.645158808460569

When (S_n) is an alternating sequence, we shall separate it into two subsequences

$$u_n = S_{2n} \quad \text{and} \quad v_n = S_{2n+1}, \quad n = 0, 1\ldots$$

and apply Aitken's Δ^2 process to (u_n) and (v_n) separately thus obtaining two sequences denoted by (u'_n) and (v'_n) respectively. We set, $\forall n$

$$J_n = [\min(u'_n, v'_n), \max(u'_n, v'_n)].$$

The following result was proved by Bellalij [21]

Theorem 3.50

If (S_n) is an alternating sequence such that $\forall n, S_n \neq S$ and $\Delta S_{n+1}/\Delta S_n \neq -1$, if (u_n) and (v_n) are monotone, if $\forall n, \Delta u_{n+1}/\Delta u_n < 1$ and $\Delta v_{n+1}/\Delta v_n < 1$ and if $(\Delta u_{n+1}/\Delta u_n)$ and $(\Delta v_{n+1}/\Delta v_n)$ are both either decreasing or increasing then $\forall n, S \in J_n$.

When $(\Delta S_{n+1}/\Delta S_n)$ is a monotone sequence, Hillion [229] proposed other intervals containing the limit. If (ΔS_n) has the form $\Delta S_n = \lambda^n \cdot n^\nu \cdot (a_0 + a_1 n^{-1} + \cdots)$ a process for simultaneously accelerating the convergence and controlling the error was given by Bellalij [22].

Instead of controlling the error (that is computing an interval containing the limit S) one can try to estimate it. We have

$$\frac{T_n - S_n}{S - S_n} = 1 - \frac{T_n - S_n}{S_n - S}.$$

Thus if $|T_n - S|$ is small with respect to $|S_n - S|$ then $T_n - S_n$ is a good estimation of the error $S - S_n$. That is, in particular, the case if (T_n) converges to S faster than (S_n) and we have

$$\lim_{n\to\infty} (T_n - S_n)/(S - S_n) = 1.$$

But, in that case, we also have

$$\lim_{n\to\infty} (T_n - S)/(T_n - S_n) = 0$$

which shows that, for n sufficiently large, $|T_n - S|$ is much smaller than $|T_n - S_n|$. Thus $|T_n - S_n|$ can be taken as an upper bound of $|T_n - S|$.

If we have only

$$\lim_{n\to\infty} (T_n - S)/(S_n - S) = a$$

then it is easy to see that

$$\lim_{n\to\infty} (T_n - S_n)/(S - S_n) = 1 - a$$

$$\lim_{n\to\infty} (T_n - S_n)/(S - T_n) = (1 - a)a^{-1}$$

which shows that $|T_n - S_n|$ can still be used for estimating the errors $|S_n - S|$ and $|T_n - S|$.

If there exist a and b with $-1 < a \leq b < 1$ and N such that $\forall n \geq N$

$$-1 < a \leq \frac{T_n - S}{S_n - S} \leq b < 1$$

then $\forall n \geq N$

$$0 < 1 - b \leq \frac{T_n - S_n}{S - S_n} \leq 1 - a < 2$$

$$-\frac{b}{1-b} \leq \frac{T_n - S}{T_n - S_n} \leq -\frac{a}{1-a} < \frac{1}{2}$$

which again shows than $|T_n - S_n|$ can be used to control the error. The construction of such sequence transformations (called contractive) will be studied in the next section.

If we are given two sequence transformations $T : (S_n) \longmapsto (T_n)$ and $U : (S_n) \longmapsto (U_n)$ they can be compared by similar techniques by replacing S_n by T_n, T_n by U_n and $T_n - S_n$ by $U_n - T_n$ in the previous results.

Special devices for controlling the error and stopping rules based on them where studied by Walz [452] in the case where

$$S_n = S + \sum_{i=1}^{\infty} a_i / 2^{n\varrho_i}, \qquad n = 0, 1, \ldots$$

with $0 < \varrho_1 < \varrho_2 < \cdots$ and the proper modification of Richardson process (that is, in fact, the E-algorithm with $g_i(n) = 2^{n\varrho_i}$) is used. Walz gave a priori bounds which allow to estimate the errors of the extrapolated values before starting the extrapolation process. He also proposed stopping rules for deciding, during the process, if the accuracy desired has been already obtained or not.

Let us also mention that, due to the connection between the ε-algorithm and Padé approximation, the special procedures proposed by Brezinski [76, 83] for estimating the error of Padé approximants, can be used for the ε-algorithm. These procedures require additional terms of the sequence in supplement to those used in the computation of the $\varepsilon_{2k}^{(n)}$'s.

3.9 Contractive sequence transformations

As we saw in the previous sections, accelerating the convergence of a given sequence (or a given set of sequences) can be a difficult task. For a remanent set of sequences it is even impossible. Thus the idea came to ask less and first to find a sequence transformation $T : (S_n) \longmapsto (T_n)$ such that there exists $|a| < 1$ with $\lim_{n\to\infty} (T_n - S)/(S_n - S) = a$. But, in fact, as proved by Litovsky [290], the difficulty is the same since, if such a transformation T is known, then it is possible by the ACCES-algorithm to built another transformation accelerating (S_n) (see section 3.1).

In both cases, what makes the goal too difficult to achieve? It is the fact that a certain ratio must have a limit which is not always the case. Thus we shall suppress this requirement. Let us give an example showing that sequence transformations not having this property can still present a great practical interest.

We consider the sequence (S_n) given by $S_{2n} = (n+1)^{-1}$ and $S_{2n+1} = (-1+(-1)^n \cdot 10^{-6}) \cdot (n+1)^{-1}$ for $n = 0, 1, \ldots$ and the transformation defined by $T_n = (S_{2n} + S_{2n+1})/2, n = 0, 1, \ldots$. We have

$$T_n = (-1)^n \cdot 10^{-6} \cdot (n+1)^{-1}$$

and the ratio T_n/S_{2n} has no limit when n tends to infinity. However $|T_n/S_{2n}| = 10^{-6}$ which shows the drastic improvement brought by the transformation.

Let us begin by some definitions. Let $T : (S_n) \longmapsto (T_n)$ be a sequence transformation such that the computation of T_n needs the knowledge of $S_0, \ldots, S_{k(n)}$.

T is said to be a contraction (or is said to be contractive) on the set S of sequences if $\forall (S_n) \in$ S there exist two constants K_1 and K_2 with $-1 < K_1 \leq K_2 < 1$ and an index N such that $\forall n \geq N, K_1 \leq (T_n - S)/(S_n - S) \leq K_2$.

If $\forall n \geq N, K_1 \leq (T_n - S) / \left(S_{k(n)} - S\right) \leq K_2$, then T is said to be a true contraction on S.

Let us first consider a general quasi-linear transformation of the form

$$T_n = F(S_n, \ldots, S_{n+k}), \qquad n = 0, 1, \ldots$$

where k is a constant and the set $Lin(\alpha, \beta)$ of real convergent sequences such that $\forall n \geq N, 0 \leq \alpha \leq (S_{n+1} - S)/(S_n - S) \leq \beta$. This set is not remanent but it was proved by Delahaye [136] that it is not accelerable by such a transformation. Thus an important question is to know if a contraction on $Lin(\alpha, \beta)$ can exist or not, a possibility not forbidden by its non-accelerability. Brezinski [82] gave the following result

Theorem 3.51

If there exist K_1 and K_2 with $-1 < K_1 \leq K_2 < 1$ such that $\forall x_i \in [\alpha^i, \beta^i]$ for $i = 1 \ldots, k$ we have $K_1 \leq F(1, x_1, \ldots, x_k) \leq K_2$, then T is a contraction on $Lin(\alpha, \beta)$.

If $\forall y_i \in [\beta^{-i}, \alpha^{-i}]$ for $i = 1 \ldots, k$ we have $K_1 \leq F(y_k, \ldots, y_1, 1) \leq K_2$, then T is a true contraction on $Lin(\alpha, \beta)$.

Since Aitken's Δ^2 process in one of the simplest and of the most popular sequence transformations we shall study its contraction properties. It is easy to see that it is a contraction if and only if there exist K_1 and K_2 with $-1 < K_1 \leq K_2 < 1$ and N such that $\forall n \geq N, 0 < 1 - K_2 \leq (1 - r_n)/(1 - \varrho_n) \leq 1 - K_1 < 2$ and a true contraction if and only if $\forall n \geq N, 0 < 1 - K_2 \leq r_n^{-1}(1 - r_n)/(1 - \varrho_n) \leq 1 - K_1 < 2$ where $r_n = (S_{n+1} - S)/(S_n - S)$ and $\varrho_n = \Delta S_{n+1}/\Delta S_n$.

Of course it is difficult in practice to check if these conditions are satisfied or not. This is the reason why we shall restrict ourselves to the following sufficient conditions

Theorem 3.52

Aitken's Δ^2 process is a contraction on the set of convergent sequences such that there exist $\alpha, \beta, \alpha', \beta'$ with $\alpha \leq \beta < 1$, $\alpha' \leq \beta' < 1$ and $2\beta' < 1 + \alpha$ and N with $\forall n \geq N, \alpha \leq r_n \leq \beta$ and $\alpha' \leq \varrho_n \leq \beta'$. In that case $K_1 = (\alpha - \beta')/(1 - \beta')$ and $K_2 = (\beta - \alpha')/(1 - \alpha')$.

Let us mention that if $0 \leq \alpha$ then, as proved by Delahaye [136], the condition on ϱ_n is always satisfied with $\alpha' = \alpha(1 - \beta)/(1 - \alpha)$ and $\beta' = \beta(1 - \alpha)/(1 - \beta)$. In that case the condition $2\beta' < 1 + \alpha$ becomes $3\beta < 1 + \alpha + \alpha\beta$. However it must be noticed that the bounds given by these α' and β' are usually quite large and it is better to use sharper ones if they are known.

Similarly we can prove the

Theorem 3.53

Aitken's Δ^2 process is a true contraction on the set of convergent sequences such that there exist $\alpha, \beta, \alpha', \beta'$ with $0 < \alpha \leq \beta < 1$, $\alpha' \leq \beta' < 1 < \alpha(3 - 2\beta')$ and N with $\forall n \geq N, \alpha \leq r_n \leq \beta$ and $\alpha' \leq \varrho_n \leq \beta'$. In that case $K_1 = (2\alpha - \alpha\beta' - 1)/\alpha(1 - \beta')$ and $K_2 = (2\beta - \alpha'\beta - 1)/\beta(1 - \alpha')$.

Let us notice that a sequence (S_n) can converge without the ratio $(S_{n+1} - S)/(S_n - S)$ being bounded. For example if $S_0 = 1, S_{2n+1} = a_n S_{2n}$ and $S_{2n+2} = b_n S_{2n+1}$ we have $S_{2n+1} = a_0 \cdot \ldots \cdot a_n \cdot b_0 \cdot \ldots \cdot b_{n-1}$ and $S_{2n+2} = a_0 \cdot \ldots \cdot a_n \cdot b_0 \cdot \ldots \cdot b_n$. Thus if $a_n = (-1)^n \cdot 2^n$ and $b_n = 8^{-n}$ then (S_n) tends to zero but $S_{2n+1}/S_{2n} = (-1)^n \cdot 2^n$ and $S_{2n+2}/S_{2n+1} = 8^{-n}$ and the sequence (S_{n+1}/S_n) has three points of accumulation $0, \pm\infty$.

The conditions of the two preceding theorems are only sufficient.

Let us now consider logarithmic sequences. The set LOGSF of logarithmic sequences, that is such that $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = \lim_{n\to\infty}$

$\Delta S_{n+1}/\Delta S_n = 1$, is known to be remanent. Thus a contractive sequence transformation on it cannot exist, and we shall consider its subset LOG2 which can be defined as follows. We write $r_n = 1 - \lambda_n$ where (λ_n) tends to zero and assume that $\forall n, \lambda_n > 0$. For any sequence of LOGSF, Kowalewski [267] proved that $(\lambda_{n+1}/\lambda_n)$ tends to 1. Thus we shall set $\lambda_{n+1}/\lambda_n = 1 - \mu_n$ where (μ_n) tends to zero. LOG2 is the subset of LOGSF such that $\exists N, \forall n \geq N, \mu_n > 0$. We have the

Theorem 3.54

Aitken's Δ^2 process is a contraction and a true contraction on the subset of LOG2 *of sequences such that there exist $\beta \geq 0$ and N with $\forall n \geq N, 0 \leq \mu_n/\lambda_n \leq \beta$.*

It is not known whether or not LOG2 is accelerable.

Let us now study the contraction properties of the procedure Θ. We consider a sequence transformation $T : (S_n) \longmapsto (T_n)$ where the computation of T_n need the knowledge of $S_0, \ldots, S_{n+k}$. We can always write T_n under the forms

$$T_n = S_n + D_n = S_{n+k} + E_n\,, \qquad n = 0, 1, \ldots$$

If we apply the procedure Θ to these two forms we obtain two new sequence transformations given by

$$\Theta_n = S_n - \frac{\Delta S_n}{\Delta D_n} \cdot D_n\,, \qquad n = 0, 1, \ldots$$

$$\Theta'_n = S_{n+k} - \frac{\Delta S_{n+k}}{\Delta E_n} \cdot E_n\,, \qquad n = 0, 1, \ldots$$

We have the

Theorem 3.55

- *Let $\alpha \leq \beta < 1, \alpha' \leq \beta' < 1$ and $2\beta' < 1 + \alpha$. If $\exists N$ such that $\forall n \geq N, \alpha \leq r_n \leq \beta$ and $\alpha' \leq D_{n+1}/D_n \leq \beta'$ then $\forall n \geq N, -1 < K_1 \leq (\Theta_n - S)/(S_n - S) \leq K_2 < 1$ with $K_1 = (\alpha - \beta')/(1 - \beta')$ and $K_2 = (\beta - \alpha')/(1 - \alpha')$.*
- *If $\forall n \geq N, \alpha \leq (T_{n+1} - S)/(T_n - S) \leq \beta$ and $\alpha' \leq D_{n+1}/D_n \leq \beta'$ then $\forall n \geq N, -1 < K_1 \leq (\Theta_n - S)/(T_n - S) \leq K_2 < 1$ with K_1 and K_2 as above.*

- *Moreover if* $0 < \alpha \leq \beta'$ *and* $\alpha' \leq \beta$ *then* $\forall n \geq N, K_1/\alpha \leq (\Theta_n - S)/(T_{n+1} - S) \leq K_2/\alpha$.
- *If* $\forall n \geq N, \alpha \leq r_n \leq \beta$ *and* $\alpha' \leq E_{n+1}/E_n \leq \beta'$ *then* $\forall n \geq N, -1 < K_1 \leq (\Theta'_n - S)/(S_{n+k} - S) \leq K_2 < 1$ *with* K_1 *and* K_2 *as above.*
- *If* $\forall n \geq N, \alpha \leq (T_{n+1} - S)/(T_n - S) \leq \beta$ *and* $\alpha' \leq E_{n+1}/E_n \leq \beta'$ *then* $\forall n \geq N, -1 < K_1 \leq (\Theta'_n - S)/(T_n - S) \leq K_2 < 1$ *with* K_1 *and* K_2 *as above.*
- *Moreover if* $0 < \alpha \leq \beta'$ *and* $\alpha' \leq \beta$ *then* $\forall n \geq N, K_1/\alpha \leq (\Theta'_n - S)/(T_{n+1} - S) \leq K_2/\alpha$.

Obviously the preceding results must be completed by the study of the contraction properties of other transformations and other sets of sequences. Instead of studying the contraction properties of existing transformations, one can think about building contractive sequence transformations. This was the approach followed by Brezinski and Paszkowski [96] who studied linear sequence transformations and chose their coefficients in an optimal way. Let us consider the transformation defined by

$$T_k^{(n)} = a_0 S_n + \cdots + a_k S_{n+k}\,, \qquad n = 0, 1, \ldots$$

with $a_0 + \cdots + a_k = 1$. We have

$$\begin{aligned}
\frac{T_k^{(n)} - S}{S_n - S} &= a_0 + a_1 \cdot \frac{S_{n+1} - S}{S_n - S} + \cdots + a_k \cdot \frac{S_{n+k} - S}{S_n - S} \\
&= a_0 + a_1 \cdot \frac{S_{n+1} - S}{S_n - S} + \cdots + \\
&\qquad a_k \cdot \left(\frac{S_{n+1} - S}{S_n - S} \cdot \frac{S_{n+2} - S}{S_{n+1} - S} \cdot \ldots \cdot \frac{S_{n+k} - S}{S_{n+k-1} - S} \right).
\end{aligned}$$

We shall consider the class of sequences such that there exist α and β with $\alpha \leq \beta$ and N with $\forall n \geq N, \alpha \leq r_n \leq \beta$.

Thus

$$\frac{T_k^{(n)} - S}{S_n - S} = a_0 + a_1 r_n + \cdots + a_k (r_n \cdot \ldots \cdot r_{n+k-1})$$

and it is natural to ask for the polynomial

$$P(x_1, \ldots, x_k) = \sum_{i=0}^{k} a_i x_1 \cdot \ldots \cdot x_i$$

which minimizes

$$\max_{x_1,\ldots,x_k\in[\alpha,\beta]} |P(x_1,\ldots,x_k)|\,.$$

This problem was completely solved by Paszkowski [353]. Its solution depends on the respective values of α and β and different cases have to be consider. They lead to different optimal linear contractive sequence transformations. When applied to a sequence (S_n) such that $\forall n \geq N, \alpha \leq (S_{n+1}-S)/(S_n-S) \leq \beta$ they produce a sequence $\left(T_k^{(n)}\right)$ such that $\forall n \geq N, \left|\left(T_k^{(n)}-S\right)/(S_n-S)\right| \leq M_k < 1$. They are the following

1. $0 \leq \alpha < \beta < 1$

$$\begin{aligned} T_k^{(n)} &= \frac{2S_{n+k} - 2\alpha S_{n+k-1} - \beta^{k-1}(\beta-\alpha)S_n}{2(1-\alpha)-\beta^{k-1}(\beta-\alpha)} \\ M_k &= \frac{\beta^{k-1}(\beta-\alpha)}{2(1-\alpha)-\beta^{k-1}(\beta-\alpha)}. \end{aligned}$$

2. $\alpha \leq 0 < \beta < 1$ and $\alpha+\beta \geq 0$

$$\begin{aligned} T_k^{(n)} &= \frac{2S_{n+k} - \beta^{k-1}(\alpha+\beta)S_n}{2-\beta^{k-1}(\alpha+\beta)} \\ M_k &= \frac{\beta^{k-1}(\beta-\alpha)}{2-\beta^{k-1}(\alpha+\beta)}. \end{aligned}$$

3. $\alpha < 0 \leq \beta < 1$ and $\alpha+\beta \leq 0$

$$\begin{aligned} T_k^{(n)} &= \frac{S_{n+k} - (\alpha+\beta)\sum_{i=0}'^{k-1}(-\alpha)^{k-i-1}S_{n+i}}{1-(\alpha+\beta)\sum_{i=0}'^{k-1}(-\alpha)^{k-i-1}} \\ M_k &= \frac{(\beta-\alpha)(-\alpha)^{k-1}}{2-2(\alpha+\beta)\sum_{i=0}'^{k-1}(-\alpha)^{k-i-1}} \end{aligned}$$

where $\sum_{i=0}'^{m} b_i = b_0/2 + \sum_{i=1}^{m} b_i$.

4. $\alpha < 0 \leq \beta$ and $\alpha\cdot\beta < 1\,(k>1)$

$$\begin{aligned} T_k^{(n)} &= \frac{S_{n+k} - \alpha S_{n+k-1} + (\beta-\alpha)\sum_{i=0}'^{k-2}(-\alpha)^{k-i-1}S_{n+i}}{1-\alpha+(\beta-\alpha)\sum_{i=0}'^{k-2}(-\alpha)^{k-i-1}} \\ M_k &= \frac{(\beta-\alpha)(-\alpha)^{k-1}}{2(1-\alpha)+2(\beta-\alpha)\sum_{i=0}'^{k-2}(-\alpha)^{k-i-1}}. \end{aligned}$$

5. $\alpha < \beta < 0$ and $\alpha \cdot \beta > 1$

$$T_k^{(n)} = \frac{2S_{n+1} - (\alpha+\beta)S_n}{2-(\alpha+\beta)}$$
$$M_k = \frac{\beta-\alpha}{2-\alpha-\beta}.$$

When $k = 1$ all the cases (except 4 which is only valid for $k > 1$) reduce to the last case which corresponds to the usual Chebyshev's method of acceleration since $p_1(x_1) = (2x_1 - \alpha - \beta)/(2 - \alpha - \beta)$ is the Chebyshev polynomial of the first degree on $[\alpha, \beta]$.

Let us consider the sequence (S_n) given by

$$\frac{S_n - S}{S_{n-1} - S} = \begin{cases} 0.5 \cdot a_n & \text{if} \quad a_n \leq 0 \\ 0.3 \cdot a_n & \text{if} \quad a_n > 0 \end{cases}$$

with $a_n = \sin(a \cdot n + b/n)$, $S_0 = 0, S = 1, a = 2$ and $b = 1$. It corresponds to the third case above with $\alpha = -0.5$ and $\beta = 0.3$. We obtain (number of exact digits)

n	S_n	$T_1^{(n)}$	$T_2^{(n)}$	$T_3^{(n)}$	$T_4^{(n)}$	$T_5^{(n)}$
0	0.00	0.89	1.52	1.73	2.04	2.34
1	1.37	1.82	2.73	3.06	3.42	3.70
2	1.68	2.66	3.02	3.39	3.68	3.99
3	3.51	3.97	5.67	4.91	5.23	5.64
4	4.06	4.71	5.48	5.81	6.95	6.66
10	7.47	8.81	8.77	9.12	9.47	9.79
15	11.28	11.87	12.32	12.85	13.11	13.41

The values of M_k for $k = 1$ to 5 are

$$0.3636 \qquad 0.1600 \qquad 0.0754 \qquad 0.0367 \qquad 0.0181 \ .$$

Let us now look at true contraction. We assume that $\forall n \geq N, \alpha \leq (S_n - S)/(S_{n+1} - S) \leq \beta$ and we obtain the following optimal linear transformations having the property of true contraction (of course, although the same notation is used, α and β are not the same as before).

1. $\alpha \leq 0 < \beta < 1$ and $\alpha + \beta \geq 0$

$$T_k^{(n)} = \frac{2S_n - \beta^{k-1}(\alpha+\beta)S_{n+k}}{2-\beta^{k-1}(\alpha+\beta)}.$$

2. $\alpha < 0 \leq \beta < 1$ and $\alpha + \beta \leq 0$

$$T_k^{(n)} = \frac{S_n - (\alpha+\beta)\sum_{i=0}'^{k-1}(-\alpha)^{k-i-1}S_{n+k-i}}{1-(\alpha+\beta)\sum_{i=0}'^{k-1}(-\alpha)^{k-i-1}}.$$

3. $\alpha < \beta \leq 0$ and $\alpha \cdot \beta \leq 1\,(k > 1)$

$$T_k^{(n)} = \frac{S_n - \alpha S_{n+1} + (\beta-\alpha)\sum_{i=0}'^{k-2}(-\alpha)^{k-i-1}S_{n+k-i}}{1-\alpha+(\beta-\alpha)\sum_{i=0}'^{k-2}(-\alpha)^{k-i-1}}.$$

4. $\alpha < \beta < 0$ and $\alpha \cdot \beta > 1$

$$T_k^{(n)} = \frac{2S_{n+k-1} - (\alpha+\beta)S_{n+k}}{2-(\alpha+\beta)}.$$

5. $1 < \alpha < \beta$

$$T_k^{(n)} = \frac{2S_{n+k-1} - (\alpha+\beta)S_{n+k}}{2-(\alpha+\beta)}.$$

In these cases the coefficient M_k is the same as in the corresponding case for contraction. Thus it is interesting to study the asymptotic behaviour of M_k as k tends to infinity since it is a measure of the gain obtained by the contractive transformation. We have

1. $0 \leq \alpha < \beta < 1 : M_k \sim \frac{\beta-\alpha}{2(1-\alpha)} \cdot \beta^{k-1}$.

2. $\alpha \leq 0 < \beta < 1$ and $\alpha + \beta \geq 0 : M_k \sim (\beta-\alpha) \cdot \frac{\beta^{k-1}}{2}$.

3. $\alpha < 0 \leq \beta < 1$ and $\alpha + \beta \leq 0$:

$M_k \sim (\beta-\alpha)\cdot(1+\alpha)\cdot\frac{(-\alpha)^{k-1}}{2}$ if $\alpha > -1$,

$M_k \sim -\frac{\beta+1}{2(k-1)}\cdot\frac{1}{k}$ if $\alpha = -1$, and

$\lim_{k\to\infty} M_k = \frac{(\beta-\alpha)(1+\alpha)}{2(1+\alpha)+(\alpha+\beta)(1-\alpha)}$ if $\alpha < -1$.

4. $\alpha < \beta \leq 0$ and $\alpha \cdot \beta < 1$:
$$M_k \sim \frac{(\beta - \alpha)(1 + \alpha)}{2(1 - \alpha\beta)} \cdot (-\alpha)^{k-1} \quad \text{if } \alpha > -1,$$
$$M_k \sim \frac{1}{2k} \quad \text{if } \alpha = -1, \text{ and}$$
$$\lim_{k \to \infty} M_k = \frac{\alpha + 1}{\alpha - 1} \quad \text{if } \alpha < -1.$$

5. $\alpha < \beta < 0$ and $\alpha \cdot \beta > 1$ or $1 < \alpha < \beta$: M_k is independent of k.

Let us consider the sequence (S_n) given by

$$\frac{S_n - S}{S_{n-1} - S} = -0.5 + 0.2 \sin(2n), \qquad n = 0, 1, \ldots$$

with $S_0 = 0$ and $S = 1$. It corresponds to the fourth case above with $\alpha = -1/0.3$ and $\beta = -1/0.7$. We shall only consider the case $k = 1$ since other values of k are just a shift in the indexes. We obtain (number of exact digits)

n	S_n	$T_1^{(n)}$
0	0.00	1.14
1	0.50	1.29
2	0.68	1.70
3	0.94	2.02
4	1.46	2.33
5	1.67	2.55
10	3.35	4.59
15	4.78	6.45

The value of M_k is $M_k = 0.2817$.

In section 3.1 we saw that the ACCES-algorithm was able to construct a sequence transformation accelerating (S_n) if a sequence (T_n) such that $\exists |a| < 1$

$$\lim_{n \to \infty} (T_n - S)/(S_n - S) = a$$

is known.

We shall now ask less. We shall no more assume that the preceding ratio has a limit but only that it is contained in a known interval. Then the following result holds

Theorem 3.56

If there exist $k_1 \leq k_2$ *with* $1 \notin [k_1, k_2]$, $k_1' \leq k_2'$ *with* $1 \notin [k_1', k_2']$ *and* N *such that* $\forall n \geq N, (T_n - S)/(S_n - S) \in [k_1, k_2]$ *and* $a_n \in [k_1', k_2']$ *then the sequence* U_n *defined by*

$$U_n = (T_n - a_n S_n)/(1 - a_n), \qquad n = 0, 1, \ldots$$

satisfies $\forall n \geq N$

i) $-1 < \dfrac{k_1' - k_2}{k_1' - 1} \leq \dfrac{U_n - S}{S_n - S} \leq \dfrac{k_2' - k_1}{k_2' - 1} < 1$ *if* $1 < k_1, 1 < k_1'$ *and* $2k_1' > k_2 + 1$.

ii) $-1 < \dfrac{k_1 - k_2'}{1 - k_2'} \leq \dfrac{U_n - S}{S_n - S} \leq \dfrac{k_2 - k_1'}{1 - k_1'} < 1$ *if* $k_2 < 1, k_2' < 1$ *and* $2k_2' < k_1 + 1$.

The two cases $1 < k_1$ and $k_2' < 1$ or $1 < k_1'$ and $k_2 < 1$ lead to an impossibility which is not surprising since a_n must be an approximation of $(T_n - S)/(S_n - S)$ as explained in section 3.1.

If $1 < k_1 \leq k_2$ then (U_n) is better than (T_n) since $\forall n \geq N$

$$-1 < \frac{1}{k_2} \cdot \frac{k_1' - k_2}{k_1' - 1} \leq \frac{U_n - S}{T_n - S} \leq \frac{1}{k_1} \cdot \frac{k_2' - k_1}{k_2' - 1} < 1.$$

The simplest choice is $T_n = S_{n+1}$ but in that case the preceding assumptions cannot hold with $1 < k_1 \leq k_2$.

3.10 Least squares extrapolation

In section 2.1 we saw that the application of the E-algorithm to a sequence (S_n) consists in writing

$$S_{n+i} = S + a_1 g_1(n+i) + \cdots + a_k g_k(n+i)$$

for $i = 0, \ldots, k$ and then solving this system for the unknown S denoted $E_k^{(n)}$.

We shall now write this system for $i = 0, \ldots, m$ with $m \geq k$ and then solve this overdetermined system in the least squares sense for the unknown S which will be denoted by ${}_mE_k^{(n)}$ since it depends on the three indexes m, k and n.

If we set

$$\begin{aligned} g_i &= (g_i(n), \ldots, g_i(n+m))^T \ , \quad i \geq 1 \\ g_0 &= (1, \ldots, 1)^T \\ V &= (S_n, \ldots, S_{n+m})^T \end{aligned}$$

and if (u, v) denotes the scalar product between the vectors u and v with components u_i and v_i, that is

$$(u, v) = u_0 \overline{v}_0 + \cdots + u_m \overline{v}_m$$

then we have

$$\begin{pmatrix} 1 & (g_0, g_1)/(g_0, g_0) & \cdots & (g_0, g_k)/(g_0, g_0) \\ \vdots & \vdots & & \vdots \\ 1 & (g_k, g_1)/(g_k, g_0) & \cdots & (g_k, g_k)/(g_k, g_0) \end{pmatrix} \cdot \begin{pmatrix} {}_m E_k^{(n)} \\ a_1 \\ \vdots \\ a_k \end{pmatrix} = \begin{pmatrix} (g_0, V)/(g_0, g_0) \\ \vdots \\ (g_k, V)/(g_k, g_0) \end{pmatrix}.$$

Thus this system has the same structure as the system giving $E_k^{(n)}$ which shows that, for fixed values of m and n, the E-algorithm can be used to compute recursively ${}_m E_k^{(n)}$ for $k = 0, \ldots, m$.
Setting, in the E-algorithm

$$\begin{aligned} E_0^{(j)} &= \frac{(g_j, V)}{(g_j, g_0)} \\ g_{0,i}^{(j)} &= \frac{(g_j, g_i)}{(g_j, g_0)} \ , \qquad j = 0, 1, \ldots \text{ and } i = 1, 2, \ldots \end{aligned}$$

we shall obtain

$$E_k^{(0)} = {}_m E_k^{(n)}.$$

For $k = m$, the least squares solution of the system is identical to the value obtained by the E-algorithm applied to the S_n's and the $g_i(n)$'s.

If the values of n or/and m are modified, then the definition of the scalar product is also modified and the computations have to be started again from the beginning.

The main advantage of least squares extrapolation is to be less sensitive to small perturbations on the data. Let us, for example, take $S_n = 1/(n+1)$ and $g_i(n) = \Delta S_{n+i-1}$ (that is Shanks' transformation). We know that we shall obtain $E_k^{(n)} = (k+1)^{-1}(n+k+1)^{-1}$. Let us add a random perturbation not greater than 10^{-4} to the sequence (S_n). Then the $g_i(n)$'s are also perturbed accordingly and we have for $n = 0$ and $m = 4$

k	${}_4E_k^{(0)}$ perturbed S_n's	${}_4E_k^{(0)}$ unperturbed S_n's
0	0.4567459	0.4566666
1	0.1782071	0.1781660
2	0.0937578	0.0938178
3	0.0581327	0.0583216
4	0.0398319	0.0399999

Thus the smallest k is, the less the ${}_4E_k^{(0)}$'s are sensitive to the perturbation. When $k = 4$ we see that the results obtained (which are identical to those given by direct application of the E-algorithm to (S_n)) are quite sensitive to perturbations since we should have $E_4^{(0)} = 0.04$.

Least squares extrapolation is treated in details by Cordellier [121].

Chapter 4

VECTOR EXTRAPOLATION ALGORITHMS

Very often in numerical analysis, vector sequences are obtained. This is, in particular, the case in iterative methods for solving systems of linear and nonlinear equations, in the solution of differential and partial differential equations by finite differences or finite elements methods (see Marchuk and Shaidurov [306, 307] on this topics), in the computation of eigenelements of a matrix and their derivatives and in some other subjects. Very often also the convergence of such sequences is slow and convergence acceleration is mandatory. Of course it is possible to apply separately on the sequences of each component an acceleration algorithm. However vector acceleration methods were specially developed for dealing with vector sequences. Usually vector extrapolation algorithms are more interesting than the scalar ones on each component, because their theory is more advanced and because, in particular, they can be related to projection methods which are well known in numerical analysis. Their theory recently attracted a wide attention, a review of which was given by Smith, Ford and Sidi [415, 416] (see also, Sidi, Ford and Smith [408]). It is not our purpose here to give the details of this theory and we shall restrict ourselves to the description of the various algorithms and their main properties and applications.

We shall begin with the vector ε-algorithm, obtained by Wynn [473], which is the first algorithm found for accelerating vector sequences. It was obtained directly from the rule of the scalar algorithm and thus lacks of a firm theoretical basis. This explains why very few theoretical results on it are known and why they were quite difficult to get. To avoid this drawback, another way for obtaining a vector generalization of Shanks' transformation and the ε-algorithm was explored by Brezinski [51]; it leads to the so-called topological ε-algorithm. Then we shall present the

vector E-algorithm and the recursive projection algorithm which are the two algorithms obtained chronologically after that. Then came the H-algorithm and algorithms by Ford and Sidi [163] which can be used for implementing economically the topological ε-algorithm and the vector E-algorithm in particular. A section will also be devoted to the vector G-transformation.

As explained by Brezinski [86] all these vector sequence transformations (except possibly the vector ε-algorithm) can be expressed by a ratio of two determinants similar to that given in section 1.5

$$R_k = \frac{\begin{vmatrix} e_0 & \cdots & e_k \\ a_1^{(0)} & \cdots & a_1^{(k)} \\ \vdots & & \vdots \\ a_k^{(0)} & \cdots & a_k^{(k)} \end{vmatrix}}{\begin{vmatrix} b_0 & \cdots & b_k \\ a_1^{(0)} & \cdots & a_1^{(k)} \\ \vdots & & \vdots \\ a_k^{(0)} & \cdots & a_k^{(k)} \end{vmatrix}}$$

where the e_i's are vectors and the $a_i^{(j)}$'s and the b_i's scalars. The determinant in the numerator of R_k is now a vector. It is the vector obtained by expanding this determinant with respect to its first row by using the classical rules for expanding a determinant. In other words, as explained in section 1.5, we have

$$R_k = \alpha_0 e_0 + \cdots + \alpha_k e_k$$

where the α_i's are solution of the system of linear equations

$$\begin{cases} \alpha_0 b_0 + \cdots + \alpha_k b_k & = & 1 \\ \alpha_0 a_1^{(0)} + \cdots + \alpha_k a_1^{(k)} & = & 0 \\ \vdots & & \vdots \\ \alpha_0 a_k^{(0)} + \cdots + \alpha_k a_k^{(k)} & = & 0. \end{cases}$$

This interpretation shows that (R_k) can be computed recursively by applying the bordering method to the previous system as seen in section 1.8.

As we shall see below the topological ε-algorithm is recovered for the choice $e_i = S_{n+i}$, $a_j^{(i)} =< y, \Delta S_{n+i+j-1} >$ and $b_i = 1$ where y is an arbitrary vector and $< y, z >= \sum_{i=1}^{p} y_i z_i$ where the y_i's and the z_i's are the components of the (real or complex) vectors y and z of dimension p. In that case we shall have $R_k = \varepsilon_{2k}^{(n)}$.

If we take $e_0 = S_n$, $e_i = g_i(n)$ for $i \geq 1, a_j^{(0)} =< y, \Delta S_{n+j-1} >$, $a_j^{(i)} =< y, \Delta g_i(n+j-1) >$ for $i \geq 1$ where Δ operates on the index $n, b_0 = 1$ and $b_i = 0$ for $i \geq 0$ we obtain the vector E-algorithm and $R_k = E_k^{(n)}$.

Choosing $e_0 = y, e_i = x_i$ for $i \geq 1, a_j^{(0)} =< z_j, y >$, $a_j^{(i)} =< z_j, x_i >$ for $i \geq 1$ where the z_j's are arbitrary vectors, $b_0 = 1$ and $b_i = 0$ for $i \geq 1$, leads to the recursive projection algorithm (RPA) and we have $R_k = E_k$. The compact recursive projection algorithm and one of its variants can also be put into this framework.

Finally if $e_i = S_{n+i}$, $a_j^{(i)} = g_j(n+i)$ and $b_i = 1$, we recover the H-algorithm and we obtain $R_k = H_k^{(n)}$.

In the sequel we shall mainly consider two cases: $b_0 = \cdots = b_k = 1$ and $b_0 = 1$, $b_1 = \cdots = b_k = 0$. It is easy to go from one case to the other one since

$$\frac{\begin{vmatrix} e_0 & \cdots & e_k \\ a_1^{(0)} & \cdots & a_1^{(k)} \\ \vdots & & \vdots \\ a_k^{(0)} & \cdots & a_k^{(k)} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ a_1^{(0)} & \cdots & a_1^{(k)} \\ \vdots & & \vdots \\ a_k^{(0)} & \cdots & a_k^{(k)} \end{vmatrix}} = \frac{\begin{vmatrix} e_0 & e_1 - e_0 & \cdots & e_k - e_{k-1} \\ a_1^{(0)} & a_1^{(1)} - a_1^{(0)} & \cdots & a_1^{(k)} - a_1^{(k-1)} \\ \vdots & \vdots & & \vdots \\ a_k^{(0)} & a_k^{(1)} - a_k^{(0)} & \cdots & a_k^{(k)} - a_k^{(k-1)} \end{vmatrix}}{\begin{vmatrix} 1 & 0 & \cdots & 0 \\ a_1^{(0)} & a_1^{(1)} - a_1^{(0)} & \cdots & a_1^{(k)} - a_1^{(k-1)} \\ \vdots & \vdots & & \vdots \\ a_k^{(0)} & a_k^{(1)} - a_k^{(0)} & \cdots & a_k^{(k)} - a_k^{(k-1)} \end{vmatrix}}.$$

All these algorithms fit into the framework of biorthogonality as explained by Brezinski [89]. Their properties of translativity were recently studied by Sadok [384] whose results will be now briefly described.

We consider the vector sequence transformation $T : (S_n) \longmapsto (T_n)$ given by

$$T_n = F(S_n, \ldots, S_{n+k}), \qquad n = 0, 1, \ldots$$

where $S_n \in \mathbf{C}^p$ and $F : (\mathbf{C}^p)^{k+1} \longmapsto \mathbf{C}^p$. We shall say that F is translative on A if $\forall b \in \mathbf{C}^p$, $\forall x_0, \ldots, x_k \in A \subseteq \mathbf{C}^p$

$$F(x_0 + b, \ldots, x_k + b) = F(x_0, \ldots, x_k) + b.$$

F is assumed to be twice continuously differentiable on A in the sense of Fréchet. Let $f : A^{k+1} \longmapsto A$. We set

$$D_i f(x_0, \ldots, x_k) = \frac{\partial f(x_0, \ldots, x_k)}{\partial x_i}.$$

$D_i f$ is the matrix whose elements are $\partial f_i / (\partial x_i)_j$ the partial derivative of the i-th component of f with respect to the j-th component of x_i.

We have the

Theorem 4.1

A necessary and sufficient condition that F be translative on A is that there exists f such that

$$F(x_0, \ldots, x_k) = [Df(x_0, \ldots, x_k)]^{-1} \cdot f(x_0, \ldots, x_k)$$

with $D^2 f(x_0, \ldots, x_k)$ identically zero on A and where $D = D_0 + \cdots + D_k$.

Let us recall (see Dieudonné [143]) that $D^2 f$ is a multilinear application. It is a linear mapping from $\mathcal{L}(\mathbf{C}^p, \mathbf{C}^p)$ into $\mathbf{C}^p$ where $\mathcal{L}(\mathbf{C}^p, \mathbf{C}^p)$ denotes the set of linear applications from $\mathbf{C}^p$ into $\mathbf{C}^p$ (that is the set of complex $p \times p$ matrices).

This result was applied by Sadok [384] to the study of the translativity of the H-algorithm (section 4.5) and the vector E-algorithm (section 4.3). The interested reader is referred to this paper for details. An application of this theorem to convergence acceleration will be given in section 6.1.5.

Let us now study in details vector extrapolation algorithms.

4.1 The vector ε-algorithm

Let us again consider the rule of the ε-algorithm

$$\begin{aligned} & \varepsilon_{-1}^{(n)} = 0\,, \quad \varepsilon_0^{(n)} = S_n\,, && n = 0, 1, \ldots \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \left[\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}\right]^{-1}, && k, n = 0, 1, \ldots \end{aligned}$$

Now if S_n is a (real or complex) vector of dimension p and if $\varepsilon_{-1}^{(n)}$ is the zero vector, then this algorithm can be used if the inverse y^{-1} of the vector y is defined. Of course this definition must reduce to $y^{-1} = 1/y$ if y is a vector of dimension 1, that is a (real or complex) number. We shall take for the inverse y^{-1} of the vector y the vector given by

$$y^{-1} = \frac{y}{(y, y)}$$

where (y, y) is the scalar product of the vector y by itself that is

$$(y, y) = \sum_{i=1}^{p} y_i \overline{y_i} = \sum_{i=1}^{p} |y_i|^2$$

where $\overline{y_i}$ denotes the complex conjugate of y_i and $|y_i|$ its modulus. With this definition of the inverse of a vector, the ε-algorithm can be applied to vector sequences; it is the so-called vector ε-algorithm as obtained by Wynn [473]. Up to now it has been impossible to obtain determinantal expressions for the vectors $\varepsilon_k^{(n)}$ even by using computer algebra and it results in a great difficulty for finding the kernel of the vector ε-algorithm. This kernel was first obtained in the real case by McLeod [317] and then another quite different proof for the complex case was given by Graves-Morris [192]. It is as follows

Theorem 4.2

If the vector ε-algorithm is applied to a sequence of complex vectors (S_n) *such that* $\forall n \geq N$

$$\sum_{i=0}^{k} a_i (S_{n+i} - S) = 0$$

where S *is a vector and the* a_i*'s are complex numbers such that* $a_k \neq 0$ *and* $a_0 + \cdots + a_k \neq 0$ *then* $\forall n \geq N, \varepsilon_{2k}^{(n)} = S$.

It must be noticed that this relation has exactly the same form as the relation defining the kernel of the scalar ε-algorithm (theorem 2.18). However, in the vector case, the condition is only sufficient while it was also necessary in the scalar case.

Due to the Cayley-Hamilton theorem, such a relation is satisfied if the sequence (S_n) is given by $S_{n+1} = A\, S_n + b, n = 0, 1, \ldots$ with an

arbitrary S_0 where b is a vector and A a matrix. Since this result has applications in the solution of systems of linear and nonlinear equations, this question will be studied in details in section 6.2.

Obviously the vector ε-algorithm is quasi-linear that is if (S_n) is replaced by $(a\,S_n + b)$ where a is a scalar and b a vector then $\varepsilon_{2k}^{(n)}$ becomes $a\,\varepsilon_{2k}^{(n)} + b$ and $\varepsilon_{2k+1}^{(n)}$ becomes $\varepsilon_{2k+1}^{(n)} / a$. But we even have more. Let $\varepsilon_k^{(n)}$ be the vectors obtained by applying it to the vectors S_n and let $\tilde{\varepsilon}_k^{(n)}$ be those obtained from $A\,S_n + b$ where b is a vector and A a unitary matrix (that is $A^{-1} = A^*$ where the star denotes the transpose conjugate) then $\tilde{\varepsilon}_{2k}^{(n)} = A\,\varepsilon_{2k}^{(n)} + b$ and $\tilde{\varepsilon}_{2k+1}^{(n)} = A\,\varepsilon_{2k+1}^{(n)}$.

As in the scalar ε-algorithm, the vectors with an odd lower index are intermediate computations without any interest (except in one case which will be studied in section 6.3.1). Thus they can be eliminated and a cross rule for the vector ε-algorithm is obtained. It has the same form as the scalar one

$$E = C + \left[(N-C)^{-1} + (S-C)^{-1} - (W-C)^{-1}\right]^{-1}$$

where $y^{-1} = y/(y,y)$ for any non-zero vector y and with

$$C = \varepsilon_{2k}^{(n+1)},\ N = \varepsilon_{2k}^{(n)},\ S = \varepsilon_{2k}^{(n+2)},\ W = \varepsilon_{2k-2}^{(n+2)},\ E = \varepsilon_{2k+2}^{(n)}.$$

The initializations are $\varepsilon_0^{(n)} = S_n$ and $\varepsilon_{-2}^{(n)} = \infty$.

Of course, as in the scalar case, numerical instability occurs if two adjacent vectors are almost equal. A particular rule which is more stable was obtained by Cordellier [118]: if N, S and W are different from C and if $(N-C)^{-1} + (S-C)^{-1}$ is not equal to $(W-C)^{-1}$ then E can be computed by

$$E = (\alpha N + \beta S - \gamma W + \lambda C)/(\alpha + \beta - \gamma + \lambda)$$

with

$$\alpha = (N-C, N-C)^{-1},\quad \beta = (S-C, S-C)^{-1},\quad \gamma = (W-C, W-C)^{-1}$$

and

$$\lambda = \alpha\gamma(N-W, N-W) + \beta\gamma(W-S, W-S) - \alpha\beta(S-N, S-N).$$

If C is infinity, this rule reduces to

$$E = N + S - W.$$

Let us give a numerical example to illustrate the improvement brought by this particular rule. We consider the sequence of vectors generated by $S_n = S_{n-4}/2$ for $n = 4, 5, \ldots$ with

$$S_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad S_1 = \begin{pmatrix} 1 \\ 10^{-8} \\ 0 \end{pmatrix} \quad S_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad S_3 = \begin{pmatrix} 0 \\ 10^{-8} \\ 1 \end{pmatrix}.$$

Then, from theorem 4.2, we should have $\varepsilon_8^{(0)} = 0$. Using the normal rule of the vector ε-algorithm we obtain

$$\varepsilon_8^{(0)} = \begin{pmatrix} 0.61 \\ -0.36 \cdot 10^{-6} \\ 0.83 \end{pmatrix}$$

while the particular rule gives

$$\varepsilon_8^{(0)} = \begin{pmatrix} 0.72 \cdot 10^{-15} \\ 0.25 \cdot 10^{-22} \\ 0.36 \cdot 10^{-15} \end{pmatrix}.$$

The vector ε-algorithm and its particular rule can be implemented via the subroutine EPSVEC.

In the vector ε-algorithm, use was made of the inverse of a vector. This inverse involved in its denominator the scalar product of a vector by itself that is the square of its euclidean norm (that is its length). Instead of the euclidean norm any other norm can be used and we can define the inverse of the vector y as

$$y^{-1} = \frac{y}{\|y\|^2}.$$

For example one can take

$$\|y\| = \max_{1 \le i \le p} |y_i|$$

or

$$\|y\| = \sum_{i=1}^{p} |y_i|.$$

Thus we obtain a new ε-algorithm whose properties were studied by Brezinski [46]. Such an algorithm can be applied to a sequence of elements of an arbitrary Banach space instead of $\mathbb{C}^p$. It was proved in particular that the sequence $\left(\varepsilon_2^{(n)}\right)$ converges faster than (S_{n+1}) for a linear sequence, that is

$$\lim_{n\to\infty} \left\|\varepsilon_2^{(n)} - S\right\| \Big/ \|S_n - S\| = 0$$

if (S_n) satisfies

$$S_{n+1} - S = a_n (S_n - S)$$

where (a_n) is a sequence of numbers converging to $a \in]-1, 1]$ or if

$$S_n - S = \lambda^n (y + e_n)$$

with $-1 \leq \lambda < 1, y \in \mathbb{C}^p$ and (e_n) a sequence of vectors converging to zero.

To end this section let us mention that the ε-algorithm can be applied to a sequence of matrices (S_n) where now $\left(\Delta\varepsilon_k^{(n)}\right)^{-1}$ is the inverse of the matrix $\Delta\varepsilon_k^{(n)}$ and $\varepsilon_{-1}^{(n)}$ is the null matrix.

4.2 The topological ε-algorithm

As we saw in the previous section, the vector ε-algorithm lacks of a determinantal formula. This is because the starting point for its achievement was the rule of the scalar algorithm. To avoid this drawback another approach was proposed by Brezinski [51]. It will lead to a determinantal formula and to a recursive algorithm, which is different from the vector ε-algorithm, called the topological ε-algorithm.

The starting point of this new approach is similar to Shanks'. We consider a sequence of vectors such that, for all n

$$a_0(S_n - S) + \cdots + a_k(S_{n+k} - S) = 0$$

where S is a vector and the a_i's are scalars with $a_k \neq 0$ and $a_0 + \cdots + a_k \neq 0$. The problem is to compute the vector S. If the a_i's are known then S will be given by

$$S = \frac{a_0 S_n + \cdots + a_k S_{n+k}}{a_0 + \cdots + a_k}.$$

Thus let us compute these a_i's.

We have, by subtraction

$$a_0 \Delta S_n + \cdots + a_k \Delta S_{n+k} = 0.$$

Let y be an arbitrary vector. We consider the bilinear form $<y, u>$, defined by

$$<y, u> = \sum_{i=1}^{p} y_i u_i$$

where the y_i's and the u_i's are the components of y and u respectively. Then we have, for all n

$$a_0 <y, \Delta S_n> + \cdots + a_k <y, \Delta S_{n+k}> = 0.$$

Writing this equation for the indexes $n, n+1, \ldots, n+k-1$ and assuming without any loss of generality that the sum of the a_i's equals one, we obtain the system

$$\begin{cases} a_0 & + \cdots + & a_k & = 1 \\ a_0 <y, \Delta S_n> & + \cdots + & a_k <y, \Delta S_{n+k}> & = 0 \\ \vdots & & & \vdots \\ a_0 <y, \Delta S_{n+k-1}> & + \cdots + & a_k <y, \Delta S_{n+2k-1}> & = 0. \end{cases}$$

If the determinant of this system is different from zero (which will be assumed in the sequel) its solution provides $a_0, \ldots, a_k$ and then S can be obtained from

$$S = a_0 S_{n+i} + \cdots + a_k S_{n+k+i}$$

for any i. Thanks to the particular form of the right hand side of the previous system we obtain for S the determinantal formula (the numerator having the meaning explained in the introduction of this chapter)

$$\frac{\begin{vmatrix} S_{n+i} & \cdots & S_{n+k+i} \\ <y, \Delta S_n> & \cdots & <y, \Delta S_{n+k}> \\ \vdots & & \vdots \\ <y, \Delta S_{n+k-1}> & \cdots & <y, \Delta S_{n+2k-1}> \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ <y, \Delta S_n> & \cdots & <y, \Delta S_{n+k}> \\ \vdots & & \vdots \\ <y, \Delta S_{n+k-1}> & \cdots & <y, \Delta S_{n+2k-1}> \end{vmatrix}}.$$

When $i = 0$, we shall denote this ratio of determinants by $e_k(S_n)$ and when $i = k$ we shall use $\tilde{e}_k(S_n)$. For other values of i, it will be denoted ${}_ie_k(S_n)$. By construction we have the

Theorem 4.3

If $\forall n \geq N, a_0(S_n - S) + \cdots + a_k(S_{n+k} - S) = 0$ *where* S *is a vector and the* a_i*'s are (real or complex) numbers such that* $a_k \neq 0$ *and* $a_0 + \cdots + a_k \neq 0$ *then,* $\forall n \geq N, e_k(S_n) = S$ *and* $\tilde{e}_k(S_n) = S$.

Obviously $e_k(aS_n + b) = ae_k(S_n) + b$ and $\tilde{e}_k(aS_n + b) = a\tilde{e}_k(S_n) + b$ where b is a vector and a a non-zero constant.

If (S_n) is replaced by $(A\,S_n + b)$ and y by Ay where A is a unitary matrix and b a vector then ${}_ie_k(A\,S_n + b) = A\,{}_ie_k(S_n) + b$. If A is hermitian regular, if (S_n) is replaced by $(A\,S_n + b)$ and y by $A^{-1}y$ then the same result holds. As pointed out by Midy [320] such a property is of interest if the components of the vectors S_n are physical quantities not of the same dimension (in the physical sense) and if scaling is needed.

The vectors $e_k(S_n)$ can be recursively computed by the so-called topological ε-algorithm whose rules are as follows

$$\varepsilon_{-1}^{(n)} = 0\,, \quad \varepsilon_0^{(n)} = S_n\,, \qquad n = 0, 1, \ldots$$

$$\varepsilon_{2k+1}^{(n)} = \varepsilon_{2k-1}^{(n+1)} + \frac{y}{< y\,,\, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >}\,, \qquad k, n = 0, 1, \ldots$$

$$\varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}}{< \varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}\,,\, \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)} >}\,, \qquad k, n = 0, 1, \ldots$$

and we have

$$\varepsilon_{2k}^{(n)} = e_k(S_n)\,, \quad \varepsilon_{2k+1}^{(n)} = \frac{y}{< y, e_k(\Delta S_n) >}\,, \quad k, n = 0, 1, \ldots$$

When the dimension of the vectors is one, the rules of the topological ε-algorithm reduce to that of the scalar one and the above connection also.

Let $\tilde{H}_{k+1}(S_n)$ be the determinant in the numerator of $e_k(S_n)$ then $H_{k+1}(< y, S_n >) = < y, \tilde{H}_{k+1}(S_n) >$ and we have

$$e_k(S_n) = \frac{\tilde{H}_{k+1}(S_n)}{H_k(< y, \Delta^2 S_n >)}$$

$$e_{k+1}(S_n) = e_k(S_n) - \frac{H_{k+1}(< y, \Delta S_n >) \cdot \tilde{H}_{k+1}(\Delta S_n)}{H_{k+1}(< y, \Delta^2 S_n >) \cdot H_k(< y, \Delta^2 S_n >)}.$$

We also have the following recurrence relationship among these generalizations of Hankel determinants

$$\tilde{H}_{k+1}(S_n)\cdot H_{k-1}(<y,\Delta S_{n+1}>) = \tilde{H}_k(S_n)\cdot H_k(<y,\Delta S_{n+1}>) - \tilde{H}_k(S_{n+1})\cdot H_k(<y,\Delta S_n>).$$

The proof for the topological ε-algorithm is based on the fact that for determinants similar to the numerator of $e_k(S_n)$, determinantal identities generalizing those of Sylvester and Schweins hold as proved by Brezinski [69]. The relation between the scalar Shanks' transformation and the vector ones is clear; one has

$$e_k(<y,S_n>) = <y, e_k(S_n)> = <y, \tilde{e}_k(S_n)> .$$

The subroutine EPSTOP performs the topological ε-algorithm.

For computing recursively the $\tilde{e}_k(S_n)$'s the second topological ε-algorithm can be used. It is as follows

$$\varepsilon_{-1}^{(n)} = 0\,, \quad \varepsilon_0^{(n)} = S_n\,, \qquad n = 0,1,\ldots$$

$$\varepsilon_{2k+1}^{(n)} = \varepsilon_{2k-1}^{(n+1)} + \frac{y}{<y\,,\ \varepsilon_{2k}^{(n+1)} - \varepsilon_{2k}^{(n)}>}\,, \qquad k,n = 0,1,\ldots$$

$$\varepsilon_{2k+2}^{(n)} = \varepsilon_{2k}^{(n+1)} + \frac{\varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}}{<\varepsilon_{2k+1}^{(n+1)} - \varepsilon_{2k+1}^{(n)}\,,\ \varepsilon_{2k}^{(n+2)} - \varepsilon_{2k}^{(n+1)}>}\,, \qquad k,n = 0,1,\ldots$$

and we obtain

$$\varepsilon_{2k}^{(n)} = \tilde{e}_k(S_n)\,, \quad \varepsilon_{2k+1}^{(n)} = \frac{y}{<y\,,\ \tilde{e}_k(\Delta S_n)>}.$$

In the preceding algorithms one can think of replacing the bilinear form $<y,u>$ by the usual scalar product

$$(y,u) = \sum_{i=1}^{p} y_i \overline{u}_i.$$

Of course, when all the vectors are real, it leads to the same results. However, in the complex case, the results obtained are usually different although theorem 4.2 still holds. It must also be noticed that (y,u) is not equal to (u,y) and thus the order of the vectors in the scalar product is very important. These questions are discussed in details by Tan [428].

For a fixed value of n (usually $n = 0$), the ${}_i e_k(S_n)$'s can be recursively computed thanks to their connection with orthogonal polynomials as described by Brezinski [63] (pp. 182–184). We set

$$
\begin{aligned}
&{}_n e_{-1}(S_0) = 0\,, \qquad {}_n e_0(S_0) = S_n\,, \qquad n = 0, 1, \ldots \\
&p_0^{(-1)} = p_{-1}^{(-1)} = 0\,, \quad h_{-1} = 1\,, \qquad d_{-1} = 0 \\
&p_0^{(0)} = 1\,, \qquad p_{-1}^{(0)} = p_1^{(0)} = 0\,, \quad h_0 = \langle y, \Delta S_0 \rangle\,, \quad d_0 = 1.
\end{aligned}
$$

Then for $k = 0, 1, \ldots$ we compute

$$
\begin{aligned}
h_k &= \sum_{i=0}^{k} \langle y, \Delta S_{k+i} \rangle\, p_i^{(k)} \\
\gamma_k &= \sum_{i=0}^{k} \langle y, \Delta S_{k+i+1} \rangle\, p_i^{(k)} \\
\alpha_k &= \gamma_k + p_{k-1}^{(k)} h_k \\
B_{k+1} &= -\frac{\alpha_k}{h_k} \\
C_{k+1} &= \frac{h_k}{h_{k-1}} \\
p_{k+1}^{(k+1)} &= 1\,, \quad p_{-1}^{(k+1)} = p_{k+2}^{(k+1)} = 0 \\
p_i^{(k+1)} &= p_{i-1}^{(k)} + B_{k+1} p_i^{(k)} - C_{k+1} p_i^{(k-1)}\,, \qquad i = 0, \ldots, k \\
d_{k+1} &= \sum_{i=0}^{k+1} p_i^{(k+1)}.
\end{aligned}
$$

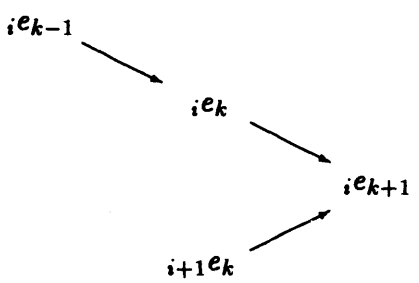
Then we have, for $i = 0, 1, \ldots$

$${}_i e_{k+1}(S_0) = B_{k+1} \cdot \frac{d_k}{d_{k+1}} \cdot {}_i e_k(S_0) + \frac{d_k}{d_{k+1}} \cdot {}_{i+1} e_k(S_0) - C_{k+1} \cdot \frac{d_{k-1}}{d_{k+1}} \cdot {}_i e_{k-1}(S_0).$$

These vectors can be displayed in a double entry table as follows

$$
\begin{array}{lllll}
{}_0e_{-1} = 0 & & & & \\
 & {}_0e_0 = S_0 & & & \\
{}_1e_{-1} = 0 & & {}_0e_1 & & \\
 & {}_1e_0 = S_1 & & {}_0e_2 & \\
{}_2e_{-1} = 0 & & {}_1e_1 & & {}_0e_3 \\
 & {}_2e_0 = S_2 & & {}_1e_2 & \\
{}_3e_{-1} = 0 & & {}_2e_1 & & {}_1e_3 \\
\vdots & {}_3e_0 = S_3 & & {}_2e_2 & \\
\vdots & \vdots & {}_3e_1 & & {}_2e_3 \\
\vdots & \vdots & \vdots & {}_3e_2 & \\
\vdots & \vdots & \vdots & \vdots & {}_3e_3 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{array}
$$

where ${}_ie_k = {}_ie_k(S_0)$ for simplicity. Thus, the preceding relation links the vectors



We previously saw that, for obtaining the topological ε-algorithm, we have to solve a certain linear system which gives us the a_i's. In this system we make use of the equations

$$a_0 < y, \Delta S_i > + \cdots + a_k < y, \Delta S_{i+k} > = 0$$

for $i = n, \ldots, n + k - 1$. Instead of this, we can use only the equation for $i = n$ but with k different vectors $y_1, \ldots, y_k$. Thus the a_i's will be

given as the solution of the system

$$\begin{cases} a_0 + \cdots + a_k = 1 \\ a_0 <y_1, \Delta S_n> + \cdots + a_k <y_1, \Delta S_{n+k}> = 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ a_0 <y_k, \Delta S_n> + \cdots + a_k <y_k, \Delta S_{n+k}> = 0 \end{cases}$$

provided its determinant is different from zero (which will be assumed in the sequel and is only possible when $k \leq p$, the dimension of the vectors). Thus we set

$$\bar{e}_k(S_n) = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ <y_1, \Delta S_n> & \cdots & <y_1, \Delta S_{n+k}> \\ \vdots & & \vdots \\ <y_k, \Delta S_n> & \cdots & <y_k, \Delta S_{n+k}> \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ <y_1, \Delta S_n> & \cdots & <y_1, \Delta S_{n+k}> \\ \vdots & & \vdots \\ <y_k, \Delta S_n> & \cdots & <y_k, \Delta S_{n+k}> \end{vmatrix}}.$$

Obviously a result similar to that of theorem 4.3 holds and the property of quasi-linearity as well. Although this vector sequence transformation was given by Brezinski [51], a recursive algorithm for its implementation was only recently discovered by Jbilou [244]. If we set

$$\beta_k^{(n)} = \frac{\begin{vmatrix} \Delta S_n & \cdots & \Delta S_{n+k} \\ <y_1, \Delta S_n> & \cdots & <y_1, \Delta S_{n+k}> \\ \vdots & & \vdots \\ <y_k, \Delta S_n> & \cdots & <y_k, \Delta S_{n+k}> \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ <y_1, \Delta S_n> & \cdots & <y_1, \Delta S_{n+k}> \\ \vdots & & \vdots \\ <y_k, \Delta S_n> & \cdots & <y_k, \Delta S_{n+k}> \end{vmatrix}}$$

then we have

$$\beta_0^{(n)} = \Delta S_n, \quad \bar{e}_0(S_n) = S_n, \qquad n = 0, 1, \ldots$$

Then for $k, n = 0, 1, \ldots$ we compute

$$\bar{e}_{k+1}(S_n) = \frac{< y_{k+1}, \beta_k^{(n+1)} > \bar{e}_k(S_n) - < y_{k+1}, \beta_k^{(n)} > \bar{e}_k(S_{n+1})}{< y_{k+1}, \beta_k^{(n+1)} > - < y_{k+1}, \beta_k^{(n)} >}$$

$$\beta_{k+1}^{(n)} = \frac{< y_{k+1}, \beta_k^{(n+1)} > \beta_k^{(n)} - < y_{k+1}, \beta_k^{(n)} > \beta_k^{(n+1)}}{< y_{k+1}, \beta_k^{(n+1)} > - < y_{k+1}, \beta_k^{(n)} >}.$$

This algorithm is called the $S\beta$-algorithm. Its proof is a direct application of the extension of Sylvester's identity to the vector case. Although it is very elegant, this algorithm presents the drawback of needing the construction of two arrays of vectors. If their dimension is large then the algorithm is very much storage consuming. This drawback can be avoided as follows.

If we set

$$g_{k,i}^{(n)} = < y_i, \beta_k^{(n)} >$$

then these quantities are exactly those computed in the auxiliary rule of the scalar E-algorithm with $g_i(n) = < y_i, \Delta S_n >$ and the auxiliary rule of the $S\beta$-algorithm reduces to that auxiliary rule. Thus we finally have the following algorithm

$$\bar{e}_0(S_n) = S_n\,, \quad g_{0,i}^{(n)} = < y_i, \Delta S_n >\,, \qquad n = 0, 1, \ldots\,;\ i \geq 1.$$

Then for $k = 1, 2, \ldots$ and $n = 0, 1, \ldots$ we compute

$$\bar{e}_k(S_n) = \frac{g_{k-1,k}^{(n+1)} \cdot \bar{e}_{k-1}(S_n) - g_{k-1,k}^{(n)} \cdot \bar{e}_{k-1}(S_{n+1})}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}}$$

$$g_{k,i}^{(n)} = \frac{g_{k-1,i}^{(n)} \cdot g_{k-1,k}^{(n+1)} - g_{k-1,i}^{(n+1)} \cdot g_{k-1,k}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}}\,, \quad \text{for } i > k.$$

This algorithm is exactly the H-algorithm which will be studied in more details in section 4.5.

The choice of the arbitrary vectors y and $y_1, \ldots, y_k$ in these algorithms is an open problem.

Instead of the ε-algorithm, vector sequences can also be extrapolated by the vector Padé approximants of van Iseghem [443] which share the same algebraic properties and have similar applications (see chapter 6).

4.3 The vector E-algorithm

For obtaining a vector version of the E-algorithm, almost the same path as for the topological ε-algorithm has to be followed. We consider a sequence of vectors such that, for all n

$$S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n)$$

where S is a vector, where the $(g_i(n))$ are auxiliary known vector sequences and the a_i's are (real or complex) numbers. The problem is to compute the unknown vector S. By subtraction, we have

$$\Delta S_n = a_1 \Delta g_1(n) + \cdots + a_k \Delta g_k(n)$$

which can be written as

$$a_0 \Delta S_n + a_1 \Delta g_1(n) + \cdots + a_k \Delta g_k(n) = 0$$

with $a_0 = -1$. If y is an arbitrary vector then we have, for all n

$$a_0 < y, \Delta S_n > + a_1 < y, \Delta g_1(n) > + \cdots + a_k < y, \Delta g_k(n) > = 0.$$

Writing this equation for the indexes $n, n+1, \ldots, n+k-1$ leads to the system of linear equations

$$\begin{cases} a_0 = -1 \\ a_0 < y, \Delta S_n > + a_1 < y, \Delta g_1(n) > + \cdots + a_k < y, \Delta g_k(n) > = 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ a_0 < y, \Delta S_{n+k-1} > + \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \qquad a_1 < y, \Delta g_1(n+k-1) > + \cdots + a_k < y, \Delta g_k(n+k-1) > = 0. \end{cases}$$

If the determinant of this system is different from zero (which will be assumed in the sequel) then its solution provides $a_0, \ldots, a_k$ and then S, which will be denoted by $E_k^{(n)}$ since it depends on n and k, is given by

$$S = -a_0 S_n - a_1 g_1(n) - \cdots - a_k g_k(n).$$

Thanks to the particular form of the right hand side of the previous system we obtain the following determinantal formula, where the nu-

merator has the meaning explained in the introduction of this chapter

$$E_k^{(n)} = \frac{\begin{vmatrix} S_n & g_1(n) & \cdots & g_k(n) \\ <y, \Delta S_n> & <y, \Delta g_1(n)> & \cdots & <y, \Delta g_k(n)> \\ \vdots & \vdots & & \vdots \\ <y, \Delta S_{n+k-1}> & <y, \Delta g_1(n+k-1)> & \cdots & <y, \Delta g_k(n+k-1)> \end{vmatrix}}{\begin{vmatrix} <y, \Delta g_1(n)> & \cdots & <y, \Delta g_k(n)> \\ \vdots & & \vdots \\ <y, \Delta g_1(n+k-1)> & \cdots & <y, \Delta g_k(n+k-1)> \end{vmatrix}}.$$

By construction we have the

Theorem 4.4

If $\forall n \geq N, S_n = S + a_1 g_1(n) + \cdots + a_k g_k(n)$ *with* $a_k \neq 0$ *then* $\forall n \geq N, E_k^{(n)} = S$.

The vectors $E_k^{(n)}$ can be recursively computed by the vector E-algorithm which is as follows

$$E_0^{(n)} = S_n\,, \quad g_{0,i}^{(n)} = g_i(n)\,, \qquad n = 0, 1, \ldots\,;\ i \geq 1.$$

Then, for $k = 1, 2, \ldots$ and $n = 0, 1, \ldots$

$$E_k^{(n)} = E_{k-1}^{(n)} - \frac{<y, E_{k-1}^{(n+1)} - E_{k-1}^{(n)}>}{<y, g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}>} \cdot g_{k-1,k}^{(n)}$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{<y, g_{k-1,i}^{(n+1)} - g_{k-1,i}^{(n)}>}{<y, g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}>} \cdot g_{k-1,k}^{(n)}\,, \qquad \text{for } i > k.$$

Obviously if (S_n) and $(g_i(n))$ are replaced by $(aS_n + b)$ and $(ag_i(n) + b)$ where a is a scalar and b a vector then $E_k^{(n)}$ and $g_{k,i}^{(n)}$ become respectively $a\,E_k^{(n)} + b$ and $a\,g_{k,i}^{(n)} + b$. If y is replaced by $a\,y$, $E_k^{(n)}$ and $g_{k,i}^{(n)}$ remain unchanged.

From the rule of the algorithm, it can be proved by induction that it holds

Theorem 4.5

If $\forall n \geq N, S_n = S + a_1 g_1(n) + a_2 g_2(n) + \cdots$ *then* $\forall n \geq N, E_k^{(n)} = S + a_{k+1} g_{k,k+1}(n) + a_{k+2} g_{k,k+2}(n) + \cdots$.

For the choice $g_i(n) = \Delta S_{n+i-1}$, we recover the generalization of Shanks' transformation studied in the preceding section.

The subroutine VEALGO performs the vector E-algorithm.

Another algorithm for implementing the vector E-algorithm and some others will be studied in section 4.6.

A particular rule for the vector E-algorithm was given by Brezinski [79]. It allows to jump over breakdowns and near-breakdowns thus avoiding division by zero and improving the numerical stability of the algorithm. Let us mention that there is a misprint in Brezinski [79]: k in the upper indexes of the last row of the numerator and of the denominator has to be replaced by m. This particular rule is as follows (Δ operating on the upper indexes)

$$E_{k+m}^{(n)} = \frac{\begin{vmatrix} E_k^{(n)} & g_{k,k+1}^{(n)} & \cdots & g_{k,k+m}^{(n)} \\ <y, \Delta E_k^{(n)}> & <y, \Delta g_{k,k+1}^{(n)}> & \cdots & <y, \Delta g_{k,k+m}^{(n)}> \\ \vdots & \vdots & & \vdots \\ <y, \Delta E_k^{(n+m-1)}> & <y, \Delta g_{k,k+1}^{(n+m-1)}> & \cdots & <y, \Delta g_{k,k+m}^{(n+m-1)}> \end{vmatrix}}{\begin{vmatrix} <y, \Delta g_{k,k+1}^{(n)}> & \cdots & <y, \Delta g_{k,k+m}^{(n)}> \\ \vdots & & \vdots \\ <y, \Delta g_{k,k+1}^{(n+m-1)}> & \cdots & <y, \Delta g_{k,k+m}^{(n+m-1)}> \end{vmatrix}}$$

and a similar relation for $g_{k+m,i}^{(n)}$ by replacing the $E_k^{(j)}$'s is the first column of the numerator by the $g_{k,i}^{(j)}$'s. For $m = 1$, this relation reduces to the rule of the E-algorithm.

Because of its generality, it will be very much interesting to have acceleration results for the vector E-algorithm.

The first result is due to Wimp [467] who proved the

Theorem 4.6

If $\forall i, \exists b_i \neq 1$ *such that* $\lim_{n\to\infty} <y, g_i(n+1)> / <y, g_i(n)> = b_i$, *if* $\forall j \neq i$, $b_j \neq b_i$ *and if* $\exists j$ *such that* $\lim_{n\to\infty} <y, S_{n+1} - S> / <y, S_n - S> = b_j$ *then* $\forall k$, $\lim_{n\to\infty} <y, E_k^{(n)} - S> / <y, S_n - S> = 0$.

Moreover if for $m \neq j$, $\|g_m(n)\| / <y, g_m(n)> = O(1)$ *and if*

$$\left\| e_n - \frac{<y, e_n>}{<y, g_j(n)>} \cdot g_j(n) \right\| = o(\|e_n\|)$$

where $e_n = S_n - S$ *then* $\lim_{n\to\infty} \left\| E_k^{(n)} - S \right\| / \|S_n - S\| = 0$.

Other results were obtained by Matos [314]. However a drawback of the generality of the algorithm is the complexity of the results.

We first need some definitions. Let $I = \{1, 2, \ldots, N\}$ or $I = \mathbb{N}$ and let p be the dimension of the vectors concerned.

We say that $\{(g_i(n)), i \in I\}$ belongs to $F_{LIN}(I, p)$ if and only if the sequences of vectors $(g_i(n))_n$ satisfy the following conditions

i) $\forall i \in I$, $\lim_{n\to\infty} g_i^j(n) = 0$ for $j = 1, \ldots, p$ where $g_i^j(n)$ is the j-th component of the real or complex vector $g_i(n)$.

ii) $\forall i \in I$, let $j_i \in \{1, 2, \ldots, p\}$ be such that $\forall j \in \{1, 2, \ldots, p\}, \exists C, N$ (both depending on i and j_i), $\forall n \geq N$,

- $\left| g_i^j(n)/g_i^{j_i}(n) \right| \leq C$,
- $\lim_{n\to\infty} g_i^{j_i}(n+1)/g_i^{j_i}(n) = b_i$ with $|b_i| < 1$,
- $\lim_{n\to\infty} g_{i+1}^{j_{i+1}}(n)/g_i^{j_i}(n) = 0$,
- $\forall k \neq i$, $b_k \neq b_i$.

We say that $\{(g_i(n)), i \in I\}$ belongs to $F_{LOG}(I, p)$ if and only if the sequences $(g_i(n))_n$ satisfy the following conditions

i) $\forall i \in I$, $\lim_{n\to\infty} g_i^j(n) = 0$ for $j = 1, \ldots, , p$.

ii) $\forall i \in I$, let $j_i \in \{1, 2, \ldots, p\}$ be such that $\forall j \in \{1, 2, \ldots, p\}, \exists C, N$ (both depending on i and j_i), $\forall n \geq N$,

- $\left| g_i^j(n)/g_i^{j_i}(n) \right| \leq C$,
- $g_i^{j_i}(n+1)/g_i^{j_i}(n) = 1 + u_i(n)$ with $u_i(n) \sim \alpha_i/n$, $\alpha_i \neq 0$, and $u_i(n)/u_k(n) = \alpha_{ik}(1 + \beta_{ik}(n))$ with $\alpha_{ik} \neq 1$, $\beta_{ik}(n) \sim a_{ik}/\ln n, \forall k < i$ and $k \in I$,
- $\lim_{n\to\infty} g_{i+1}^{j_{i+1}}(n)/g_i^{j_i}(n) = 0$,
- $\forall j \neq j_i, \exists K, M$ (both depending on i and j), $\forall n \geq M$, $\left| \Delta g_i^j(n) / \Delta g_i^{j_i}(n) \right| \leq K$ (Δ operates on n).

We have the following results

Theorem 4.7

Let us assume that $\forall n, S_n = S + \sum_{i=1}^{k} a_i g_i(n)$. *If* $\{(g_i(n))\,,\ i \in I = \{1, \ldots, k\}\} \in F_{LIN}(I,p)$ *and if* y *satisfies* $\lim_{n\to\infty} (y, g_i(n))/g_i^{j_i}(n) = \alpha_i \neq 0, \forall i \in I$ *or if* $\{(g_i(n)), i \in I\} \in F_{LOG}(I,p)$ *and if* y *satisfies* $(y, g_i(n))/g_i^{j_i}(n) = a_i\,(1 + \beta_i(n))$ *with* $\beta_i(n) \sim K_i/\ln n, \forall i \in I$, *then*

$$\lim_{n\to\infty} \frac{\left\| E_i^{(n)} - S \right\|}{\left\| E_{i-1}^{(n)} - S \right\|} = 0\,, \qquad \forall i \in I.$$

Theorem 4.8

Let us assume that $\forall n, S_n = S + \sum_{i=1}^{\infty} a_i g_i(n)$. *If the conditions of theorem 4.7 hold with* $I = \mathbb{N}$ *then* $\exists C, N, \forall n \geq N$

$$\left\| E_i^{(n)} - S \right\| \leq C \left\| g_{i+1}(n) \right\|\,, \qquad \forall i \in I.$$

There are some cases where it is easy to find a vector y satisfying the conditions of the previous theorems

i) if $\forall i \in I, j_i = m$ then y can be taken as a vector with all components equal to zero except the m-th which is one.

ii) If $\forall i \in I, \forall j \neq j_i, \lim_{n\to\infty} g_i^j(n)/g_i^{j_i}(n) = 0$ and $\{(g_i(n)), i \in I\} \in F_{LIN}(I,p)$ or if $g_i^j(n)/g_i^{j_i}(n) \sim K_{ij}/\ln n$ and $\{(g_i(n)), i \in I\} \in F_{LOG}(I,p)$ then the choice $y = (1, 1, \ldots, 1)^T$ satisfies the above conditions.

Matos [314] also compared the vector E-algorithm to the scalar one applied on each component of the vectors separately and concluded that their acceleration properties are equivalent. The only advantage of the vector E-algorithm is that it needs less arithmetical operations.

The choice of the arbitrary vector y in this algorithm is an open problem.

4.4 The recursive projection algorithm

There is a very strong connection between vector extrapolation algorithms, fixed point iterations and projection methods. It will not be developed here into details and we refer the interested reader to Sadok [382], Brezinski and Sadok [101], Brezinski [89] and the papers mentioned in their bibliographies. However let us present some algorithms for recursive projection, relate them to vector extrapolation processes and give some of their applications. Let $y, x_1, x_2, \ldots, z_1, z_2, \ldots$ be vectors. We shall use the notation $<\cdot,\cdot>$ to designate either the bilinear form or the scalar product defined in the previous sections. We consider the determinants

$$N_k = \begin{vmatrix} y & x_1 & \cdots & x_k \\ <z_1, y> & <z_1, x_1> & \cdots & <z_1, x_k> \\ \vdots & \vdots & & \vdots \\ <z_k, y> & <z_k, x_1> & \cdots & <z_k, x_k> \end{vmatrix}$$

$$D_k = \begin{vmatrix} <z_1, x_1> & \cdots & <z_1, x_k> \\ \vdots & & \vdots \\ <z_k, x_1> & \cdots & <z_k, x_k> \end{vmatrix}$$

$$N_{k,i} = \begin{vmatrix} x_i & x_1 & \cdots & x_k \\ <z_1, x_i> & <z_1, x_1> & \cdots & <z_1, x_k> \\ \vdots & \vdots & & \vdots \\ <z_k, x_i> & <z_k, x_1> & \cdots & <z_k, x_k> \end{vmatrix}.$$

We set

$$E_k = \frac{N_k}{D_k} \quad \text{and} \quad g_{k,i} = \frac{N_{k,i}}{D_k}.$$

Using an extension of Sylvester's determinantal identity to determinants whose first row consists of vectors (as in N_k and $N_{k,i}$), Brezinski [69] proved that the vectors E_k can be recursively computed by the following algorithm called the recursive projection algorithm (RPA in short)

$$E_0 = y\,, \quad g_{0,i} = x_i\,, \quad i \geq 1$$

$$E_k = E_{k-1} - \frac{<z_k, E_{k-1}>}{<z_k, g_{k-1,k}>} \cdot g_{k-1,k}, \qquad k \geq 1$$

$$g_{k,i} = g_{k-1,i} - \frac{<z_k, g_{k-1,i}>}{<z_k, g_{k-1,k}>} \cdot g_{k-1,k}, \qquad i > k \geq 1.$$

This algorithm is not very simple since, as the E-algorithm, it involves a principal rule for the E's and an auxiliary rule for the g's. For that reason let us introduce some more notations in order to propose a more compact algorithm involving only one single rule. We set $x_0 = y$,

$$N_k^{(i)} = \begin{vmatrix} x_i & x_{i+1} & \cdots & x_{i+k} \\ <z_1, x_i> & <z_1, x_{i+1}> & \cdots & <z_1, x_{i+k}> \\ \vdots & \vdots & & \vdots \\ <z_k, x_i> & <z_k, x_{i+1}> & \cdots & <z_k, x_{i+k}> \end{vmatrix}$$

$$D_k^{(i)} = \begin{vmatrix} <z_1, x_{i+1}> & \cdots & <z_1, x_{i+k}> \\ \vdots & & \vdots \\ <z_k, x_{i+1}> & \cdots & <z_k, x_{i+k}> \end{vmatrix}$$

and $e_k^{(i)} = N_k^{(i)}/D_k^{(i)}$.

Since $x_0 = y$, obviously $D_k^{(0)} = D_k$, $N_k^{(0)} = N_k$ and $N_k^{(1)} = (-1)^k N_{k,k+1}$. Thus

$$e_k^{(0)} = E_k \quad \text{and} \quad e_k^{(1)} = (-1)^k g_{k,k+1} \cdot \frac{D_k^{(0)}}{D_k^{(1)}}.$$

The vectors $e_k^{(i)}$ can be recursively computed by the following algorithm called the compact recursive projection algorithm (CRPA in short)

$$e_0^{(i)} = x_i, \qquad i \geq 0$$

$$e_k^{(i)} = e_{k-1}^{(i)} - \frac{<z_k, e_{k-1}^{(i)}>}{<z_k, e_{k-1}^{(i+1)}>} \cdot e_{k-1}^{(i+1)}, \qquad k \geq 1,\ i \geq 0.$$

Thus the RPA and the CRPA compute the same vectors recursively, namely $e_k^{(0)}$ for $k = 1, 2, \ldots$. They require the same storage and the same number of arithmetical operations and we shall distinguish them only when necessary (in particular when the $g_{k-1,k}$'s of the RPA have an

interesting meaning and since they are simpler than the corresponding $e_{k-1}^{(1)}$'s of the CRPA).

Some variants of these algorithms will be seen at the end of this section but, for the moment, let us review some of their applications and connections.

Let us set $P_k = y - E_k$ and $w_i = < z_i, y >$. Then we have

$$< z_i, P_k >= w_i\,, \qquad \text{for } i = 1, \ldots, k.$$

If now y, x_1, $x_2, \ldots$ are elements of a vector space E and if we replace $< z_i, x_i >$ by $L_i(x_j)$ where L_i is a linear functional on E (that is an element of the dual space E^*) then $P_k \in$ span $(x_1, \ldots, x_k)$ is the solution of the general interpolation problem

$$L_i(P_k) = w_i\,, \qquad \text{for } i = 1, \ldots, k.$$

and the RPA becomes

$$\begin{aligned} & P_0 = 0 \\ P_k \;&=\; P_{k-1} + \frac{w_k - L_k(P_{k-1})}{L_k(g_{k-1,k})} \cdot g_{k-1,k}\,, \qquad k \geq 1 \\ & g_{0,i} = x_i\,, \qquad i \geq 1 \\ g_{k,i} \;&=\; g_{k-1,i} - \frac{L_k(g_{k-1,i})}{L_k(g_{k-1,k})} \cdot g_{k-1,k}\,, \qquad i > k \geq 1. \end{aligned}$$

The connection between the RPA and the general interpolation problem can be fully exploited in the framework of biorthogonality as explained in Brezinski [89]. In particular, the RPA can be used for computing adjacent families of formal orthogonal polynomials and thus Padé approximants. It has also many other applications.

Let us now consider the case where $< \cdot, \cdot >$ is the usual scalar product $(\cdot, \cdot)$ and where $z_i = x_i$. Then P_k is the orthogonal projection of y on the subspace spanned by $x_1, \ldots, x_k$ and the RPA was named after this property. The P_k's are the truncated Fourier sums of y and the RPA performs the Gram-Schmidt orthonormalization of the x_i's since $\left(x_i^*, x_j^*\right) = \delta_{ij}$ where

$$x_k^* = \frac{g_{k-1,k}}{(g_{k-1,k}\,,\; g_{k-1,k})^{1/2}}.$$

In that case we have

$$\begin{aligned} & E_0 = y\,, \quad g_{0,i} = x_i\,, && i \geq 1 \\ E_k &= E_{k-1} - \frac{(y\,,\, g_{k-1,k})}{(g_{k-1,k}\,,\, g_{k-1,k})} \cdot g_{k-1,k}\,, && k \geq 1 \\ g_{k,i} &= g_{k-1,i} - \frac{(x_i\,,\, g_{k-1,k})}{(g_{k-1,k}\,,\, g_{k-1,k})} \cdot g_{k-1,k}\,, && i > k \geq 1 \end{aligned}$$

which shows that the principal rule of the RPA is identical with the projection method of Rosen [376] for nonlinear programming in which the auxiliary vectors needed are computed by the auxiliary rule of the RPA. We also have from the CRPA

$$\begin{aligned} & e_0^{(0)} = y\,, \quad e_0^{(i)} = x_i\,, && i \geq 1 \\ e_k^{(i)} &= e_{k-1}^{(i)} - \frac{(y, e_{k-1}^{(i+1)})}{(e_{k-1}^{(i+1)}, e_{k-1}^{(i+1)})} \cdot e_{k-1}^{(i+1)}\,, && i \geq 0,\ k \geq 1. \end{aligned}$$

Let us now consider the system of linear equations

$$Ax = b$$

where A is a symmetric positive definite matrix. If we set

$$y = A^{-1}b\,, \quad x_i = A^{i-1}b\,, \quad z_i = A^i b$$

then $(z_j, x_i) = (A^{i+j-1}b, b) = c_{i+j-1}$ and $(z_j, y) = (A^{i-1}b, b) = c_{i-1}$ which shows that the knowledge of A^{-1} is not needed for computing these scalar products. From the determinantal formula for E_k and that for the conjugate gradient method given by Brezinski [63] (p. 87) we see that the vectors $P_k = y - E_k$ are exactly those obtained by the conjugate gradient method. The same vectors can also be obtained by the topological ε-algorithm but less economically since this algorithm does not use the fact that for a symmetric matrix $(A^i b, A^j b) = (A^{i+j}b, b)$. In the case of a nonsymmetric matrix we can take $z_i = \left(A^T\right)^i b$. Thus the RPA is very much connected with projection methods used for solving systems of linear and nonlinear equations and it is, in particular, related to the algorithms of the ABS class as described by Abaffy and Spedicato [2], to the method of Sloboda [411] and that of Pugachev [368].

There are some variants of the CRPA which can be of interest. Using the same notations as above, we set

$$\bar{e}_k^{(i)} = \frac{N_k^{(i)}}{D_k^{(i-1)}}.$$

Then, the following recursive scheme holds

$$\begin{aligned} \bar{e}_0^{(i)} &= x_i\,, && i \geq 0 \\ \bar{e}_k^{(i)} &= \frac{< z_k, \bar{e}_{k-1}^{(i+1)} >}{< z_k, \bar{e}_{k-1}^{(i)} >} \cdot \bar{e}_{k-1}^{(i)} - \bar{e}_{k-1}^{(i+1)}\,, && i \geq 0\,,\ k \geq 1. \end{aligned}$$

Let us now set

$$\tilde{e}_k^{(i)} = \frac{N_k^{(i)}}{C_k^{(i)}}$$

where

$$C_k^{(i)} = \begin{vmatrix} 1 & \cdots & 1 \\ < z_1, x_i > & \cdots & < z_1, x_{i+k} > \\ \vdots & & \vdots \\ < z_k, x_i > & \cdots & < z_k, x_{i+k} > \end{vmatrix}.$$

Then

$$\begin{aligned} \tilde{e}_0^{(i)} &= x_i\,, && i \geq 0 \\ \tilde{e}_k^{(i)} &= \tilde{e}_{k-1}^{(i)} - \frac{< z_k, \tilde{e}_{k-1}^{(i)} >}{< z_k, \tilde{e}_{k-1}^{(i+1)} - \tilde{e}_{k-1}^{(i)} >} \cdot \left(\tilde{e}_{k-1}^{(i+1)} - \tilde{e}_{k-1}^{(i)} \right)\,, && i \geq 0,\ k \geq 1. \end{aligned}$$

These two algorithms are respectively called the first and the second variant of the CRPA.

Brezinski [79] gave a particular rule for the CRPA which can be used to jump over breakdowns or near-breakdowns. It is as follows

$$e_{k+m}^{(i)} = \frac{\begin{vmatrix} e_k^{(i)} & \cdots & e_k^{(i+m)} \\ < z_{k+1}, e_k^{(i)} > & \cdots & < z_{k+1}, e_k^{(i+m)} > \\ \vdots & & \vdots \\ < z_{k+m}, e_k^{(i)} > & \cdots & < z_{k+m}, e_k^{(i+m)} > \end{vmatrix}}{\begin{vmatrix} < z_{k+1}, e_k^{(i+1)} > & \cdots & < z_{k+1}, e_k^{(i+m)} > \\ \vdots & & \vdots \\ < z_{k+m}, e_k^{(i+1)} > & \cdots & < z_{k+m}, e_k^{(i+m)} > \end{vmatrix}}.$$

If $m = 1$, this relation reduces to the rule of the CRPA. Similar relations can be obtained for the variants of the CRPA.

The RPA, the CRPA and their variants possess quasi-linearity properties. If y and the x_i's are replaced by ay and ax_i where a is a scalar then the quantities computed by the various algorithms are also multiplied by a. If the z_i's are multiplied by a, the quantities computed by the algorithms remain unchanged. The addition of a vector b does not lead to interesting properties.

The subroutine CRPA performs the CRPA and its two variants.

4.5 The H-algorithm

Let us now consider the ratio of determinants

$$H_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(n) & \cdots & g_1(n+k) \\ \vdots & & \vdots \\ g_k(n) & \cdots & g_k(n+k) \end{vmatrix}}$$

where the S_n's are vectors and the $g_i(n)$'s scalars. The vectors $H_k^{(n)}$ can be recursively computed by the H-algorithm

$$H_0^{(n)} = S_n\,, \quad g_{0,i}^{(n)} = g_i(n)\,,$$
$$n = 0, 1, \ldots\,;\ i \geq 1$$

$$H_k^{(n)} = H_{k-1}^{(n)} - \frac{g_{k-1,k}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot \left(H_{k-1}^{(n+1)} - H_{k-1}^{(n)}\right)\,,$$
$$n = 0, 1, \ldots\,;\ k = 1, 2, \ldots$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{g_{k-1,k}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot \left(g_{k-1,i}^{(n+1)} - g_{k-1,i}^{(n)}\right)\,,$$
$$n = 0, 1, \ldots\,;\ k = 1, 2, \ldots\,;\ i > k.$$

This algorithm, first introduced by Brezinski [66], was named after Henrici since it can be used for implementing a vector sequence trans-

formation due to Henrici [225] (p. 115) which corresponds to the choice $k = p$, the dimension of the vectors, and $g_i(n) = (S_{n+1} - S_n, e_i)$ where e_i is a vector whose components are all zero except the i-th which is one. This transformation can be used for solving systems of nonlinear equations as will be explained in section 6.2.4.

The H-algorithm can also be used for implementing the vectors $\bar{e}_k(S_n)$ defined at the end of section 4.2 which corresponds to the choice $g_i(n) = < y_i, \Delta S_n >$. For $g_i(n) = (z_i, S_n)$, $H_k^{(n)}$ is identical to the vector $\tilde{e}_k^{(n)}$ of the second variant of the CRPA while for $g_i(n) =< y, \Delta S_{n+i-1} >$ it is identical to the vector $\varepsilon_{2k}^{(n)} = e_k(S_n)$ of the topological ε-algorithm.

The subroutine HALGO performs the H-algorithm.

A particular rule for jumping over breakdowns and near-breakdowns can be obtained for the H-algorithm. It is as follows

$$H_{k+m}^{(n)} = \frac{\begin{vmatrix} H_k^{(n)} & \cdots & H_k^{(n+m)} \\ g_{k,k+1}^{(n)} & \cdots & g_{k,k+1}^{(n+m)} \\ \vdots & & \vdots \\ g_{k,k+m}^{(n)} & \cdots & g_{k,k+m}^{(n+m)} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_{k,k+1}^{(n)} & \cdots & g_{k,k+1}^{(n+m)} \\ \vdots & & \vdots \\ g_{k,k+m}^{(n)} & \cdots & g_{k,k+m}^{(n+m)} \end{vmatrix}}$$

and a similar relation for $g_{k+m,i}^{(n)}$ by replacing the $H_k^{(j)}$'s in the first row of the numerator by the $g_{k,i}^{(j)}$'s.

The H-algorithm is identical to the application of the scalar E-algorithm simultaneously to each component of the vectors S_n with the same auxiliary scalar sequences $(g_i(n))$. Thus the results of section 2.1 on the scalar E-algorithm remain valid. Theorems 2.1, 2.2 and 2.3 still hold where now the a_i's are constant vectors. Theorem 2.8 remains unchanged. Theorems 2.9 and 2.10 are true with the conditions and the results holding for each component of the vectors $E_k^{(n)}$'s. The quasi-linearity properties also remain true.

Henrici's vector sequence transformation, that is for $k = p$ and $g_i(n) = (\Delta S_n, e_i)$, was studied by Sadok [383] who gave the following results

based on the formula given by Brezinski [79]

$$H_p^{(n)} = S_n - \Delta_n \left(\Delta_n^2\right)^{-1} \Delta S_n$$

where Δ_n^i is the $p \times p$ matrix with columns $\Delta^i S_n, \ldots, \Delta^i S_{n+p-1}$ for $i = 1, 2$ and $\Delta_n = \Delta_n^1$.

First of all if $H_p^{(n)}$ and $\tilde{H}_p^{(n)}$ are the vectors respectively obtained by applying Henrici's transformation to the vector sequences (S_n) and $(A\, S_n + b)$ where A is a regular matrix and b a vector, then

$$\tilde{H}_p^{(n)} = A\, H_p^{(n)} + b$$

which shows that a generalization of the property of quasi-linearity holds.

Concerning the kernel of Henrici's transformation, we have the

Theorem 4.9

Assume that $\forall n, \det\left(\Delta_n^i\right) \neq 0$ *for* $i = 1$ *and* 2. *A necessary and sufficient condition that* $\forall n, H_p^{(n)} = S$ *is that* $\forall n$,

$$a_0 (S_0 - S) + \cdots + a_p (S_{n+p} - S) = 0$$

where the a_i*'s are complex numbers with* $a_p \neq 0$ *and* $a_0 + \cdots + a_p = 1$.

Before giving an acceleration result, we need a definition. We consider a matrix A_n with columns $a_1^n, \ldots, a_p^n$. The sequence (A_n) is said to be uniformly invertible if and only if for all $n \geq N, a_i^n \neq 0$ for $i = 1, \ldots, p$ and there exists a strictly positive constant c (independent of n) such that

$$|\det A_n| \geq c \prod_{i=1}^{p} \|a_i^n\|_2$$

where $\|a_i^n\|_2 = (a_i^n, a_i^n)^{1/2}$.

The following fundamental result was proved by Sadok [383]

Theorem 4.10

Let (S_n) *be a sequence of complex vectors converging to* S *and such that*

$$S_{n+1} - S = (B + B_n) \cdot (S_n - S), \quad n = 0, 1, \ldots$$

where B and B_n are matrices with $\varrho(B) < 1$ and $\lim_{n\to\infty} B_n = 0$. If the sequence (Δ_n) is uniformly invertible then $H_p^{(n)}$ exists for n sufficiently large and

$$\lim_{n\to\infty} \frac{\left\| H_p^{(n)} - S \right\|_2}{\|S_n - S\|_2} = 0.$$

This result shows that Henrici's transformation can be considered as the generalization to vectors of Aitken's Δ^2 process since, as seen as a consequence of theorem 1.8, Aitken's process accelerates scalar sequences such that $S_{n+1} - S = (B + B_n) \cdot (S_n - S)$, $n = 0, 1, \ldots$ where B and B_n are scalars with $|B| < 1$ and $\lim_{n\to\infty} B_n = 0$. Moreover, when $p = 1$, Henrici's transformation reduces to Aitken's.

Henrici's transformation can be implemented via the H-algorithm or by QR factorization of the matrices Δ_n^2 with updating. See Sadok [383] for the details.

Let $t_i : (S_n) \longmapsto \left(t_i^{(n)}\right)$ for $i = 1$ and 2, be two vector sequence transformations. A generalization to the vector case of composite sequence transformations (see section 3.7) was given by Sadok [382].

Let Δ_n^i be the $p \times p$ matrix whose columns are $\Delta t_i^{(n)}, \ldots, \Delta t_i^{(n+p-1)}$. Then the vector composite sequence transformation $T : (S_n) \longmapsto (T_n)$ is defined by

$$T_n = t_1^{(n)} + \left(I - \Delta_n^2 \left(\Delta_n^1\right)^{-1}\right)^{-1} \cdot \left(t_2^{(n)} - t_1^{(n)}\right), \qquad n = 0, 1, \ldots$$

It can be proved that T_n is identical with $H_p^{(n)}$ if $g_i(n)$ is the i-th component of the vector $t_2^{(n)} - t_1^{(n)}$. Thus (T_n) can be computed by the H-algorithm. The following results were given by Sadok [382] (see also, Brezinski and Sadok [101])

Theorem 4.11

If $\lim_{n\to\infty} t_1^{(n)} = \lim_{n\to\infty} t_2^{(n)} = S$, if $\forall n, \Delta_n^1$ is regular and if $\exists N, \exists \gamma \in]0, 1[$ such that $\forall n \geq N$, $\left\| \Delta_n^2 \left(\Delta_n^1\right)^{-1} \right\| \leq \gamma$ then $\forall n \geq N, \Delta_n^2 - \Delta_n^1$ is regular and $\lim_{n\to\infty} T_n = S$. Moreover if $\lim_{n\to\infty} \left\| t_1^{(n)} - S \right\| / \|S_n - S\| = \lim_{n\to\infty} \left\| t_2^{(n)} - S \right\| / \|S_n - S\| = 0$ then $\lim_{n\to\infty} \|T_n - S\| / \|S_n - S\| = 0$.

As is often the case when going from dimension one to higher dimensions, there are several possible generalizations of composite sequence

transformations. Thus we can also define the vector composite sequence transformation $V : (S_n) \longmapsto (V_n)$ by

$$V_n = t_1^{(n)} - \left(T_n^2 - T_n^1\right) \cdot \left(\Delta_n^2 - \Delta_n^1\right)^{-1} \cdot \left(t_1^{(n+1)} - t_1^{(n)}\right), \quad n = 0, 1, \ldots$$

where T_n^i is the $p \times p$ matrix whose columns are $t_i^{(n)}, \ldots, t_i^{(n+p-1)}$. This transformation can be implemented by the CRPA acting on $2p$-dimensional vectors as follows

$$e_0^{(0)} = \begin{pmatrix} t_1^{(n)} \\ t_1^{(n+1)} \end{pmatrix}, \quad e_0^{(i)} = \begin{pmatrix} t_2^{(n+i-1)} - t_1^{(n+i-1)} \\ t_2^{(n+i)} - t_1^{(n+i)} \end{pmatrix}, \quad i = 1, \ldots, p$$

and then

$$e_k^{(i)} = e_{k-1}^{(i)} - \frac{\left(e_{k-1}^{(i)}, e_{p+k}\right) - \left(e_{k-1}^{(i)}, e_k\right)}{\left(e_{k-1}^{(i+1)}, e_{p+k}\right) - \left(e_{k-1}^{(i+1)}, e_k\right)} \cdot e_{k-1}^{(i+1)},$$

$$k = 1, \ldots, p\,;\ i = 0, \ldots, p-k$$

where e_i is the vector whose all components are zero except the i-th one which is equal to one. The first p components of the vector $e_p^{(0)}$ are equal to the vector V_n.

For this second vector composite sequence transformation, a convergence result similar to that of theorem 4.11 can be proved but not a similar one for acceleration.

More generally the RPA (or the CRPA) can be used for implementing Henrici's transformation. We set

$$y = \begin{pmatrix} S_n \\ S_{n+1} \end{pmatrix}, \quad x_i = \begin{pmatrix} \Delta S_{n+i-1} \\ \Delta S_{n+1} \end{pmatrix}, \quad i = 1, \ldots, p.$$

For any vector $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ of dimension $2p$ we define $< z_i, v >$ as equal to the i-th component of $v_2 - v_1$. Applying the RPA gives a vector E_k whose first p components are those of $H_p^{(n)}$.

To end this section, let us consider the particular case where $g_i(n) =$

c_{n+i-1}. Then we have

$$H_k^{(n)} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ c_n & \cdots & c_{n+k} \\ \vdots & & \vdots \\ c_{n+k-1} & \cdots & c_{n+2k-1} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ c_n & \cdots & c_{n+k} \\ \vdots & & \vdots \\ c_{n+k-1} & \cdots & c_{n+2k-1} \end{vmatrix}}$$

which shows that this transformation is a generalization to the vector case of the G-transformation discussed in section 2.4. For that reason we shall set $G_k^{(n)} = H_k^{(n)}$ and we have

$$\left(1 - \frac{r_{k+1}^{(n+1)}}{r_{k+1}^{(n)}}\right) \cdot G_{k+1}^{(n)} = G_k^{(n+1)} - G_k^{(n)} \cdot \frac{r_{k+1}^{(n+1)}}{r_{k+1}^{(n)}}$$

with $G_0^{(n)} = S_n$, where the scalars $r_i^{(n)}$ are given by the rs-algorithm. Thus the $G_k^{(n)}$'s can be computed either by this vector G-transformation or by the H-algorithm with $g_i(n) = c_{n+i-1}$.

When $c_n = < y, \Delta S_n >$ we recover the first generalization of Shanks' transformation given in section 4.2 and we have $G_k^{(n)} = e_k(S_n)$ which can be computed by the topological ε-algorithm. However the vector G-transformation is more economical in storage and number of arithmetical operations as stated by Brezinski [60].

As shown by Gander, Golub and Gruntz [172] other projections methods used for solving systems of linear and nonlinear equations can be put into the above framework and thus implemented either by the H-algorithm or by the bordering method. We have

$$H_k^{(n)} = \sum_{i=0}^{k} b_i S_{n+i}$$

where the b_i's (which depend on n and k) are solutions of the system

$$\begin{cases} b_0 + \cdots + b_k = 1 \\ b_0 g_1(n) + \cdots + b_k g_1(n+k) = 0 \\ \vdots \qquad\qquad\qquad\qquad \vdots \\ b_0 g_k(n) + \cdots + b_k g_k(n+k) = 0. \end{cases}$$

If $g_i(n) = < y, \Delta S_{n+i-1} >$ we recover the first generalization of Shanks' transformation given in section 4.2. If $g_i(n) = (\Delta S_{n+i-1}, \Delta S_n)$ we obtain the minimal polynomial extrapolation method (MPE). For $g_i(n) = (\Delta^2 S_{n+i-1}, \Delta S_n)$ we have the reduced rank extrapolation method (RRE) and for $g_i(n) = (y_i, \Delta S_n)$ the modified minimal polynomial extrapolation method (MMPE). Other algorithms for implementing these methods will be described in the next section where references are given.

4.6 The Ford-Sidi algorithms

In section 2.1, we studied an algorithm obtained by Ford and Sidi [162] for implementing more economically the E-transformation. Ford and Sidi [163] generalized this algorithm to the vector case thus obtaining several interesting algorithms. We set

$$G_k^{n,m} = \begin{vmatrix} g_m(n) & \cdots & g_m(n+k-1) \\ \vdots & & \vdots \\ g_{m+k-1}(n) & \cdots & g_{m+k-1}(n+k-1) \end{vmatrix}$$

$$f_k^{n,m}(b) = \begin{vmatrix} b_n & \cdots & b_{n+k} \\ g_m(n) & \cdots & g_m(n+k) \\ \vdots & & \vdots \\ g_{m+k-1}(n) & \cdots & g_{m+k-1}(n+k) \end{vmatrix}.$$

When the b_n's are scalars then $f_k^{n,m}(b)$ is also a scalar and when they are vectors it is a vector. We set

$$G_0^{n,m} = 1 \quad \text{and} \quad f_0^{n,m}(b) = b_n\,,$$

and we define the following ratios of determinants

$$S_k^{n,m}(b) = \frac{f_k^{n,m}(b)}{f_k^{n,m}(I)}$$

where $I = (1, \ldots, 1)$ and

$$T_k^{n,m}(b) = \frac{f_k^{n,m}(b)}{G_k^{n+1,m}}.$$

We have

$$S_k^{n,m}(b) = \frac{T_k^{n,m}(b)}{T_k^{n,m}(I)}.$$

This formalism covers many vector sequence transformations and projection methods; let $S = (S_n)$ be a sequence of vectors, then

i) if $g_i(n) = (\Delta S_n, y_{i+1})$ where the y_i's are arbitrary vectors then $S_k^{n,0}(S)$ are the vectors obtained by the modified minimal polynomial extrapolation method of Sidi, Ford and Smith [408].

ii) if $g_i(n) =< y, \Delta S_{n+m} >$ then we recover the topological ε-algorithm, $S_k^{n,0}(S) = e_k(S_n) = \varepsilon_{2k}^{(n)}$.

iii) if $g_i(n) = (\Delta S_i, \Delta S_n)$ then $S_k^{n,n}(S)$ are the vectors obtained by the minimal polynomial extrapolation method of Cabay and Jackson [109].

iv) if $g_i(n) = (\Delta^2 S_i, \Delta S_n)$ then $S_k^{n,n}(S)$ are the vectors given by the reduced rank extrapolation method of Eddy [149] and Mešina [319].

v) $S_k^{n,1}(S) = H_k^{(n)}$ of the H-algorithm.

vi) if $g_i(n) =< z_i, S_n >$ then $S_k^{n,1}(S) = \tilde{e}_k^{(n)}$ obtained by the second generalization of the CRPA and $T_k^{n,1}(S) = e_k^{(n)}$ given by the CRPA.

vii) if $b_0 = S_n, b_i = G_i(n)$ for $i = 1, \ldots, k$, $g_{m+i}(0) =< y, \Delta S_{n+i} >$, $g_{m+i}(j) =< y, \Delta G_j(n+i) >$, for $i = 0, \ldots, k-1$ and $j = 1, \ldots, k$ then $T_k^{0,m}(S)$ is identical to the vector $E_k^{(n)}$ obtained from the vector E-algorithm with $E_0^{(n)} = S_n$ and $g_{0,i}^{(n)} = G_i(n)$.

Thus this formalism covers all the vector transformations previously studied. We shall now see how to compute the $S_k^{n,m}(S)$'s and the $T_k^{n,m}(S)$'s when m is fixed and also when m varies with n as $m = n + q$ where q is a fixed integer.

Let us set $g_i = (g_i(0), g_i(1), \ldots)$. For a fixed value of m we have

$$\begin{aligned} S_k^{n,m}(b) &= \frac{S_{k-1}^{n,m}(b) - c_k^n S_{k-1}^{n+1,m}(b)}{1 - c_k^n} \\ T_k^{n,m}(b) &= T_{k-1}^{n,m}(b) - d_k^n T_{k-1}^{n+1,m}(b) \end{aligned}$$

where

$$\begin{aligned} c_k^n &= \frac{S_{k-1}^{n,m}(g_{m+k-1})}{S_{k-1}^{n+1,m}(g_{m+k-1})} \\ d_k^n &= \frac{T_{k-1}^{n,m}(g_{m+k-1})}{T_{k-1}^{n+1,m}(g_{m+k-1})}. \end{aligned}$$

In the case where $g_i(n) =< z_i, S_n >$, $T_k^{n,m}(S)$ is more economically computed by the CRPA.

Let us now consider the case where $m = n+q$, q being a fixed integer. For simplification and without any ambiguity we set

$$\begin{aligned} S_k^n(b) &= S_k^{n,n+q}(b), & T_k^n(b) &= T_k^{n,n+q}(b), \\ \tilde{S}_k^n(b) &= S_k^{n,n+q-1}(b) \quad \text{and} & \tilde{T}_k^n(b) &= T_k^{n,n+q-1}(b). \end{aligned}$$

Then we have

$$\begin{aligned} S_k^n(b) &= \frac{S_{k-1}^n(b) - c_k^n \tilde{S}_{k-1}^{n+1}(b)}{1 - c_k^n} \\ \tilde{S}_k^n(b) &= \frac{S_{k-1}^n(b) - \tilde{c}_k^n \tilde{S}_{k-1}^{n+1}(b)}{1 - \tilde{c}_k^n} \\ T_k^n(b) &= T_{k-1}^n(b) - d_k^n \tilde{T}_{k-1}^{n+1}(b) \\ \tilde{T}_k^n(b) &= T_{k-1}^n(b) - \tilde{d}_k^n \tilde{T}_{k-1}^{n+1}(b) \end{aligned}$$

with

$$\begin{aligned} c_k^n &= \frac{S_{k-1}^n(g_{n+q+k-1})}{\tilde{S}_{k-1}^{n+1}(g_{n+q+k-1})} \\ \tilde{c}_k^n &= \frac{S_{k-1}^n(g_{n+q-1})}{\tilde{S}_{k-1}^{n+1}(g_{n+q-1})} \\ d_k^n &= \frac{T_{k-1}^n(g_{n+q+k-1})}{\tilde{T}_{k-1}^{n+1}(g_{n+q+k-1})} \\ \tilde{d}_k^n &= \frac{T_{k-1}^n(g_{n+q-1})}{\tilde{T}_{k-1}^{n+1}(g_{n+q-1})}. \end{aligned}$$

The following four-term recurrence relations also hold (there is a misprint in the formulæ given in Ford and Sidi [163]. The corrected formulæ were sent to us by the authors)

$$S_{k+1}^n(b) = \frac{\begin{vmatrix} S_k^n(b) & S_{k-1}^{n+1}(b) & S_k^{n+1}(b) \\ S_k^n(g_{n+q}) & S_{k-1}^{n+1}(g_{n+q}) & S_k^{n+1}(g_{n+q}) \\ S_k^n(g_{n+q+k}) & S_{k-1}^{n+1}(g_{n+q+k}) & S_k^{n+1}(g_{n+q+k}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ S_k^n(g_{n+q}) & S_{k-1}^{n+1}(g_{n+q}) & S_k^{n+1}(g_{n+q}) \\ S_k^n(g_{n+q+k}) & S_{k-1}^{n+1}(g_{n+q+k}) & S_k^{n+1}(g_{n+q+k}) \end{vmatrix}}$$

$$T_{k+1}^n(b) = \frac{\begin{vmatrix} T_k^n(b) & T_{k-1}^{n+1}(b) & T_k^{n+1}(b) \\ T_k^n(g_{n+q}) & T_{k-1}^{n+1}(g_{n+q}) & T_k^{n+1}(g_{n+q}) \\ T_k^n(g_{n+q+k}) & T_{k-1}^{n+1}(g_{n+q+k}) & T_k^{n+1}(g_{n+q+k}) \end{vmatrix}}{\begin{vmatrix} T_{k-1}^{n+1}(g_{n+q}) & T_k^{n+1}(g_{n+q}) \\ T_{k-1}^{n+1}(g_{n+q+k}) & T_k^{n+1}(g_{n+q+k}) \end{vmatrix}}.$$

Finally when n is fixed and m varies we have

$$\begin{aligned} S_k^{n,m}(b) &= \frac{S_{k-1}^{n,m}(b) - c_k^m S_k^{n,m-1}(b)}{1 - c_k^m} \\ T_k^{n,m}(b) &= \frac{T_{k-1}^{n,m}(b) - d_k^m T_k^{n,m-1}(b)}{1 - d_k^m} \end{aligned}$$

with

$$\begin{aligned} c_k^m &= \frac{S_{k-1}^{n,m}(g_{m+k-1})}{S_k^{n,m-1}(g_{m+k-1})} \\ d_k^m &= \frac{T_{k-1}^{n,m}(g_{m+k-1})}{T_k^{n,m-1}(g_{m+k-1})}. \end{aligned}$$

Some special cases are treated by Ford and Sidi [163]. They also compare the number of arithmetical operations of the various algorithms.

4.7 Miscellaneous algorithms

In this section we shall review some extensions to the vector case of various scalar algorithms studied in the second chapter.

The first of this algorithm is an extension of the Θ-algorithm due to Brezinski [51]. Its rules are as follows

$$\begin{aligned}
&\Theta_{-1}^{(n)} = 0\,, \quad \Theta_0^{(n)} = S_n\,, && n = 0,1,\ldots \\
\Theta_{2k+1}^{(n)} &= \Theta_{2k-1}^{(n+1)} + \frac{y}{<y, \Delta\Theta_{2k}^{(n)}>}\,, && k,n = 0,1,\ldots \\
\Theta_{2k+2}^{(n)} &= \Theta_{2k}^{(n+1)} + w_k^{(n)}\, D_{2k+1}^{(n)}\,, && k,n = 0,1,\ldots
\end{aligned}$$

with

$$\begin{aligned}
w_k^{(n)} &= -\frac{<z, \Delta\Theta_{2k}^{(n+1)}>}{<z, \Delta D_{2k+1}^{(n)}>} \\
D_{2k+1}^{(n)} &= \frac{\Delta\Theta_{2k}^{(n)}}{<\Delta\Theta_{2k+1}^{(n)}, \Delta\Theta_{2k}^{(n)}>}
\end{aligned}$$

where y and z are two arbitrary vectors such that the denominators do not vanish. This algorithm will be called the generalized Θ-algorithm (GTH).

Other generalizations of scalar algorithms were proposed by Osada [348]. They use either the bilinear form

$$<x, y> = \sum_{i=1}^{p} x_i y_i$$

where $x_1, \ldots, x_p$ and $y_1, \ldots, y_p$ are the components of the vectors x and $y \in \mathbb{C}^p$ respectively or the usual scalar product

$$(x, y) = \sum_{i=1}^{p} x_i \bar{y}_i.$$

In the last case we recall that the inverse x^{-1} of the vector x is defined as

$$x^{-1} = \frac{x}{(x, x)}.$$

The generalizations due to Osada are the following. First an extension, called the Euclidean W transform (EWT) of Lubkin's W transformation

$$W_0^{(n)} = S_n\,, \qquad n = 0,1,\ldots$$

$$W_k = W_{k-1}^{(n-1)} - \frac{<\Delta W_{k-1}^{(n-1)}, \Delta W_{k-1}^{(n-3)}>}{<\Delta W_{k-1}^{(n-1)}, \Delta^2 W_{k-1}^{(n-3)}> - <\Delta W_{k-1}^{(n-3)}, \Delta^2 W_{k-1}^{(n-2)}>} \cdot \Delta W_{k-1}^{(n-2)},$$
$$k = 1, 2, \ldots ; n = 3k, 3k+1, \ldots$$

Then a vector W transform (VWT) whose rules are

$$W_0^{(n)} = S_n , \qquad n = 0, 1, \ldots$$
$$W_k = W_{k-1}^{(n-1)} + \left(1 - \frac{\left(\Delta W_{k-1}^{(n-2)}, \Delta W_{k-1}^{(n-3)}\right)}{\left(\Delta W_{k-1}^{(n-3)}, \Delta W_{k-1}^{(n-3)}\right)}\right) \cdot \left(\left(\Delta W_{k-1}^{(n-1)}\right)^{-1} - 2\left(\Delta W_{k-1}^{(n-2)}\right)^{-1} + \left(\Delta W_{k-1}^{(n-3)}\right)^{-1}\right)^{-1},$$
$$k = 1, 2, \ldots ; n = 3k, 3k+1, \ldots$$

Theoretical results similar to those given at the end of section 4.1 hold for the sequence $\left(W_1^{(n)}\right)$ obtained by the EWT.

In the GTH the scalar product can be used instead of the bilinear form thus leading to the vector Θ-algorithm (VTH) which is

$$\Theta_{-1}^{(n)} = 0 , \quad \Theta_0^{(n)} = S_n , \qquad n = 0, 1, \ldots$$
$$\Theta_{2k+1}^{(n)} = \Theta_{2k-1}^{(n+1)} + \left(\Delta\Theta_{2k}^{(n)}\right)^{-1} , \qquad k, n = 0, 1, \ldots$$
$$\Theta_{2k+2}^{(n)} = \Theta_{2k}^{(n+1)} + \frac{\left(\Delta\Theta_{2k+1}^{(n+1)}, \Delta\Theta_{2k+1}^{(n)}\right)}{\left(\Delta^2\Theta_{2k+1}^{(n)}, \Delta^2\Theta_{2k+1}^{(n)}\right)} \cdot \Delta\Theta_{2k}^{(n+1)} , \qquad k, n = 0, 1, \ldots$$

The scalar ϱ-algorithm can also be generalized in two different ways. First a vector ϱ-algorithm (VRA) whose rules are

$$\varrho_{-1}^{(n)} = 0 , \quad \varrho_0^{(n)} = S_n , \qquad n = 0, 1, \ldots$$
$$\varrho_{k+1}^{(n)} = \varrho_{k-1}^{(n+1)} + (k+1) \cdot \left(\Delta\varrho_k^{(n)}\right)^{-1} , \qquad k, n = 0, 1, \ldots$$

Then a topological ϱ-algorithm (TRA) which is

$$\varrho_{-1}^{(n)} = 0 , \quad \varrho_0^{(n)} = S_n , \qquad n = 0, 1, \ldots$$
$$\varrho_{2k+1}^{(n)} = \varrho_{2k-1}^{(n+1)} + \frac{(2k+1)\, y}{<y, \Delta\varrho_{2k}^{(n)}>} , \qquad k, n = 0, 1, \ldots$$

$$\varrho_{2k+2}^{(n)} = \varrho_{2k}^{(n+1)} + \frac{(2k+2)\,\Delta\varrho_{2k}^{(n+1)}}{<\Delta\varrho_{2k}^{(n+1)}, \Delta\varrho_{2k+1}^{(n)}>}, \qquad k, n = 0, 1, \ldots$$

Of course in the case of a vector sequence any scalar algorithm can always be applied componentwise.

Let us give a numerical example to illustrate these algorithms.

We consider the system of equations

$$\begin{cases} x + xy + y^2 &= 0 \\ x^2 - 2x + y^2 &= 0 \\ x + z^2 &= 0 \end{cases}$$

whose unique real solution is $x = y = z = 0$ and whose Jacobian at the solution is

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

Thus Newton's method will converge only linearly. Starting from $x_0 = 0.1$, $y_0 = 0.5$ and $z_0 = 1$ we obtain the following results ($-\log_{10}$ of the supremum norm of the error) with $y = z = (1.5, 2.5, 3.5)^T$

k	iter.	VEA	MPE	EWT	VWT	W	GTH	VTH	Θ
1	0.30								
2	0.60	1.09	0.32						
3	0.91	2.47	3.42						
4	1.21	4.99	5.49	3.67	2.88	2.98		3.73	2.98
5	1.51	5.49	7.50	5.03	4.19	4.33	3.82	5.04	4.33
6	1.81	6.29	9.72	6.67	5.81	5.97	5.48	6.67	5.97
7	2.11	9.17	12.24	8.61	7.75	7.92	7.43	8.61	7.92
8	2.41	11.02	15.10	10.85	10.00	10.13	9.69	10.85	10.17
9	2.71	13.76	18.24	13.40	12.55	12.72	12.24	13.40	12.72
10	3.01	15.72	18.66	16.26	15.40	15.58	15.10	16.26	15.58

To end this section let us mention the following generalization of Aitken's process which can be found, for example, in Křížek and Neittaanmäki [268]

$$T_n = S_{n+1} + \frac{\Delta S_n}{1 - q_n}, \qquad n = 1, 2, \ldots$$

with $q_n = \frac{1}{p}\sum_{i=1}^{p} \frac{(\Delta S_n)_i}{(\Delta S_{n-1})_i}$ and where $(\Delta S_n)_i$ is the i-th component of the vector ΔS_n.

These algorithms deserve further theoretical studies.

In his thesis, Germain-Bonne [182] introduced a vector sequence transformation which generalizes Aitken's Δ^2 process but which is different from the preceding ones. He considered the ratios of determinants

$$T_k^{(n)} = \frac{\begin{vmatrix} S_n & S_{n+1} & \cdots & S_{n+k} \\ (\Delta S_n, \Delta S_n) & (\Delta S_n, \Delta S_{n+1}) & \cdots & (\Delta S_n, \Delta S_{n+k}) \\ \vdots & \vdots & & \vdots \\ (\Delta S_{n+k-1}, \Delta S_n) & (\Delta S_{n+k-1}, \Delta S_{n+1}) & \cdots & (\Delta S_{n+k-1}, \Delta S_{n+k}) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ (\Delta S_n, \Delta S_n) & (\Delta S_n, \Delta S_{n+1}) & \cdots & (\Delta S_n, \Delta S_{n+k}) \\ \vdots & \vdots & & \vdots \\ (\Delta S_{n+k-1}, \Delta S_n) & (\Delta S_{n+k-1}, \Delta S_{n+1}) & \cdots & (\Delta S_{n+k-1}, \Delta S_{n+k}) \end{vmatrix}}.$$

These vectors can be recursively computed via the RPA by setting

$$y = \begin{pmatrix} S_n \\ S_{n+1} \end{pmatrix}, \quad x_i = \begin{pmatrix} \Delta S_{n+i-1} \\ \Delta S_{n+i} \end{pmatrix}, \quad z_i = \begin{pmatrix} -\Delta S_{n+i-1} \\ \Delta S_{n+i-1} \end{pmatrix}$$

and $(z_j, x_i) = (\Delta S_{n+j-1}, \Delta S_{n+i}) - (\Delta S_{n+j-1}, \Delta S_{n+i-1})$. Thus we are working with vectors of $2p$ components and the first p components of the vectors E_k given by the RPA will be the vectors $T_k^{(n)}$.

The case $k = 1$ was rediscovered by Zienkiewicz and Löhner [488] and it was used for accelerating *dynamic* and *viscous* relaxation procedures in problems coming from finite elements.

Another generalization of Aitken's process was studied by Graves-Morris [193] who used

$$T_n = S_{n+1} - \frac{(\Delta S_n, \Delta S_n)}{(\Delta S_n, \Delta^2 S_n)} \cdot \Delta S_{n+1}, \qquad n = 0, 1, \ldots$$

This formula is a kind of mixture between the vector ε-algorithm and the vector-valued Padé approximants and its study was motivated by its numerical performances.

Other generalizations of the form

$$T_n = S_n + \alpha_n \cdot \Delta S_n$$

can also be found in the literature. The case $\alpha_n = -(\Delta S_n, \Delta^2 S_n)/(\Delta^2 S_n, \Delta^2 S_n)$ was studied by Irons and Tuck [237] and the choice $\alpha_n = -\|\Delta S_n\|_1 / \|\Delta^2 S_n\|_1$ was made by Karrholm [256].

Chapter 5

CONTINUOUS PREDICTION ALGORITHMS

Up to now we were concerned with the problem of finding an estimation of $S = \lim_{n\to\infty} S_n$ from $S_n, S_{n+1}, \ldots, S_{n+k}$ or, equivalently, from $S_n, \Delta S_n, \ldots, \Delta^k S_n$. We shall now be interested in obtaining an approximation of $S = \lim_{t\to\infty} f(t)$ from $f(t), f'(t), \ldots, f^{(k)}(t)$. This is the so-called continuous prediction (or extrapolation) problem and for that purpose we shall either use the algorithms given in the previous chapters for the sequence (S_n) defined by $\Delta^n S_0 = f^{(n)}(t)$ (or another related definition) or their so-called confluent forms. Such a procedure for predicting $S = \lim_{t\to\infty} f(t)$ has to be compared to the previous algorithms when applied to $S_n = f(t_n)$ where (t_n) tends to infinity. Of course, by a change of variable, the continuous prediction algorithms could be used for calculating $S = \lim_{t\to a} f(t)$. As we shall see in chapter 6, they have applications in the computation of Cauchy's principal values and improper integrals, among others. The idea of continuous prediction algorithms is due to Wynn [472] who first obtained the confluent form of the ε-algorithm. However in order to explain the process followed by Wynn we shall make use of a simpler example, namely Richardson extrapolation process and we shall see that its confluent form is the Taylor expansion. Then we shall study the confluent forms of Overholt's process, the scalar and topological ε-algorithms, the ϱ-algorithm, the Θ-algorithm and procedure, the G-transform and finally the E-algorithm.

Instead of producing a set of sequences, continuous prediction algorithms will produce a set of functions and we shall say that the function G *converges* to S *faster* than the function F if

$$\lim_{t\to\infty} (G(t) - S)/(F(t) - S) = 0.$$

Of course we shall assume that the function f to which a continuous

prediction algorithm is applied is sufficiently differentiable.

The construction of continuous prediction algorithms can be done by two different ways. The first one will be explained in section 5.1. It consists in modifying the rules of a sequence transformation and obtaining a function transformation called the confluent form of the corresponding algorithm for sequences. The theoretical study of such a function transformation is then often a quite difficult task since, as for some sequence transformations, the starting point is the rule of the algorithm itself. The second approach consists in assuming that some relation of the form

$$R\left(f(t), f'(t), \ldots, f^{(q)}(t), S\right) = 0$$

holds for all t and then to compute S. This approach is similar to the approach described in section 1.2 for sequences although less developed. The only quite general results in this domain are those given by Brezinski [42] which are quite similar to the results due to Pennacchi [355] and explained at the beginning of section 1.11. Both approaches will be used in the sequel. The notion of kernel introduced in the first chapter can be easily extended to continuous prediction algorithms. Let $T : f \longmapsto g$ be a function transformation. The kernel $\mathcal{K}_T$ of T is the set of functions f such that $\forall t, g(t)$ is constant.

5.1 The Taylor expansion

We shall illustrate the process followed by Wynn for obtaining the confluent form of the ε-algorithm from the scalar ε-algorithm on a simpler example, namely Richardson extrapolation process as explained in section 2.2. This process can be written as

$$\begin{aligned} T_0^{(n)} &= S_n \\ T_{k+1}^{(n)} &= T_k^{(n)} + x_n \cdot \frac{T_k^{(n)} - T_k^{(n+1)}}{x_{n+k+1} - x_n}, \qquad k, n = 0, 1, \ldots \end{aligned}$$

Let us replace x_n by $t + nh$ and assume that $T_k^{(n)}$ represents the value of the function T_k at the point x_n. Setting $f(x_n) = S_n$ we obtain

$$\begin{aligned} T_0(t + nh) &= f(t + nh) \\ T_{k+1}(t + nh) &= T_k(t + nh) - \frac{t + nh}{k + 1} \cdot \frac{T_k\left(t + (n+1)h\right) - T_k(t + nh)}{h}. \end{aligned}$$

Letting now h tend to zero gives the confluent form of Richardson extrapolation process

$$\begin{aligned} T_0(t) &= f(t) \\ T_{k+1}(t) &= T_k(t) - \frac{t}{k+1} \cdot T_k'(t), \qquad k = 0, 1, \ldots \end{aligned}$$

It is easy to see that

$$T_k(t) = f(t) - tf'(t) + \cdots + \frac{(-1)^k}{k!} \cdot t^k f^{(k)}(t)$$

which is the truncated Taylor expansion of f around zero (that is with $h = -t$). Thus we don't obtain any new interesting algorithm but this example clearly shows how to obtain the confluent form of a sequence transformation by replacing the discrete variable n by the continuous variable $x = t + nh$ and then letting h tend to zero.

5.2 Confluent Overholt's process

Let us assume that, for all t, f satisfies

$$f(t) = S + \sum_{i=1}^{\infty} a_i \, [f'(t)]^i .$$

The problem is to find approximations of S. Proceeding as in the scalar case, leads to the confluent form of Overholt's process which is as follows

$$\begin{aligned} V_0(t) &= f(t) \\ V_{k+1}(t) &= V_k(t) - \frac{f'(t)}{f''(t)} \cdot \frac{V_k'(t)}{k+1}, \qquad k = 0, 1, \ldots \end{aligned}$$

and we have the

Theorem 5.1

If $f(t) = S + \sum_{i=1}^{\infty} a_i \, [f'(t)]^i$ *then* $\forall k \geq 1$

$$V_k(t) = S + \sum_{i=k+1}^{\infty} a_i \cdot \left(1 - \frac{i}{k}\right) \cdot \left(1 - \frac{i}{k-1}\right) \cdot \ldots \cdot (1 - i) \cdot [f'(t)]^i .$$

Let us now assume that the confluent Overholt's process is applied to a function f such that $\lim_{t\to\infty} f(t) = S$. Then we immediately have the

Theorem 5.2

A sufficient condition that $\forall k \geq 1$, $\lim_{t\to\infty} V_k(t) = S$ *is that* $\exists \varepsilon > 0$ *such that* $\forall t \geq T, |f''(t)/f'(t)| \geq \varepsilon$.

A necessary and sufficient condition that V_{k+1} *converges to* S *faster than* V_k *is that*

$$\lim_{t\to\infty} \frac{f'(t)}{f''(t)} \cdot \frac{V_k'(t)}{V_k(t) - S} = k + 1.$$

The main drawback of the confluent form of Overholt's process is that V_k must be obtained in closed form and then differentiated which means, most of the time, that computer algebra has to be used.

5.3 Confluent ε-algorithms

The confluent form of the scalar ε-algorithm was obtained by Wynn [472] as explained in section 5.1. The discrete variable n is replaced by $x = t + nh$, $\varepsilon_{2k}^{(n)}$ is replaced by $\varepsilon_{2k}(x)$ and $\varepsilon_{2k+1}^{(n)}$ by $h^{-1}\varepsilon_{2k+1}(x)$. Then we let h tend to zero and we obtain the confluent form of the scalar ε-algorithm

$$\begin{aligned} \varepsilon_{-1}(t) = 0\,, \quad & \varepsilon_0(t) = f(t) \\ \varepsilon_{k+1}(t) \;=\; \varepsilon_{k-1}(t) + \frac{1}{\varepsilon_k'(t)}\,, \quad & k = 0, 1, \ldots \end{aligned}$$

This algorithm is quasi-linear which means that if $f(t)$ is replaced by $af(t)+b$ where $a \neq 0$ and b are constants then $\varepsilon_{2k}(t)$ becomes a $\varepsilon_{2k}(t)+b$ and $\varepsilon_{2k+1}(t)$ is replaced by $\varepsilon_{2k+1}(t)/a$.

As said in the introduction of this chapter, the starting point for the study of the confluent ε-algorithm is the rule of the algorithm. So we know from our experience with scalar algorithms that it is important to be able to express the $\varepsilon_k(t)$'s as ratios of determinant. Let us set

$$\begin{aligned} H_0^{(n)}(t) \;&=\; 1 \\ H_k^{(n)}(t) \;&=\; \begin{vmatrix} f^{(n)}(t) & f^{(n+1)}(t) & \ldots & f^{(n+k-1)}(t) \\ \vdots & \vdots & & \vdots \\ f^{(n+k-1)}(t) & f^{(n+k)}(t) & \ldots & f^{(n+2k-2)}(t) \end{vmatrix}. \end{aligned}$$

These functional Hankel determinants satisfy the same recurrence relation that the Hankel determinants defined in section 2.3

$$H_0^{(n)}(t)=1\,,\quad H_1^{(n)}(t)=f^{(n)}(t),\qquad n=0,1,\dots$$
$$H_{k+1}^{(n)}(t)\cdot H_{k-1}^{(n+2)}(t) = H_k^{(n)}(t)\cdot H_k^{(n+2)}(t)-\left[H_k^{(n+1)}(t)\right]^2,$$
$$k=1,2,\dots;\ n=0,1,\dots$$

It was proved by Wynn [472] that we have the

Theorem 5.3

$$\varepsilon_{2k}(t)=\frac{H_{k+1}^{(0)}(t)}{H_k^{(2)}(t)}\,,\qquad \varepsilon_{2k+1}(t)=\frac{H_k^{(3)}(t)}{H_{k+1}^{(1)}(t)}.$$

As in the scalar case the $\varepsilon_{2k+1}(t)$'s are intermediate computations without interest. Thus we see that the computation of the $\varepsilon_k(t)$'s can be conducted by, at least, five different methods. First, the rule of the confluent ε-algorithm can be used. However it must be noticed that, as was the case for the confluent form of Overholt's process, it implies that the closed form of $\varepsilon_k(t)$ is known and then differentiated (by computer algebra). The second method consists in using the result of theorem 5.3 and computing recursively the functional Hankel determinants as explained above. The third method is based on the fact that the ratio of determinants for $\varepsilon_{2k}(t)$ is exactly the same as the ratio for $\omega_{2k}^{(0)}$ in the ω-algorithm (see section 2.3) with $a_n=f^{(n)}(t)$. In fact the ω-algorithm was derived by Wynn [484] for implementing the confluent form of the scalar ε-algorithm. Let us consider the system of linear equations

$$\begin{cases} a_0f(t)+a_1f'(t)+\cdots+a_kf^{(k)}(t) &= 1\\ a_0f'(t)+a_1f''(t)+\cdots+a_kf^{(k+1)}(t) &= 0\\ \vdots & \vdots\\ a_0f^{(k)}(t)+a_1f^{(k+1)}(t)+\cdots+a_kf^{(2k)}(t) &= 0. \end{cases}$$

Then $a_0=H_k^{(2)}(t)\,/H_{k+1}^{(0)}(t)$ which shows that $\varepsilon_{2k}(t)=1/a_0$. Thus the $\varepsilon_{2k}(t)$'s can be obtained recursively by solving this system by the bordering method explained in section 1.8.

Finally another method for computing $\varepsilon_{2k}(t)$ is to apply the scalar ε-algorithm to the sequence (S_n) defined by $\Delta^nS_0=f^{(n)}(t)$ for $n=0,1,\dots$

and we shall obtain $\varepsilon_{2k}^{(0)} = \varepsilon_{2k}(t)$. Thus, of course, the other algorithms for implementing Shanks' transformation can also be used: G-transform, E-algorithm, ...

The following relations were also proved to hold

$$\begin{aligned}
\varepsilon_{2k+2}(t) &= \varepsilon_{2k}(t) - \frac{\left[H_{k+1}^{(1)}(t)\right]^2}{H_k^{(2)}(t) \cdot H_{k+1}^{(2)}(t)} \\
\varepsilon'_{2k}(t) &= \frac{H_k^{(1)}(t) \cdot H_{k+1}^{(1)}(t)}{\left[H_k^{(2)}(t)\right]^2} \\
\varepsilon'_{2k+1}(t) &= -\frac{H_k^{(2)}(t) \cdot H_{k+1}^{(2)}(t)}{\left[H_{k+1}^{(1)}(t)\right]^2}.
\end{aligned}$$

Let us now study the kernel of the confluent form of the ε-algorithm. We have the

Theorem 5.4

A necessary and sufficient condition that $\forall t \geq T$, $\varepsilon_{2k}(t) = S$ *is that* $\forall t \geq T$

$$f(t) = S + a_1 f'(t) + \cdots + a_k f^{(k)}(t)$$

where the a_i*'s are constants independent of* t *or, in other words, that*

$$f(t) = S + \sum_{i=1}^{p} A_i(t) e^{r_i t} + \sum_{i=p+1}^{q} \left[B_i(t) \cos b_i t + C_i(t) \sin b_i t\right] e^{r_i t}$$

with $r_i \neq 0$ *for* $i = 1, \ldots, p$, *where* A_i, B_i *and* C_i *are polynomials in* t *such that, if* d_i *is equal to the degree of* A_i *plus one for* $i = 1, \ldots, p$ *and to the maximum of the degrees of* B_i *and* C_i *plus one for* $i = p+1, \ldots, q$, *one has*

$$\sum_{i=1}^{p} d_i + 2 \sum_{i=p+1}^{q} d_i = k.$$

This result is similar to that of theorem 2.18 for the scalar ε-algorithm. Some other algebraic properties of the confluent form of the ε-algorithm can be found in Brezinski [55]. Let us now give some convergence results. Such results are only known for totally monotonic functions. We say

that the function f is totally monotonic (in $[T, \infty)$ to be more precise) if $\forall t \geq T$,

$$(-1)^k f^{(k)}(t) \geq 0\,, \qquad k = 0, 1, \ldots$$

Such functions are also called, sometimes, completely monotonic.

Obviously if f is totally monotonic, there exists S such that $S = \lim_{t \to \infty} f(t)$.

It is known, see Widder [462] for example, that a necessary and sufficient condition that f be totally monotonic in $[0, \infty)$ is that there exists α bounded and non decreasing in $[0, \infty)$ such that

$$f(t) = \int_0^\infty e^{-xt} d\alpha(t)\,, \qquad \forall t \in [0, \infty).$$

Moreover $\forall k \geq 0$ and $\forall t \geq T, H_k^{(0)}(t) \geq 0$. But, if f is totally monotonic, then so is $(-1)^k f^{(k)}$ and thus $(-1)^{kn} H_k^{(n)}(t) \geq 0, \forall k, n \geq 0$. From these inequalities the sign of $\varepsilon_k(t)$'s can be obtained and we have the following result, Brezinski [38]

Theorem 5.5

If the confluent form of the ε-algorithm is applied to a totally monotonic function f then $\forall k \geq 0$ and $\forall t \geq T$

$$\begin{aligned}
&\varepsilon_{2k}(t) \geq 0\,, \qquad \varepsilon_{2k+1}(t) \leq 0\\
&\varepsilon'_{2k}(t) \leq 0\,, \qquad \varepsilon'_{2k+1}(t) \leq 0\\
&0 \leq \varepsilon_{2k+2}(t) \leq \varepsilon_{2k}(t)\\
&\varepsilon_{2k+1}(t) \leq \varepsilon_{2k-1}(t) \leq 0.
\end{aligned}$$

From these inequalities it can be immediately proved

Theorem 5.6

If the confluent form of the ε-algorithm is applied to a function f such that $\lim_{t \to \infty} f(t) = S$ and if there exist two constants $a \neq 0$ and b such that $af + b$ is totally monotonic, then

$$\lim_{t \to \infty} \varepsilon_{2k}(t) = S\,, \qquad k = 0, 1, \ldots$$

Contrarily to the results given in section 2.3 for totally monotonic sequences it has not yet been possible to prove that ε_{2k+2} converges to S faster than ε_{2k}. This result is certainly not always true since, for the

totally monotonic function $f(t) = 1/t$, we have $\varepsilon_{2k}(t) = 1/(k+1)t$ and thus

$$\frac{\varepsilon_{2k+2}(t)}{\varepsilon_{2k}(t)} = \frac{k+1}{k+2}.$$

It also remains to study the convergence of the sequence $(\varepsilon_{2k}(t))_k$ for all $t \geq T$.

The confluent form of the topological ε-algorithm was defined and studied by Brezinski [50]. In section 4.2, we assume that the S_n's and y were vectors. More generally the same formulæ, algorithms and results hold if the S_n's are elements of a vector space E on R or C, if y is an element of its dual space E^* that is a linear functional on E and if $<\cdot,\cdot>$ denotes the bilinear form of the duality that is if $<y,u>$ denotes the number obtained by applying the linear functional y to $u \in E$. In that case the $\varepsilon_{2k}^{(n)}$'s are elements of E while the $\varepsilon_{2k+1}^{(n)}$'s are elements of E^*. For defining the confluent form of the topological ε-algorithm we need E to be a space of differentiable functions of R into R^p. Thus $f(t)$ will denote a p-uple of real differentiable functions $(f_1(t),\ldots,f_p(t))$ and $y = y(t)$ will be a differentiable application of E into R that is $y(t) : f(t) = (f_1(t),\ldots,f_p(t)) \longmapsto <y(t), f(t)> \in \mathrm{R}$.

The confluent form of the topological ε-algorithm was obtained by replacing n by $x = t + nh$, $\varepsilon_{2k}^{(n)}$ by $\varepsilon_{2k}(x)$, $\varepsilon_{2k+1}^{(n)}$ by $h^{-1}\varepsilon_{2k+1}(x)$ and then letting h tend to zero. Its rules are

$$\begin{aligned}
& \varepsilon_{-1}(t) = 0\,, \quad \varepsilon_0(t) = f(t) \\
\varepsilon_{2k+1}(t) &= \varepsilon_{2k-1}(t) + \frac{y}{<y\,,\,\varepsilon'_{2k}(t)>}\,, \qquad k = 0,1,\ldots \\
\varepsilon_{2k+2}(t) &= \varepsilon_{2k}(t) + \frac{\varepsilon'_{2k}(t)}{<\varepsilon'_{2k+1}(t)\,,\,\varepsilon'_{2k}(t)>}\,, \qquad k = 0,1,\ldots
\end{aligned}$$

where y is independent of t.

Thus the $\varepsilon_{2k}(t)$'s are elements of E and the $\varepsilon_{2k+1}(t)$'s are elements of E^* which depend on t although y does not.

Algebraic results similar to those of the confluent form of the scalar ε-algorithm can be obtained. First a similar property of quasi-linearity

holds. Let us set

$$\tilde{H}_k^{(n)}(t) = \begin{vmatrix} f^{(n)}(t) & \cdots & f^{(n+k-1)}(t) \\ <y, f^{(n+1)}(t)> & \cdots & <y, f^{(n+k)}(t)> \\ \vdots & & \vdots \\ <y, f^{(n+k-1)}(t)> & \cdots & <y, f^{(n+2k-2)}(t)> \end{vmatrix}$$

and $H_k^{(n)}(t) = <y, \tilde{H}_k^{(n)}(t)>$.

By Sylvester's identity we have

$$\tilde{H}_{k+1}^{(n)}(t) \cdot H_{k-1}^{(n+2)}(t) = \tilde{H}_k^{(n)}(t) \cdot H_k^{(n+2)}(t) - \tilde{H}_k^{(n+1)}(t) \cdot H_k^{(n+1)}(t)$$

with $H_0^{(n)}(t) = \tilde{H}_0^{(n)}(t) = 1$, $\tilde{H}_1^{(n)}(t) = f^{(n)}(t)$ and $H_1^{(n)}(t) = <y, f^{(n)}(t)>$.

The following determinantal formulæ hold

Theorem 5.7

$$\varepsilon_{2k}(t) = \frac{\tilde{H}_{k+1}^{(0)}(t)}{H_k^{(2)}(t)}$$

$$\varepsilon_{2k+1}(t) = y \cdot \frac{H_k^{(3)}(t)}{H_{k+1}^{(1)}(t)}$$

$$\varepsilon_{2k+2}(t) = \varepsilon_{2k}(t) - \frac{H_{k+1}^{(1)}(t) \cdot \tilde{H}_{k+1}^{(1)}(t)}{H_k^{(2)}(t) \cdot H_{k+1}^{(2)}(t)}.$$

If we set $e_k(t; f) = \varepsilon_{2k}(t)$ then we see that $\varepsilon_{2k+1}(t) = y/<y, e_k(t; f')>$.

It can be proved, from theorem 5.7, that the result of theorem 5.4 still holds, the condition being now only sufficient and we shall not recopy it.

From the practical point of view, the $\varepsilon_{2k}(t)$'s are not computed by the rule of the confluent form since it involves differentiation, but by the recurrence relations of Hankel determinants and the relations of theorem 5.7. They can also be obtained by using the bordering method for solving the system

$$\begin{cases} a_0 = 1 \\ a_0 <y, f'(t)> + \cdots + a_k <y, f^{(k+1)}(t)> = 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ a_0 <y, f^{(k)}(t)> + \cdots + a_k <y, f^{(2k)}(t)> = 0 \end{cases}$$

and then computing

$$\varepsilon_{2k}(t) = a_0 f(t) + a_1 f'(t) + \cdots + a_k f^{(k)}(t).$$

It must be noticed that $< y, \varepsilon_{2k}(t) >$ is equal to $\varepsilon_{2k}(t)$ obtained by applying the confluent form of the scalar ε-algorithm to $< y, f(t) >$. Thus if $< y, f(t) >$ is totally monotonic then the convergence of $< y, \varepsilon_{2k}(t) >$ to $< y, S >$ holds by theorem 5.6.

5.4 Confluent ϱ-algorithm

The confluent form of the ϱ-algorithm was obtained by Wynn [472] by the same method he used for the ε-algorithm. Its rules are

$$\begin{aligned} \varrho_{-1}(t) &= 0\,, \quad \varrho_0(t) = f(t) \\ \varrho_{k+1}(t) &= \varrho_{k-1}(t) + \frac{k+1}{\varrho_k'(t)}\,, \qquad k = 0, 1, \ldots \end{aligned}$$

Let $\overline{H}_k^{(n)}(t)$ be the functional Hankel determinant obtained by replacing $f^{(i)}(t)$ in $H_k^{(n)}(t)$ by $f^{(i)}(t)/i!$. Wynn [484] proved

Theorem 5.8

$$\varrho_{2k}(t) = \frac{\overline{H}_{k+1}^{(0)}(t)}{\overline{H}_k^{(2)}(t)}\,, \qquad \varrho_{2k+1}(t) = \frac{\overline{H}_k^{(3)}(t)}{\overline{H}_{k+1}^{(1)}(t)}.$$

Let $\varrho_{2k}(t)$ and $\tilde{\varrho}_{2k}(t)$ be the functions obtained respectively by applying the confluent form of the ϱ-algorithm to $f(t)$ and

$$\tilde{f}(t) = \frac{a + b\, f(t)}{c + d\, f(t)}$$

where $ad - bc \neq 0$; then

$$\tilde{\varrho}_{2k}(t) = \frac{a + b\, \varrho_{2k}(t)}{c + d\, \varrho_{2k}(t)}.$$

The $\varrho_{2k}(t)$'s can be computed by the rule of the confluent ϱ-algorithm. However, as for the other confluent algorithms, this method requires to know the closed form of $\varrho_k(t)$ and then to differentiate it. Another

method is to use the recurrence relations for the functional Hankel determinants with the initializations $H_1^{(n)}(t) = f^{(n)}(t)/n!$. Another possibility is to use the ω-algorithm (see section 2.3) with $\omega_0^{(n)} = f^{(n)}(t)/n!$ and we shall have $\omega_{2k}^{(0)} = \varrho_{2k}(t)$ and $\omega_{2k}^{(1)} = 1/\varrho_{2k+1}(t)$. Finally the scalar ε-algorithm applied to the sequence (S_n) defined by $\Delta^n S_0 = f^{(n)}(t)/n!$ for $n = 0, 1, \ldots$ will provide $\varepsilon_{2k}^{(0)} = \varrho_{2k}(t)$.

In section 2.5, we saw that the ϱ-algorithm is connected to rational interpolation and continued fractions. Now, when using the confluent form of the ϱ-algorithm, all the interpolation points coincide and we obtain the so-called Thiele expansion formula which generalizes Taylor's

$$f(t+h) = f(t) + \cfrac{h}{\alpha_1(t)} + \cfrac{h}{\alpha_2(t)} + \cdots$$

with $\alpha_k(t) = \varrho_k(t) - \varrho_{k-2}(t)$, $k = 1, 2, \ldots$.

While Taylor's expansion terminates if f is a polynomial, Thiele's terminates when f is a rational function with the same degrees in the numerator and the denominator or with the degree of the numerator equal to that of the denominator plus one. Let us replace t by 0 and h by x, then Thiele formula becomes

$$f(x) = f(0) + \cfrac{x}{\alpha_1} + \cfrac{x}{\alpha_2} + \cdots$$

with $\alpha_k = \alpha_k(0)$. Let $C_k(x) = A_k(x)/B_k(x)$ be the successive convergents of this continued fraction. Then A_{2k-1}, A_{2k}, B_{2k} and B_{2k+1} are polynomials of degree k in x. If f is expanded into a formal power series

$$f(x) = \sum_{i=0}^{\infty} c_i x^i$$

and $C_k(x)$ also, then it can be proved that

$$f(x) - C_k(x) = O\left(x^{k+1}\right) \qquad (x \to 0).$$

Looking at the degrees of A_k and B_k and at the definition of Padé approximants (see section 2.3) this means that we have

$$C_{2k}(x) = [k/k]_f(x), \qquad C_{2k-1}(x) = [k/k-1]_f(x).$$

Thus there exists a relation between the confluent form of the ϱ-algorithm and the qd-algorithm (see section 2.4) which is related to Padé approximants and continued fractions, see Brezinski [63]. We have

$$\begin{aligned} -\alpha_{2k}\alpha_{2k+1}e_k^{(1)} &= 1 \\ -\alpha_{2k}\alpha_{2k-1}q_k^{(1)} &= 1 \end{aligned}$$

and the qd-algorithm can be used for computing the α_k's and reciprocally.

By the link between Padé approximants and orthogonal polynomials there is also a connection between the confluent ϱ-algorithm and formal orthogonal polynomials as explained by Brezinski [60]. Let c be the linear functional on the space of polynomials defined by $c\,(x^i) = c_i$ for $i = 0, 1, \ldots$. $\{P_k\}$ is said to be a family of formal orthogonal polynomials with respect to c if, for all k, P_k has the exact degree k and

$$c\left(x^i P_k(x)\right) = 0\,, \qquad \text{for } i = 0, \ldots, k-1.$$

Such polynomials satisfy a three-term recurrence relation and can be computed via the confluent form of the ϱ-algorithm as follows. We set

$$P_{-1}(x) = 0\,, \quad P_0(x) = 1\,, \quad \beta_{-1} = \beta_0 = 1\,, \quad \varrho_{-1}(x) = 0\,, \; \varrho_0(x) = f(x).$$

Then for $k = 0, 1, \ldots$ we do

- computation of $\varrho_{2k+1}(0)$ and $\varrho_{2k+2}(0)$.
- computation of

$$\begin{aligned} \beta_{2k+1} &= \varrho_{2k+1}(0) - \varrho_{2k-1}(0) \\ \beta_{2k+2} &= \varrho_{2k+2}(0) - \varrho_{2k}(0) \\ B_{k+1} &= \frac{\beta_{2k} + \beta_{2k+2}}{\beta_{2k}\beta_{2k+1}\beta_{2k+2}} \\ C_{k+1} &= \frac{1}{\beta_{2k-1}\beta_{2k}^2\beta_{2k+1}} \\ P_{k+1}(x) &= (x + B_{k+1})P_k(x) - C_{k+1}P_{k-1}(x). \end{aligned}$$

5.5 Confluent G-transform

The confluent G-transform, called by its authors Gray, Atchison and McWilliams [197] a higher order G-transformation, is a mixing of confluent and scalar transformations. It is defined by

$$G_k(t) = \frac{\begin{vmatrix} f(t) & f(t+h) & \cdots & f(t+kh) \\ f'(t) & f'(t+h) & \cdots & f'(t+kh) \\ f'(t+h) & f'(t+2h) & \cdots & f'(t+(k+1)h) \\ \vdots & \vdots & & \vdots \\ f'(t+(k-1)h) & f'(t+kh) & \cdots & f'(t+(2k-1)h) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ f'(t) & f'(t+h) & \cdots & f'(t+kh) \\ f'(t+h) & f'(t+2h) & \cdots & f'(t+(k+1)h) \\ \vdots & \vdots & & \vdots \\ f'(t+(k-1)h) & f'(t+kh) & \cdots & f'(t+(2k-1)h) \end{vmatrix}}.$$

$G_k(t)$ can be recursively computed by the G-transform given in section 2.4. Let (S_n) and (x_n) be the sequences respectively defined by

$$S_n = f(t+nh), \qquad x_n = f'(t+nh), \qquad n = 0, 1, \ldots$$

then applying the G-algorithm gives

$$G_k^{(n)} = G_k(t+nh), \qquad k, n = 0, 1, \ldots$$

By construction we have the

Theorem 5.9

If $\forall t \geq T$, $f(t) = S + a_1 f'(t) + \cdots + a_k f^{(k)}(t)$ *then* $\forall t \geq T, G_k(t) = S$.

In particular if $f(t) = \int_0^t e^{-sx} g(x)dx$ where g satisfies a homogeneous linear differential equation of order k with constant coefficients then the condition of this theorem is satisfied and we obtain

$$G_k(t) = \int_0^\infty e^{-sx} g(x)dx.$$

If we let h tend to zero in the confluent G-transform then we have

$$\lim_{h \to 0} G_k(t) = \varepsilon_{2k}(t).$$

5.6 Confluent E-algorithm

Let us assume that, for all t, f satisfies

$$f(t) = S + a_1 g_1(t) + \cdots + a_k g_k(t).$$

The problem, which is similar to the problem solved by the E-transformation for sequences, is to calculate S. Differentiating the preceding equation leads to the system

$$\left\{ \begin{array}{lcl} f(t) & = & S + a_1 g_1(t) + \cdots + a_k g_k(t) \\ f'(t) & = & a_1 g_1'(t) + \cdots + a_k g_k'(t) \\ \vdots & & \vdots \\ f^{(k)}(t) & = & a_1 g_1^{(k)}(t) + \cdots + a_k g_k^{(k)}(t). \end{array} \right.$$

Thus S is given as a ratio of two determinants which will be denoted by $E_k(t)$

$$E_k(t) = \frac{\begin{vmatrix} f(t) & f'(t) & \cdots & f^{(k)}(t) \\ g_1(t) & g_1'(t) & \cdots & g_1^{(k)}(t) \\ \vdots & \vdots & & \vdots \\ g_k(t) & g_k'(t) & \cdots & g_k^{(k)}(t) \end{vmatrix}}{\begin{vmatrix} g_1'(t) & \cdots & g_1^{(k)}(t) \\ \vdots & & \vdots \\ g_k'(t) & \cdots & g_k^{(k)}(t) \end{vmatrix}}.$$

By construction we have the following result

Theorem 5.10

If $\forall t \geq T$, $f(t) = S + a_1 g_1(t) + \cdots + a_k g_k(t)$ *then* $\forall t \geq T, E_k(t) = S$.

This functional transformation contains several cases studied in the previous sections. For $g_i(t) = t^i$ we recover Taylor's expansion. The choice $g_i(t) = f^{(i)}(t)$ leads to the confluent form of the ε-algorithm.

Let v be an arbitrary function. Let us define the element z_i of the dual space by

$$< z_i, v >= v^{(i)}(t).$$

Then $E_k(t)$ is similar to the ratio defining E_k in the RPA and thus it can be computed by this algorithm which is, in this case

$$\begin{aligned} E_0(t) &= f(t)\,, \quad g_{0,i}(t) = g_i(t), && i \geq 1 \\ E_k(t) &= E_{k-1}(t) - \frac{E_{k-1}^{(k)}(t)}{g_{k-1,k}^{(k)}(t)} \cdot g_{k-1,k}(t)\,, && k \geq 1 \\ g_{k,i}(t) &= g_{k-1,i}(t) - \frac{g_{k-1,i}^{(k)}(t)}{g_{k-1,k}^{(k)}(t)} \cdot g_{k-1,k}(t)\,, && i > k \geq 1 \end{aligned}$$

where the upper index designates the k-th derivative with respect to t. The $E_k(t)$'s can also be computed by the scalar E-algorithm applied to the sequence (S_n) defined by $\Delta^n S_0 = f^{(n)}(t)$, $n = 0, 1, \ldots$ and with the auxiliary sequences $\left(g_{0,i}^{(n)}\right)$ given by $\Delta^n g_{0,i}^{(0)} = g_i^{(n)}(t)$, $n = 0, 1, \ldots$, and we shall obtain $E_k^{(0)} = E_k(t)$.

5.7 Θ-type confluent algorithms

Let us first begin with the confluent form of the ε-algorithm and follow the same method as in the scalar case for obtaining the Θ-algorithm. The confluent ε-algorithm has the form

$$\begin{aligned} \varepsilon_{-1}(t) &= 0\,, \quad \varepsilon_0(t) = f(t) \\ \varepsilon_{k+1}(t) &= \varepsilon_{k-1}(t) + D_k(t) \quad \text{with } D_k(t) = \frac{1}{\varepsilon_k'(t)}. \end{aligned}$$

Thus ε_{2k+2}' will converge faster than ε_{2k}' (that is $\lim_{t\to\infty} \varepsilon_{2k+2}'(t)/\varepsilon_{2k}'(t) = 0$) if and only if

$$\lim_{t\to\infty} D_{2k+1}'(t)/\varepsilon_{2k}'(t) = -1.$$

If this condition is not satisfied we can introduce a parameter ω_k in the rule of the algorithm which becomes

$$\varepsilon_{2k+2}(t) = \varepsilon_{2k}(t) + \omega_k D_{2k+1}(t)\,,$$

the rule for the functions with an odd index being unchanged. If ω_k is chosen as

$$\omega_k = -\lim_{t\to\infty} \varepsilon_{2k}'(t)/D_{2k+1}'(t)$$

then the new ε_{2k+2}' will converge faster than ε_{2k}'.

For this new algorithm we have (keeping the same Greek letter ε)

Theorem 5.11

A necessary and sufficient condition that $\forall t \geq T, \varepsilon_2(t) = S$ *is that* $\forall t \geq T$

$$\begin{aligned} f(t) &= S + ce^{-at} \\ \text{or} \quad f(t) &= [(1-\omega_0)at + c]^{1/(1-\omega_0)}. \end{aligned}$$

The first case corresponds to $\omega_0 = 1$ and the second one to $\omega_0 \neq 1$. Since the computation of ω_k is difficult because it involves a limit, let us replace it by an arbitrary function $\omega_k(t)$ In that case we have the

Theorem 5.12

A necessary and sufficient condition that $\forall t \geq T, \varepsilon_2(t) = S$ *is that* $\forall t \geq T$

$$f(t) = S + c \cdot \exp \int \frac{dt}{a + t - \Omega_0(t)}$$

where $\Omega_0'(t) = \omega_0(t)$.

Of course, due to the previous relation for ω_k, we are naturally led to the choice

$$\omega_k = -\varepsilon_{2k}'(t)/D_{2k+1}'(t)$$

and we obtain the confluent form of the Θ-algorithm

$$\Theta_{-1}(t) = 0\,, \quad \Theta_0(t) = f(t)$$

$$\Theta_{2k+1}(t) = \Theta_{2k-1}(t) + \frac{1}{\Theta_{2k}'(t)}\,, \qquad k = 0, 1, \ldots$$

$$\Theta_{2k+2}(t) = \Theta_{2k}(t) + \frac{\Theta_{2k}'(t) \cdot \Theta_{2k+1}'(t)}{\Theta_{2k+1}''(t)}\,, \qquad k = 0, 1, \ldots$$

Of course the usefulness of this algorithm is limited by the fact that its implementation needs the closed form of the $\Theta_k(t)$ and their differentiation. This is the reason why we shall not give here more results on it and refer the interested reader to Brezinski [55, 56].

The same procedure (the procedure Θ) can be applied to any continuous prediction algorithm whose rule can be written as

$$T(t) = a(t) + b(t).$$

The Θ-type algorithm associated with this algorithm consists in the continuous prediction algorithm (which has not yet been studied)

$$\Theta(t) = a(t) - \frac{a'(t)}{b'(t)} \cdot b(t).$$

Chapter 6

APPLICATIONS

Of course the fundamental domain of application of the algorithms studied in the previous chapters is convergence acceleration. In numerical analysis and in applied mathematics and sciences, sequences are very often obtained. They come from series, continued fractions and infinite products which are used, for instance, for computing special functions, see, for example, Nikiforov and Uvarov [336] or Spanier and Oldham [417]. They also come from perturbation series obtained from perturbation theory and from asymptotic series derived from differential equations, difference equations, integration by parts, Laplace's method, steepest-descent methods for finding the asymptotic behaviour of some integrals, etc., see, for example, Bender and Orszag [25]. In numerical analysis many methods are iterative or depend on a parameter h. This is the case in methods for solving systems of linear and nonlinear equations, for computing eigenelements of a matrix and their derivatives, in quadrature methods, in methods for solving ordinary and partial differential equations, in methods for integral equations, in approximation, in numerical differentiation, etc.. Thus the field of application of extrapolation and convergence acceleration methods is quite wide: numerical analysis and applied mathematics.

But these methods also present another interesting characteristic. They can be used not only for the purpose of convergence acceleration but they are also able to furnish new methods for the solution of some problems. A well known example of this situation is Steffensen's method for solving a nonlinear equation which is obtained by cycling with Aitken's Δ^2 process. Some other examples will be discussed below.

Another interesting feature is their connection with various rational interpolation and approximation procedures thus leading to their use

in the solution of such problems which are now very much in demand. Finally let us mention their intimate link with biorthogonality which is connected with very many other topics, see Brezinski [89], such as projection methods, the method of moments and that of Lanczos, the conjugate and biconjugate gradient methods, some statistical methods such as the jackknife, ARMA models, and the multiple correlation coefficient, least squares, Fredholm equations, orthogonal and biorthogonal polynomials and some generalizations as well, Fourier expansion, Galerkin's method, Laplace transform inversion, splines, Borel summation process, the τ method, etc.. But, may be, their most important connection is with Padé approximants and continued fractions which received much attention since two decades because of their wide applications in numerical analysis, approximation theory, number theory, linear filtering and digital processing, theoretical physics, quantum mechanics, quantum field theory, and in many applied sciences such as circuit theory, fluid mechanics, dynamical systems and fractals, nuclear and molecular physics, biology, electronics, chemistry, etc.. For a bibliography containing about 6000 items, see Brezinski [88].

6.1 Sequences and series

Although it is not possible to base any conclusion on the effectiveness of a particular acceleration method on purely numerical results, we tried, in this section, to give a flavour of the possibilities of the various methods studied in the previous chapters. Of course we do not claim to be exhaustive but we do hope however that the numerical results will help the reader in the understanding of the theory and in the choice of an algorithm or a device adapted to his particular problem.

6.1.1 Simple sequences

The sequences we used can be classified into three categories according to their law of construction

1. $S_{n+1} = F(S_n)$
2. $S_n = f(x_n)$
3. $S_n = a_n b_n$

where $(x_n), (a_n)$ and (b_n) are given auxiliary sequences. The first case will be treated in section 6.2.4 since it is related to fixed point methods.

Into the second category we find sequences like $S_n = e^{x_n}$ where (x_n) tends to zero. According to (x_n) the ratios $(S_{n+1} - 1)/(S_n - 1)$ and $\Delta S_{n+1}/\Delta S_n$ can present very different and interesting behaviours. The same is true for sequences as $S_n = x_n^{-1} \sin x_n$.

This last example also fails in the third class of sequences which is quite general. For example if $b_n = 1$ for all n then $S_n = a_0 \cdot \ldots \cdot a_n$. Assuming that (S_n) tends to zero the ratio (S_{n+1}/S_n) can present strange characteristics. For example if $a_{3n} = -2^n, a_{3n+1} = 2^{n+1}$ and $a_{3n+2} = 8^{-n}$ it has three accumulation points which are $-\infty, +\infty$ and 0. The same is true with $a_{3n} = 0.9 + 0.05/(n+1), a_{3n+1} = 0.7 + (-0.5)^{n+1}$ and $a_{3n+2} = (a_{3n-1} + 0.9025/a_{3n-1})/2$. But, in this case, the convergence of (S_{n+1}/S_n) to its three accumulation points 0.9, 0.7 and 0.95 is quite different since it is logarithmic, linear and quadratic respectively. Finally let us also mention that (a_n) can be randomly chosen in $[a, b]$ with $-1 < a < b < 1$.

Let us consider the sequence $(S_n = e^{x_n})$ where (x_n) tends to zero. Thus (S_n) converges to 1 and we have

$$\frac{S_{n+1} - 1}{S_n - 1} = \frac{x_{n+1}}{x_n} \cdot (1 + \varepsilon_n), \qquad n = 1, 2, \ldots$$

$$\frac{\Delta S_{n+1}}{\Delta S_n} = \frac{x_{n+1}}{x_n} \cdot \frac{\Delta x_{n+1}}{\Delta x_n} \cdot (1 + \varepsilon'_n), \qquad n = 1, 2, \ldots$$

where (ε_n) and (ε'_n) both tend to zero.

Let us first take $x_n = (-1)^{\lfloor n/2 \rfloor} / n$, $n = 1, 2, \ldots$ where $\lfloor a \rfloor$ denotes the integer part of the positive real number a.

In that case

$$\lim_{n \to \infty} \frac{S_{2n+1} - 1}{S_{2n} - 1} = 1 \quad \lim_{n \to \infty} \frac{S_{2n+2} - 1}{S_{2n+1} - 1} = -1$$

$$\lim_{n \to \infty} \frac{\Delta S_{2n+1}}{\Delta S_{2n}} = +\infty \quad \lim_{n \to \infty} \frac{\Delta S_{2n+2}}{\Delta S_{2n+1}} = 0.$$

Using the first 20 terms of the sequence (S_n) we obtain the following best numbers of exact digits $(-\log_{10} |\text{absolute precision}|)$

ε-algorithm	:	3.18
ϱ-algorithm with $x_n = n$	:	1.42
Θ-algorithm	:	1.95
Richardson process with $x_n = \Delta S_n$	:	2.35
First generalization of the ε-algorithm with $x_n = 1 + n^{-1}$	:	2.55
Second generalization of the ε-algorithm with $x_n = 1 + n^{-1}$	:	2.74
Iterated Δ^2 process	:	1.05
Overholt process	:	1.29
Levin's t-transform	:	0.62

We have

$$S_n - 1 = \frac{x_n}{1!} + \frac{x_n^2}{2!} + \frac{x_n^3}{3!} + \cdots$$

Thus we can try to apply the E-algorithm with $g_i(n) = x_n^i$ and we obtain the following results (number of exact digits)

$n\backslash k$	0	1	2	3	4	5	6	7	8	9	10
1	0.00										
2	0.41	0.51									
3	0.55	1.20	1.52								
4	0.55	1.39	2.22	2.71							
5	0.65	1.54	2.54	3.49	4.11						
6	0.81	1.77	2.83	3.94	4.97	5.68					
7	0.88	1.97	3.11	4.29	5.49	6.60	7.38				
8	0.88	2.05	3.32	4.60	5.89	7.16	8.34	9.19			
9	0.93	2.12	3.47	4.87	6.25	7.61	8.96	10.19	11.09		
10	1.02	2.25	3.62	5.08	6.57	8.03	9.47	10.87	12.14	13.10	
11	1.06	2.37	3.78	5.27	6.83	8.40	9.93	11.42	12.86	14.09	15.1

Let us now take

$$x_n = \left(0.9 + \frac{\sin 2n}{n}\right) \cdot x_{n-1}, \qquad n = 1, 2, \ldots$$

with $x_0 = 1$. Obviously in that case $\lim_{n\to\infty} x_{n+1}/x_n = 0.9$ and the conditions of theorems 2.8 and 2.10 about the convergence and the acceleration for the E-algorithm are satisfied.

From the first 20 terms the best results are (number of exact digits)

ε-algorithm	: 2.76
ϱ-algorithm with $x_n = n$	: 0.54
Θ-algorithm	: 0.36
Richardson process with $x_n = \Delta S_n$	: 0.55
First generalization of the ε-algorithm with $x_n = 1 + n^{-1}$	: 2.00
Second generalization of the ε-algorithm with $x_n = 1 + n^{-1}$	: 1.16
Iterated Δ^2 process	: 0.29
Overholt process	: 0.70
Levin's t-transform	: 0.82

The E-algorithm gives

$n\backslash k$	0	1	2	3	4	5	6	7	8	9	10
1	0.00										
2	0.00	0.00									
3	0.00	0.19	0.26								
4	0.00	0.23	0.69	0.93							
5	0.00	0.29	0.86	1.45	1.81						
6	0.12	0.53	1.01	1.72	2.41	2.86					
7	0.10	0.62	1.22	1.85	2.65	3.43	3.95				
8	0.18	0.67	1.38	2.12	2.85	3.73	4.58	5.16			
9	0.29	0.85	1.53	2.37	3.21	4.02	4.97	5.88	6.52		
10	0.30	0.95	1.70	2.52	3.46	4.38	5.26	6.27	7.23	7.92	
11	0.35	1.01	1.86	2.74	3.65	4.67	5.66	6.60	7.66	8.69	9.30

Let us now consider the sequence

$$S_n = \left(1 + \frac{x}{n}\right)^n, \qquad n = 1, 2, \ldots$$

which converges to $S = e^x$.

With the 9 preceding transformations we obtain respectively for the last computed result the following numbers of exact digits for $x = 1$

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Over.	Lev. t
3	0.89	1.06	2.83	0.89	1.06	0.33	0.57	1.06	1.06	1.22
4	0.99	1.19	3.33	2.60	1.26	0.38	0.56	1.19	1.28	1.31
5	1.07	1.37	5.55	3.09	1.40	0.28	0.56	1.49	1.44	1.51
7	1.20	1.59	8.51	5.81	1.61	0.26	0.56	1.87	1.67	1.76
9	1.30	1.77	11.56	6.71	1.77	0.24	0.56	2.21	1.84	1.96
11	1.38	1.92	11.90	8.29	1.90	0.24	0.56	2.51	1.97	2.13
13	1.44	2.04	13.46	7.75	2.00	0.23	0.56	2.96	2.08	2.26

For other values of x the behaviour of the various transformations is quite similar.

Let us now consider the sequence

$$\left[\frac{\ln(1+x_n)}{\ln x_n}\right]^{x_n \ln x_n}, \qquad n = 1, 2, \ldots$$

where (x_n) is an auxiliary sequence tending to infinity. (S_n) converges to $S = e$, see Monchamp and Chamaraux [323].

With $x_n = n + 1$ we obtain

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Over.	Lev. t
3	0.80	0.98	2.96	0.80	0.98	0.33	0.53	0.98	0.98	1.21
7	1.10	1.51	3.32	3.42	1.53	0.27	0.39	1.77	1.59	1.66
11	1.27	1.83	3.86	3.58	1.81	0.24	0.51	2.54	1.88	2.04
15	1.40	2.07	4.55	3.58	2.01	0.24	0.52	2.29	2.09	2.30
20	1.52	2.30	4.67	3.58	2.19	0.25	0.49	3.14	2.28	2.55

With $x_n = 2^n$ we have

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Over.	Lev. t
3	1.10	1.93	0.74	1.10	1.93	0.27	0.55	1.93	1.93	1.15
7	2.33	4.68	1.82	3.67	4.48	0.30	0.50	4.52	4.47	4.76
9	2.95	6.46	1.81	5.81	5.31	0.30	0.64	4.77	5.31	7.08
11	3.56	7.87	2.65	5.18	6.12	0.29	0.23	4.73	6.12	7.67
15	4.78	10.96	2.84	6.18	7.68	0.25	0.20	8.90	7.68	10.90

Let us now consider the sequence

$$S_n = (\cos x_n + a \sin x_n)^{1/x_n}, \qquad n = 1, 2, \ldots$$

where (x_n) is an auxiliary sequence converging to zero. This sequence tends to $S = e^a$ as proved by Monchamp and Chamaraux [323].

With $x_n = 1/n$ we obtain for $a = 1$

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Over.	Lev. t
3	0.62	0.77	1.72	0.62	0.77	0.05	0.29	0.77	0.77	0.81
5	0.79	1.06	4.06	1.97	1.12	−0.08	0.43	1.16	1.16	1.20
7	0.91	1.29	6.85	3.84	1.33	−0.26	−0.15	1.52	1.38	1.47
9	1.01	1.46	9.33	4.84	1.48	−1.06	0.04	1.88	1.55	1.67
11	1.08	1.61	10.82	7.75	1.61	0.11	0.07	2.48	1.68	1.83
18	1.28	1.99	13.47	9.86	1.92	−0.05	0.07	3.34	2.00	2.24

With $x_n = 0.8^n$ we found

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Over.	Lev. t
3	0.48	0.53	0.02	0.48	0.53	0.33	0.39	0.53	0.53	0.21
5	0.62	0.16	−0.50	1.00	0.59	0.32	0.39	−0.51	0.66	0.83
7	0.77	2.19	0.91	1.00	0.05	0.32	0.39	1.78	0.43	2.31
9	0.93	2.13	1.18	1.55	0.40	0.32	0.39	3.48	1.27	2.63
11	1.11	4.21	1.13	3.19	1.41	0.32	0.41	4.46	3.29	4.31
20	1.94	10.56	2.36	4.43	9.36	0.32	0.37	7.50	11.54	7.40

As a last example let us take

$$S_n = a_1 \cdot \ldots \cdot a_n, \qquad n = 1, 2, \ldots$$

with $a_{3n} = 0.9 + 0.05/(3n+1)$, $a_{3n+1} = 0.7 + (-0.5)^{3n+2}$ and $a_{3n+2} = (a_{3n-1} + 0.9025/a_{3n-1})/2$ with $a_{-1} = 0.9$. The sequence (S_n) converges to zero but the ratios (S_{n+1}/S_n) have three accumulation points which are 0.9, 0.7 and 0.95 (the square root of 0.9025). Thus the convergence of (S_{n+1}/S_n) to these values is quite different since it is logarithmic for the first one, linear for the second one and quadratic for the last one. Thus such a sequence will certainly be difficult to accelerate and with the nine algorithms we obtain (number of exact digits)

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Ov.	Lev.t
3	0.08	0.00	−0.05	0.08	0.00	0.02	0.03	0.00	0.00	0.00
4	0.26	0.03	−0.02	−1.47	−0.01	0.04	0.06	0.03	0.00	0.21
5	0.28	0.00	−0.06	0.21	0.10	0.02	0.05	0.00	0.29	0.23
6	0.32	0.34	1.61	0.27	−1.55	−0.62	0.36	0.24	0.29	0.25
7	0.48	1.12	−0.15	0.30	−1.73	0.02	−0.03	0.25	0.29	0.25
8	0.50	1.13	0.32	0.19	−1.46	−0.07	0.38	0.24	0.51	0.65
9	0.54	1.12	0.93	0.26	−2.06	0.03	0.54	0.24	0.51	0.48
10	0.70	1.13	0.27	0.21	−2.33	−0.07	0.22	0.24	0.51	0.47
11	0.72	1.12	0.55	0.45	−1.90	−0.02	0.19	0.24	0.73	0.33
12	0.76	2.06	0.16	0.24	−2.27	−0.07	0.22	0.46	0.73	0.85
13	0.92	2.87	0.50	0.35	−2.66	−0.03	0.20	0.46	0.73	0.71
14	0.94	2.91	−1.38	−0.36	−2.13	0.00	−1.28	0.46	0.95	0.66
15	0.98	2.86	0.31	0.39	−2.34	−0.03	−0.32	0.46	0.95	0.00
16	1.14	3.50	−0.32	−1.20	−2.25	−0.07	−0.11	0.46	0.95	1.06
17	1.16	3.36	0.52	1.55	−2.01	−0.03	−0.29	0.46	1.17	0.93
18	1.21	3.46	−0.27	1.12	−2.26	−0.06	0.17	0.68	1.17	0.82
19	1.36	4.40	0.81	5.52	−2.81	−0.03	0.15	0.68	1.17	1.11
20	1.38	4.39	0.60	1.89	−1.34	−0.07	−0.36	0.68	1.39	1.27
21	1.43	4.40	0.84	0.64	−2.03	−0.02	0.09	0.68	1.39	1.14
22	1.58	4.89	0.90	1.18	−3.36	−0.08	−0.66	0.68	1.39	0.90
23	1.60	4.86	0.85	0.82	−2.42	0.18	−0.59	0.68	1.61	2.45
24	1.65	4.88	0.91	0.79	−0.55	−0.08	−0.59	0.90	1.61	1.48
25	1.80	7.32	0.79	0.80	−4.24	0.03	−0.59	0.90	1.61	1.32
26	1.83	5.71	1.08	0.81	−3.35	−0.08	−0.59	0.90	1.83	0.69

Then a division by zero occurs in Overholt's process and we obtain for n=50

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Over.	Lev. t
50	3.60	10.51	2.23	1.64	−0.63	−0.17	2.02	1.79	1.83	3.47

Thus, on this example, almost all the algorithms, except the ε-algorithm, behave almost chaotically. Richardson's process is always very bad and the generalizations of the ε-algorithm also which is strange since the choice $(x_n = 1 + 1/n)$ for the auxiliary sequence provides an algorithm which can be considered as intermediate between the ε-algorithm and the ϱ-algorithm. For $n = 19$, the Θ-algorithm gives quite a good result and then loses its precision.

Of course, no final conclusion can never be extracted from purely numerical results, but the preceding examples show that if nothing is known about the asymptotic behaviour of (S_n) or of related ratios such as $(S_{n+1} - S)/(S_n - S)$ or $\Delta S_{n+1}/\Delta S_n$ then it is difficult to make a good choice among all the existing sequence transformations. In particular it seems difficult to find the adequate auxiliary sequence (x_n) for the two generalizations of the ε-algorithm. However, as seen in section 2.6, when a suitable sequence (x_n) can be found then these generalizations can work quite well. Such questions were studied by Petit [356] but the results obtained are too complicated to be of a great practical interest. In such a case the best strategy seems to use an automatic selection procedure (see section 3.6) and/or a composite transformation (see section 3.7). Contractive sequence transformations can also be useful when acceleration cannot be achieved (see section 3.9).

Of course, when possible, one has to built a well adapted procedure as explained in sections 3.1 to 3.4.

The previous examples were constructed specially for illustrating our algorithms and the corresponding theoretical results. But, of course, extrapolation methods have many actual applications.

In many branches of applied sciences the solution of problems arises as a series which is usually slowly converging (when it does). Thus it is not surprising to find applications of extrapolation methods in the corresponding literature. To name a few we refer the interested reader to van Dyke and Guttmann [441] who solved by Shanks' transformation an old controversy about the critical Mach number, to Weniger, Grotendorst and Steinborn [459], Weniger and Steinborn [460], Grotendorst, Weniger and Steinborn [204], Grotendorst and Steinborn [203], Weniger [457], Belkić [19] and Killingbeck [261]. An extensive study on the question was conducted by Guttmann [207] who compared various extrapolation processes on several series. He concludes that *the Θ-algorithm was clearly the most generally successful. It was the best method for four of the first eight test functions, second best for three and third best for one*, which strengthens the conclusions of Smith and Ford [414]. Let us mention that MAPLE programs for several transformations are available, see Grotendorst [201] and Grotendorst [202].

Let us also mention that several numerical examples with divergent series were conducted by various authors. However the theoretical study of the behaviour of nonlinear transformations for such series is not much

developed except for the ε-algorithm when applied to Stieltjes series that is

$$f(z) = c_0 + c_1 z + c_2 z^2 + \cdots$$

with $c_i = \int_0^\infty x^i d\alpha(x)$ with α bounded and nondecreasing in $[0, \infty)$. On this question see Brezinski and van Iseghem [103].

As a numerical example let us take the sequence of the partial sums of the series

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

for $x = 2$. We obtain the following results ($-\log_{10}$ |relative error|)

n	S_n	ε	ϱ	Θ	Rich.	$1^{st}\varepsilon$	$2^{nd}\varepsilon$	Δ^2	Ov.	Lev.t
5	−0.56	2.59	−0.21	2.15	1.17	0.49	0.54	2.72	1.19	4.20
10	−1.78	5.28	−0.84	6.74	1.08	0.99	1.02	6.10	1.13	9.24
15	−3.11	8.36	−1.39	10.35	1.08	0.88	1.69	12.64	1.13	13.96
20	−4.50	11.12	−2.14	13.83	1.08	1.49	2.11	13.23	1.13	13.57

6.1.2 Double sequences

Several attempts have been made for generalizing the scalar algorithms of chapter 2 to double (or multiple) sequences. They can be classified into two categories: the methods transforming double sequences into one-dimensional ones and the others. Into the first class are the methods of Levin [285] and Albertsen, Jacobsen and Sorensen [7]. These procedures were synthesized and extended by Haccart [208] who showed how to use the E-algorithm for their implementation. They were proved to be particular cases of a whole class of multidimensional convergence accelerators based on multivariate Padé approximants by Cuyt [128] who proposed new methods by using multivariate rational Hermite interpolants. The second class of methods generalizes the T_{+m}-transformation and thus is a generalization of Aitken's Δ^2 process.

Let us begin by the first class of methods and consider a multiple sequence $(S_{n_1,n_2,\ldots,n_p})$. We shall set for simplicity $\underline{n} = (n_1, \ldots, n_p) \in \mathbb{N}^p$ and write $S_{\underline{n}} = S_{n_1,\ldots,n_p}$. Let w be an application from $\mathbb{N}$ to $\mathbb{N}^p$. We shall assume that w is continuous at infinity which means that every neighborhood of infinity in $\mathbb{N}^p$ is the image of a neighborhood of infinity in $\mathbb{N}$ by w and that w is a one-to-one application. With these assumptions if

$$\lim_{n_1,\dots,n_p\to\infty} S_{\underline{n}} = S$$

then

$$\lim_{i\to\infty} S_{w(i)} = S.$$

Thus the treatment of multiple sequences is brought back to the one-dimensional case and the E-algorithm can be used with

$$E_0^{(i)} = S_{w(i)}, \qquad i = 0, 1, \dots$$

$$g_{0,j}^{(i)} = g_j(w(i)), \qquad i = 0, 1, \dots \text{ and } j = 1, 2, \dots$$

and we obtain

$$E_k^{(i)} = \frac{\begin{vmatrix} S_{w(i)} & \cdots & S_{w(i+k)} \\ g_1(w(i)) & \cdots & g_1(w(i+k)) \\ \vdots & & \vdots \\ g_k(w(i)) & \cdots & g_k(w(i+k)) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ g_1(w(i)) & \cdots & g_1(w(i+k)) \\ \vdots & & \vdots \\ g_k(w(i)) & \cdots & g_k(w(i+k)) \end{vmatrix}}.$$

Thus, according to the choice of the g's, we obtain several transformations. The first two were proposed by Haccart [208]. If we take

$$g_j(w(i)) = S_{w(i+j)} - S_{w(i+j-1)}$$

then we obtain a generalization of Shanks' transformation such that $E_k^{(i)} = S, \forall i$ if the sequence $(S_{\underline{n}})$ satisfies, $\forall i$

$$a_0 \left(S_{w(i)} - S\right) + \cdots + a_k \left(S_{w(i+k)} - S\right) = 0$$

with $a_0 + \cdots + a_k \neq 0$.

A particular case of this transformation was studied by Cuyt [125]. We set

$$\sigma_q = \sum_{n_1+\cdots+n_p=q} S_{\underline{n}}$$

and we apply the scalar ε-algorithm to

$$\varepsilon_0^{(i)} = \sum_{j=0}^{p-1} (-1)^j \binom{p-1}{j} \sigma_{i-j}.$$

A generalization of this idea will be discussed below.

Another generalization of Shanks' transformation can be obtained with

$$w(i) = (n_1 + ik_1, \ldots, n_p + ik_p).$$

Let now α be another one-to-one application from $\mathbb{N}$ to $\mathbb{N}^p$. If we take

$$g_j(w(i)) = a_{w(i)+\alpha(j)}$$

with $S_{\underline{n}} = \sum_{\underline{n}=0}^{\infty} a_{\underline{n}}$, then we recover the generalization of Shanks' transformation due to Levin [285]. We shall have $E_k^{(i)} = S, \forall i$ if the sequence $(S_{\underline{n}})$ satisfies, $\forall i$

$$S_{w(i)} = S + \sum_{j=1}^{k} c_j \, a_{w(i)+\alpha(j)}.$$

Finally for the choice

$$g_j(w(i)) = S_{\alpha(j+1)+w(i)} - S_{\alpha(j)+w(i)}$$

we recover the transformation introduced by Albertsen, Jacobsen and Sorensen [7] for which no recursive algorithm for its implementation was given. We have $E_k^{(i)} = S, \forall i$ if the sequence $(S_{\underline{n}})$ satisfies, $\forall i$

$$b_0 \left(S_{\alpha(i)+w(0)} - S\right) + \cdots + b_k \left(S_{\alpha(i)+w(k)} - S\right) = 0$$

with $b_0 + \cdots + b_k \neq 0$.

Generalizations of the one-dimensional Levin's transforms (see section 2.7) are also given by these authors. They fit into our general framework and can be implemented via the E-algorithm.

As shown by Cuyt [128] all these methods are particular cases of multivariate general order Padé approximants as defined, for example, in Cuyt [126]. Using this technique, it is possible to derive a whole bunch of multiple sequence transformations which generalize the transformation of Cuyt explained above.

For example, in the case of double sequences, we can set

$$E_k^{(n)} = \frac{\begin{vmatrix} \sum_{h=0}^{n} \nabla_0 S_{i_h,j_h} & \cdots & \sum_{h=0}^{n} \nabla_k S_{i_h,j_h} \\ \nabla_0 S_{i_{n+1},j_{n+1}} & \cdots & \nabla_k S_{i_{n+1},j_{n+1}} \\ \vdots & & \vdots \\ \nabla_0 S_{i_{n+k},j_{n+k}} & \cdots & \nabla_k S_{i_{n+k},j_{n+k}} \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \nabla_0 S_{i_{n+1},j_{n+1}} & \cdots & \nabla_k S_{i_{n+1},j_{n+1}} \\ \vdots & & \vdots \\ \nabla_0 S_{i_{n+k},j_{n+k}} & \cdots & \nabla_k S_{i_{n+k},j_{n+k}} \end{vmatrix}}$$

with $\nabla_m S_{i_h,j_h} = S_{i_h-d_m,j_h-e_m} - S_{i_h-d_m-1,j_h-e_m} - S_{i_h-d_m,j_h-e_m-1} + S_{i_h-d_m-1,j_h-e_m-1}$.

This transformation can be implemented by the E-algorithm again.

Let us now come to the second class of methods. They were first proposed by Streit [419] and then extended by Haccart [208]. We set

$$S_{mn} = \sum_{i=0}^{m} \sum_{j=0}^{n} a_{ij}$$

and

$$S = \sum_{i,j=0}^{\infty} a_{ij}.$$

Streit's transformation is

$$T_{mn}(k,l) = S_{mn} + \frac{S_{m+k,n+l} - S_{mn}}{1 - R_{mn}(k,l)}$$

with $R_{mn}(k,l) = a_{m+k,n+l}/a_{mn}$.

Haccart's transformation is

$$\begin{aligned} T_{mn}(k,l) &= S_{mn} + \frac{S_{m+k,n} - S_{m+k,n-1}}{S_{1n} - S_{1,n-1}} \cdot (S_{1,n+l} - S_{1n}) + \\ &\quad \frac{S_{mn} - S_{m-1,n}}{S_{m1} - S_{m-1,1}} \cdot (S_{m+k,1} - S_{m1}). \end{aligned}$$

Several convergence and acceleration results on these two transformations were given but their conditions are difficult to check in practical situations and we shall not give them for that reason.

Let us apply these techniques to

$$S = \sum_{i,j=1}^{\infty} \frac{(-1)^{j+1}}{2^{i-1}j} = \int_0^1 \int_0^1 \frac{dx\,dy}{x+y} = 2\ln 2 = 1.386294361119891\ldots$$

The method of Albertsen et al. and the second method of Cuyt give respectively in columns 1 and 2

$E_2^{(3)}$	1.183518	1.292352
$E_3^{(6)}$	1.228489	1.374224
$E_5^{(9)}$	1.304007	1.359011
$E_8^{(12)}$	1.329994	1.373649
$E_{10}^{(17)}$	1.360150	1.385863
$E_{15}^{(20)}$	1.374274	1.386177
$E_{18}^{(26)}$	1.371675	1.386366
$E_{25}^{(29)}$	1.385897	1.386298

The method of Streit and the first method of Cuyt using the ε-algorithm provide

$T_{4,3}(0,1)$	$= 1.2250000$	$\varepsilon_2^{(0)}$	$= 1.3302949$
$T_{6,5}(0,1)$	$= 1.3454861$	$\varepsilon_4^{(0)}$	$= 1.3963958$
$T_{8,7}(0,1)$	$= 1.3763421$	$\varepsilon_6^{(0)}$	$= 1.3868720$
$T_{10,9}(0,1)$	$= 1.3839863$	$\varepsilon_8^{(0)}$	$= 1.3863089$
$T_{12,11}(0,1)$	$= 1.3858309$		
$T_{14,13}(0,1)$	$= 1.3862521$		

Other transformations are discussed by Wimp [467] and Higgins [228].

6.1.3 Chebyshev and Fourier series

In many applications one has to sum Chebyshev and Fourier series

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos kx + b_k \sin kx)$$

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k T_k(x)$$

where T_k is the Chebyshev polynomial of degree k. These polynomials can be recursively computed by

$$\begin{aligned} & T_0(x) = 1, \quad T_1(x) = x \\ T_{n+1}(x) &= 2x\,T_n(x) - T_{n-1}(x), \qquad n = 1, 2, \ldots \end{aligned}$$

When $x \in [-1, +1]$ they are directly given by

$$T_n = \cos(n \arccos x).$$

Such series can be transformed by means of the ε-algorithm applied to their partial sums, see, for example, Kiefer and Weiss [260]. However the usefulness of the ε-algorithm can be improved if the series to be treated are first transformed with the help of their conjugate series, thus generalizing a technique proposed by Wynn [480].

If we set $z = e^{ix}$, $c_0 = a_0/2$ and $c_k = a_k - ib_k$ for $k = 1, 2, \ldots$ then the above Fourier series becomes the real part of

$$f(z) = \sum_{k=0}^{\infty} c_k\, z^k.$$

If we set $\theta = \arccos x$ and $z = e^{i\theta}$ then $T_k(x) = \cos\theta$ and the above Chebyshev series becomes the real part of

$$f(z) = \sum_{k=0}^{\infty} c_k\, z^k$$

where $c_0 = a_0/2$ and $c_k = a_k$ for $k = 1, 2, \ldots$

Thus, in both cases, the application of the ε-algorithm to the partial sums of the series f gives its Padé approximants

$$\varepsilon_{2k}^{(n)} = [n + k/k]_f(z).$$

Let us give some numerical examples. We first consider the Fourier series

$$f(x) = 2\left(\sin x - \frac{1}{2} \cdot \sin 2x + \cdots + \frac{(-1)^{n+1}}{n} \cdot \sin nx + \cdots\right).$$

For $x \in]-\pi, \pi[$ it converges to x. Its conjugate series is

$$-2\left(\cos x - \frac{1}{2} \cdot \cos 2x + \cdots + \frac{(-1)^{n+1}}{n} \cdot \cos nx + \cdots\right)$$

and the preceding Fourier series is the real part of

$$-2i\left(z-\frac{1}{2}z^2+\cdots+\frac{(-1)^{n+1}}{n}z^n+\cdots\right)$$

where $z=e^{ix}=\cos x+i\sin x$.

If the ε-algorithm is applied to the partial sums of the Fourier series (column 2) or to the partial sums of the new series (column 3) we obtain for $x=0.2$

n	Fourier series	ε-alg. to Fourier series	ε-alg. to new series
0	0.39733866	0.39733866	0.39733866
1	$0.79203193\cdot 10^{-2}$	$0.79203193\cdot 10^{-2}$	$0.79203193\cdot 10^{-2}$
2	0.38434863	0.19932690	0.20765249
3	0.25670589	0.20067920	0.19625942
6	0.33313636	0.19999999	0.20000985

For $x=2$ we have

n	Fourier series	ε-alg. to Fourier series	ε-alg. to new series
2	$0.23891204\cdot 10^{1}$	$0.24159138\cdot 10^{1}$	$0.19878926\cdot 10^{1}$
3	$0.18944412\cdot 10^{1}$	$0.26879099\cdot 10^{1}$	$0.19735744\cdot 10^{1}$
6	$0.21387211\cdot 10^{1}$	$0.20280599\cdot 10^{1}$	$0.19998642\cdot 10^{1}$
9	$0.18612218\cdot 10^{1}$	$0.19986396\cdot 10^{1}$	$0.20000040\cdot 10^{1}$
12	$0.21278589\cdot 10^{1}$	$0.19992951\cdot 10^{1}$	$0.20000000\cdot 10^{1}$

The use of the ε-algorithm for summing certain Fourier series arising in the automatic temperature adjustment of buildings leads to a gain of time of 60 per cent, see Nowakowski [340].

6.1.4 Continued fractions

Continued fractions have important applications in numerical analysis (see Jones and Thron [251]) and in applied mathematics. In particular they are used for computing many special functions of mathematical physics, see, for example, Patry [354]. Thus an important topics is their acceleration which can be achieved either by using the methods developed in this book or by the technique of modification and converging

factors as described by Lorentzen and Waadeland [300] (see also the review paper by Jacobsen [239]). In fact, as we shall see below, these two approaches are very much complementary since extrapolation methods can be interpreted as modifications using special converging factors.

Let us recall some definitions and give our notations. Let C be a continued fraction

$$C = b_0 + \frac{a_1}{b_1} + \frac{a_2}{b_2} + \cdots = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cdots}}$$

where the a_i's and the b_i's are complex numbers. Let C_n be its n-th convergent

$$C_n = b_0 + \frac{a_1}{b_1} + \cdots + \frac{a_n}{b_n} = \frac{A_n}{B_n}$$

and R_n its n-th tail

$$R_n = \frac{a_{n+1}}{b_{n+1}} + \frac{a_{n+2}}{b_{n+2}} + \cdots$$

Then obviously

$$C = b_0 + \frac{a_1}{b_1} + \cdots + \frac{a_n}{b_n + R_n} = b_0 + \frac{a_1}{b_1} + \cdots + \frac{a_n}{b_n} + \frac{R_n}{1}.$$

Moreover

$$R_n = \frac{a_{n+1}}{b_{n+1} + R_{n+1}}$$

with $R_0 = C - b_0$.

Thus this relation cannot be used in practice for computing the sequence (R_n) since, usually, the value of C is not known.

The partial numerators and denominators A_n and B_n can be recursively computed by the well known formula

$$\begin{aligned} A_{-1} &= 1\,, \quad A_0 = b_0 \\ B_{-1} &= 0\,, \quad B_0 = 1 \\ A_n &= b_n A_{n-1} + a_n A_{n-2} \\ B_n &= b_n B_{n-1} + a_n B_{n-2}\,, \qquad n = 1, 2, \ldots \end{aligned}$$

Thus we have

$$C = \frac{A_n + R_n A_{n-1}}{B_n + R_n B_{n-1}}\,, \qquad n = 0, 1, \ldots$$

Since R_n is unknown we shall replace it by an approximation r_n. Thus setting

$$S_n(w) = \frac{A_n + w\,A_{n-1}}{B_n + w\,B_{n-1}}, \qquad n = 0, 1, \ldots$$

we see that

$$\begin{aligned} &S_n(0) = C_n \\ &S_n(a_{n+1}/b_{n+1}) = C_{n+1} \\ &\lim_{w \to \pm\infty} S_n(w) = C_{n-1} \\ &S_n(R_n) = C. \end{aligned}$$

Replacing (R_n) by (r_n) and considering the sequence $(C_n^* = S_n(r_n))$ is called a modification of the continued fraction and (r_n) is called a sequence of converging factors. The problem is the choice of (r_n) such that (C_n^*) converges to C faster than (C_n) (which is always assumed to converge in the sequel).

General results in this direction were given by Brezinski [73].

From the relation

$$\frac{C_n^* - C}{C_n - C} = \frac{1 - d_n}{1 - d_n \cdot (C_n - C)/(C_{n-1} - C)},$$

with $d_n = r_n/R_n$, one can easily prove the two following theorems

Theorem 6.1

If $\exists \alpha < 1 < \beta, \exists N, \forall n \geq N, (C_n - C)/(C_{n-1} - C) \notin [\alpha, \beta]$ *and if* $\lim_{n \to \infty} r_n/R_n = 1$ *then* $\lim_{n \to \infty} (C_n^* - C)/(C_n - C) = 0$.

If the ratio $(C_n - C)/(C_{n-1} - C)$ tends to 1 let us set

$$(C_n - C)/(C_{n-1} - C) = 1 + \beta_n$$

$$r_n/R_n = 1 + \alpha_n.$$

We have the

Theorem 6.2

If $\lim_{n \to \infty} \alpha_n = \lim_{n \to \infty} \beta_n = \lim_{n \to \infty} \alpha_n/\beta_n = 0$ *then* $\lim_{n \to \infty} (C_n^* - C)/(C_n - C) = 0$.

Let us now show how to connect converging factors to acceleration methods or to the equivalent notion of perfect estimation of the error as explained in section 1.4.

Let (D_n) be an arbitrary sequence and let us take

$$r_n = -\frac{D_n h_n}{D_n + C_n - C_{n-1}}$$

where $h_n = B_n/B_{n-1}$. Then it is easy to see that

$$S_n(r_n) = C_n + D_n$$

$$\frac{S_n(r_n) - C}{C_n - C} = 1 - \frac{D_n}{C - C_n}.$$

Thus $(S_n(r_n))$ converges to C faster than (C_n) if and only if (D_n) is a perfect estimation of the error of (C_n) that is $\lim_{n\to\infty} D_n/(C - C_n) = 1$.

Conversely if a sequence (r_n) such that $\lim_{n\to\infty} (S_n(r_n) - C)/(C_n - C) = 0$ is known then

$$D_n = -r_n \cdot \frac{C_n - C_{n-1}}{h_n + r_n}$$

is a perfect estimation of the error of (C_n).

On the other hand if we are given a sequence transformation $T : (C_n) \longmapsto (T_n)$ of the form

$$T_n = \frac{C_n\, g_{n+1} - C_{n+1}\, g_n}{g_{n+1} - g_n} = C_{n+1} - \frac{C_{n+1} - C_n}{g_{n+1} - g_n} \cdot g_{n+1}$$

then $T_n = S_{n+1}(r_{n+1})$ with $r_{n+1} = -B_{n+1}\, g_{n+1}/B_n\, g_n$.

Thus any sequence transformation of this form (which includes Richardson extrapolation method, Aitken's Δ^2 process, the E-algorithm, Overholt's process, ...) can be considered as a modification of the continued fraction and reciprocally.

We shall see later a particular case where (r_n) can be chosen according to the conditions of the above theorems but, for the moment, let us consider a very instructive example, namely

$$C = 1 + \frac{a\,\rfloor}{\lceil 1} + \frac{a\,\rfloor}{\lceil 1} + \cdots$$

whose convergents satisfy

$$C_0 = 1$$
$$C_{n+1} = 1 + \frac{a}{C_n}, \qquad n = 0, 1, \ldots$$

The case $a = 1$ corresponds to the continued fraction of Fibonacci and $C = (1 + \sqrt{5})/2$ is the famous golden section. This case was considered by Phillips [357] whose used the iterated Aitken's Δ^2 process to accelerate its convergence. The case of an arbitrary value of a was treated by Brezinski and Lembarki [94] who accelerate its convergence by the iterated Δ^2 process and by the ε-algorithm. In both cases such an extrapolation method produces a subsequence of (C_n).

If $a \neq -1/4 + c$ where c is a real non positive number then (C_n) converges to x_1 the zero of greatest modulus of $x^2 - x - a = 0$. Otherwise (C_n) does not converge. Let $\left(x_k^{(n)}\right)_n$ be the sequence obtained by the k-th iteration of Aitken's process (see section 2.10) to the sequence of convergents of $C = b + \left\rfloor \frac{a}{1} \right\rfloor + \left\rfloor \frac{a}{1} \right\rfloor + \cdots$ and let $\left(\varepsilon_{2k}^{(n)}\right)$ be the sequence obtained by application of the ε-algorithm. Then we have

Theorem 6.3

$$x_k^{(n)} = C_{(n+k+1)2^k-1}$$
$$\varepsilon_{2k}^{(n)} = C_{n(k+1)+k(k+2)}.$$

This result is very much interesting since it shows the acceleration brought by both processes. We shall now assume that $a \neq -1/4 + c$ with $c < 0$. The main results are the following (for the whole bunch of them see the paper by Brezinski and Lembarki [94]). We set $r = x_2/x_1$ where x_2 is the zero of smallest modulus of $x^2 - x - a = 0$ and $x_1 = C$ the other one.

Theorem 6.4

If $a \neq -1/4 + c$ with $c < 0$ then

1. $\lim_{n\to\infty} \left(x_k^{(n+1)} - C\right) \Big/ \left(x_k^{(n)} - C\right) = r^{2^k}$
2. $\lim_{n\to\infty} \left(x_{k+1}^{(n)} - C\right) \Big/ \left(x_k^{(n+2)} - C\right) = 0$

3. $\lim_{k\to\infty} \left(x_{k+1}^{(n)} - C\right) / \left(x_k^{(n)} - C\right)^2 = 0$

4. $\lim_{k\to\infty} \left(x_k^{(n+1)} - C\right) / \left(x_k^{(n)} - C\right) = 0$

5. $\lim_{n\to\infty} \left(\varepsilon_{2k}^{(n+1)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = r^{k+1}$

6. $\lim_{n\to\infty} \left(\varepsilon_{2k+2}^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n+2)} - C\right) = 0$

7. $\lim_{k\to\infty} \left(\varepsilon_{2k+2}^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = 0$

8. $\lim_{k\to\infty} \left(\varepsilon_{2k}^{(n+1)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = 0$

9. $\lim_{n\to\infty} \left(x_k^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = \lim_{k\to\infty} \left(x_k^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = 0,$
the first result holding only for $k \geq 2$ *since* $x_1^{(n)} = \varepsilon_2^{(n)}$.

If $a = -1/4$ *then*

10. $\lim_{n\to\infty} \left(x_k^{(n+1)} - C\right) / \left(x_k^{(n)} - C\right) = 1$

11. $\lim_{n\to\infty} \left(x_{k+1}^{(n)} - C\right) / \left(x_k^{(n+2)} - C\right) = 1/2$

12. $\lim_{k\to\infty} \left(x_{k+1}^{(n)} - C\right) / \left(x_k^{(n)} - C\right) = 1/2$

13. $\lim_{k\to\infty} \left(x_k^{(n+1)} - C\right) / \left(x_k^{(n)} - C\right) = 1$

14. $\lim_{n\to\infty} \left(\varepsilon_{2k}^{(n+1)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = 1$

15. $\lim_{n\to\infty} \left(\varepsilon_{2k+2}^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n+2)} - C\right) = (k+1)/(k+2)$

16. $\lim_{k\to\infty} \left(\varepsilon_{2k+2}^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = 1$

17. $\lim_{k\to\infty} \left(\varepsilon_{2k}^{(n+1)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = 1$

18. $\lim_{n\to\infty} \left(x_k^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right) = (k+1)/2^k$ *and*
$\forall q \geq 1 \lim_{k\to\infty} \left(x_k^{(n)} - C\right) / \left(\varepsilon_{2k}^{(n)} - C\right)^q = 0.$

Let us comment these results.

When $a \neq -1/4 + c$ with $c < 0$, the sequence (C_n) converges linearly. For the iterated application of Aitken's process, each column (that is k fixed and n tending to infinity) converges linearly (result 1) but faster

than the preceding one (result 2). Result 3 is very important since it shows that a linearly convergent sequence can be transformed into a sequence converging super-quadratically. Result 4 claims that each diagonal converges faster than the preceding one. If the ε-algorithm is used, each column converges linearly (result 5) but faster than the preceding one (result 6). Result 7, although less important than result 3, shows that a linearly convergent sequence can be transformed into a diagonal converging super-linearly. Each diagonal converges faster than the preceding one (result 8). Result 9 shows that Aitken's iterated process is always a better accelerator than the ε-algorithm.

When $a = -1/4$, the sequence (C_n) converges logarithmically. For the iterated application of Aitken's process, each column also converges logarithmically (result 10) and not faster than the preceding one (result 11). Result 12 is very important since it shows that a logarithmic sequence can be transformed into a linear one (which thus can be accelerated again by Aitken's process). However there is no gain from one diagonal to the following one (result 13). If the ε-algorithm is used, each column is logarithmic (result 14) and does not converge faster than the preceding one (result 15). Each diagonal is logarithmic (result 16) and there is no gain from one diagonal to the following one (result 17). Finally result 18 shows that Aitken's iterated process provides better results than the ε-algorithm.

A continued fraction of the form $C = b + \left\lfloor \frac{a}{b} \right\rfloor + \left\lfloor \frac{a}{b} \right\rfloor + \cdots$ is called 1-periodic (or periodic for short). The Fibonacci continued fraction has this form. We shall now consider a general continued fraction satisfying $\lim_{n\to\infty} a_n = a$ and $\lim_{n\to\infty} b_n = b$. Such a continued fraction is called limit periodic. It can always be transformed into an equivalent continued fraction (that is with the same sequence of convergents) where all the b_n's are equal to 1. Thus let us consider the continued fraction

$$C = \left\lfloor \frac{a_1}{1} \right\rfloor + \left\lfloor \frac{a_2}{1} \right\rfloor + \cdots$$

where the a_n's are complex numbers tending to a limit a. Before trying to accelerate this continued fraction we need to study its asymptotic behaviour. A well known and old result says that such a continued fraction converges linearly under some conditions on a. The reciprocal of this result was obtained by Brezinski and Lembarki [95] who proved

the

Theorem 6.5

Let us consider the continued fraction $C = \dfrac{a_1}{\lceil 1 \rfloor} + \dfrac{a_2}{\lceil 1 \rfloor} + \cdots$ *with* $\lim_{n\to\infty} a_n = a$. *A necessary and sufficient condition that there exists a complex number* r *with* $|r| < 1$ *such that* $\lim_{n\to\infty} (C_{n+1} - C)/(C_n - C) = r$ *is that there exists a complex number* a *with* $a \neq -1/4 + c$ *and* $c \leq 0$ *such that* $\lim_{n\to\infty} a_n = a$. *Moreover* a *and* r *are related by* $a = -r/(1+r)^2$ *and* $r = -x_1/(1+x_1)$ *where* x_1 *is the zero of smallest modulus of* $x^2 + x - a = 0$.

The condition on a insures the convergence of the continued fraction. $r = 0$ if and only if $a = 0$ and if $r = 1$ then $a = -1/4$. This theorem has important consequences since it does not only show that limit periodic continued fractions converge linearly if $a \neq 0$ but that they are the only ones to do so. It means that a continued fraction converging linearly with $|r| < 1$ is necessarily limit periodic. It follows from the theory of remanence explained in section 1.10 that a universal algorithm for accelerating the convergence of all continued fractions which are not limit periodic cannot exist.

Let us now see how to accelerate limit periodic continued fractions. This goal can be achieved either by a modification with constant converging factors or by Aitken's Δ^2 process which is known to accelerate linearly converging sequences. For such a continued fraction it can be proved that $\lim_{n\to\infty} R_n = x_1$, the zero of smallest modulus of $x^2 + x - a = 0$. Thus, according to theorem 6.1, we must choose a sequence (r_n) also tending to x_1 and the easiest choice is to take $r_n = x_1$ for all n. This choice was extensively studied and it was proved by Niethammer and Wietschorke [335] that

$$\lim_{n\to\infty} (S_{n+1}(x_1) - C)/(S_n(x_1) - C) = r^* = r\,\delta$$

where $\delta = \lim_{n\to\infty} (a_{n+1} - a)/(a_n - a)$ is assumed to exist and it can be proved that, if $r \neq 0$, the sequence $(S_n(x_1))$ converges faster than $(S_n(0) = C_n)$. But now, thanks to the result of theorem 6.5, the same procedure can be applied iteratively as explained by Brezinski [75]. The

sequence $(S_n(x_1))$ can be considered as the successive convergents of the continued fraction

$$C^* = b_0^* + \frac{a_1^*}{1} + \frac{a_2^*}{1} + \cdots$$

where the a_i^*'s and b_0^* have expressions given by Thron and Waadeland [433]. Since $(S_n(x_1))$ converges linearly then, by theorem 6.5, the continued fraction C^* is limit periodic and the modification process can be again applied. The method and the results can be summarized as follows.

We consider the continued fraction

$$C = b_0 + \frac{a_1}{1} + \frac{a_2}{1} + \cdots$$

with $\lim_{n\to\infty} a_n = a \neq -1/4 + c$ with $c \leq 0$. We assume that $a \neq 0$.

1. We set $b_0^{(0)} = b_0$ and $a_n^{(0)} = a_n$ for $n = 1, 2, \ldots$ We set $k = 0$ and $a^{(0)} = a$.

2. Let $C_n^{(k)} = A_n^{(k)} / B_n^{(k)}$ be the successive convergents of

$$C^{(k)} = b_0^{(k)} + \frac{a_1^{(k)}}{1} + \frac{a_2^{(k)}}{1} + \cdots$$

We set $S_n^{(k)}(w) = \left(A_n^{(k)} + wA_{n-1}^{(k)}\right) / \left(B_n^{(k)} + wB_{n-1}^{(k)}\right)$, $n = 0, 1, \ldots$

3. Let $x_1^{(k)}$ be the zero of smallest modulus of $x^2 + x - a^{(k)} = 0$. We set $r^{(k)} = -x_1^{(k)} / \left(1 + x_1^{(k)}\right)$. Then

$$\lim_{n\to\infty} \left(S_n^{(k)}\left(x_1^{(k)}\right) - C\right) / \left(C_n^{(k)} - C\right) = 0.$$

4. We assume that there exists $\delta^{(k)}$ such that $\lim_{n\to\infty} \delta_{n+1}^{(k)} / \delta_n^{(k)} = \delta^{(k)}$ with $\delta_n^{(k)} = a_n^{(k)} - a^{(k)}$. Then we have

$$\lim_{n\to\infty} \left(S_{n+1}^{(k)}\left(x_1^{(k)}\right) - C\right) / \left(S_n^{(k)}\left(x_1^{(k)}\right) - C\right) = r^{(k)}\delta^{(k)} = r^{(k+1)}.$$

If $\delta^{(k)} = 0$ or if $\delta^{(k)}$ does not exist, the procedure has to be stopped.

5. We set

$$C^{(k+1)} = b_0^{(k+1)} + \cfrac{a_1^{(k+1)}}{1} + \cfrac{a_2^{(k+1)}}{1} + \cdots$$

with

$$b_0^{(k+1)} = x_1^{(k)} + b_0^{(k)}, \quad a_1^{(k+1)} = \delta_1^{(k)} \Big/ \left(1 + x_1^{(k)}\right), \quad \gamma_1^{(k)} = 0,$$

$$a_n^{(k+1)} = a_{n-1}^{(k)} \gamma_n^{(k)} \Big/ \left[\left(1 + x_1^{(k)}\right)^2 \left(1 + r^{(k)} \gamma_n^{(k)}\right)\left(1 + r^{(k)} \gamma_{n-1}^{(k)}\right)\right], n = 1, 2, \ldots$$

and $\gamma_n^{(k)} = \delta_n^{(k)} \Big/ \delta_{n-1}^{(k)}$ for $n = 2, 3, \ldots$

We have

$$\lim_{n\to\infty} a_n^{(k+1)} = -r^{(k+1)} \Big/ \left(1 + r^{(k+1)}\right)^2 = a^{(k+1)} \neq -1/4 + c$$

with $c \leq 0$ and $a^{(k+1)} \neq 0$. We also have

$$x_1^{(k+1)} = -r^{(k+1)} \Big/ \left(1 + r^{(k+1)}\right).$$

6. Replace k by $k + 1$ and return to 2.

As proved by Levrie and Bultheel [287] this procedure is equivalent to the method proposed by Jacobsen and Waadeland [241, 242].

Let us consider the example

$$\ln(1 + z) = \cfrac{z}{1} + \cfrac{z/2}{1} + \cfrac{z/6}{1} + \cdots$$

that is $a_1 = z, a_{2n} = \dfrac{n}{2(2n-1)} \cdot z$ and $a_{2n+1} = \dfrac{n}{2(2n+1)} \cdot z$ for $n = 1, 2, \ldots$

Then $\lim\limits_{n\to\infty} a_n = z/4$ and $\delta^{(0)} = -1$. If $z = 24$ we have $x_1^{(0)} = 2, r^{(0)} = -2/3, x_1^{(1)} = -2/5$ and $r^{(1)} = 2/3$. Moreover

$$\delta_{n+1}^{(1)} \Big/ \delta_n^{(1)} = 1 - \lambda_n \quad \text{with} \quad \lambda_n \geq 0 \quad \text{and} \quad \lim_{n\to\infty} \lambda_n = 0.$$

The ratio λ_{n+1}/λ_n takes alternatively the values 0 and $+\infty$ and $\lim\limits_{n\to\infty} \delta_{2n+1}^{(2)} \Big/ \delta_{2n}^{(2)} = \lim\limits_{n\to\infty} \delta_{2n+1}^{(2)} \Big/ \delta_{2n+2}^{(2)} = -7/3$. The continued fraction $C^{(3)}$ is not limit periodic and the procedure has to be stopped.

Instead of modification let us now use Aitken's Δ^2 process to accelerate the convergence. As explained at the beginning of this section, Aitken's process can be interpreted as a modification of the continued fraction and it corresponds to the choice

$$r_n = \frac{a_{n+1}}{h_{n+1}}.$$

We can prove that $\lim_{n\to\infty} r_n = x_1$ and we have the

Theorem 6.6

Let (C_n^*) be the sequence obtained by applying Aitken's Δ^2 process to (C_n) where $a \neq -1/4 + c$ with $c \leq 0$. Then

$$\lim_{n\to\infty} (C_n^* - C)/(C_n - C) = 0.$$

It can be proved that, under some additional assumptions, (C_n^*) converges linearly and thus Aitken's process can be applied again as we did for modifications.

It is also possible to accelerate limit periodic continued fractions by the T_{+m} transformation as showed by Lembarki [280]. This transformation, defined by Gray and Clark [198], is a generalization of Aitken's process which is recovered for $m = 1$

$$T_n^{(m)} = \frac{C_{n+m}\Delta C_n - C_n \Delta C_{n+m}}{\Delta C_n - \Delta C_{n+m}}, \qquad n = 0, 1, \ldots$$

We have the

Theorem 6.7

If the T_{+m} transformation is applied to a limit periodic continued fraction with $a \neq 0$ and $a \neq -1/4 + c$ and $c \leq 0$ then $\forall m \geq 1$ and $\forall k \geq 0$

$$\lim_{n\to\infty} \left(T_n^{(m)} - C\right) \Big/ (C_{n+k} - C) = 0.$$

This transformation was compared to $(S_n(x_1))$ by Lembarki [280].

The preceding results show the importance of the knowledge of the asymptotic behaviour of continued fractions for their acceleration. This is the reason why this question was studied by Brezinski [78].

First, as proved by Lembarki [280], any continued fraction

$$C = b_0 + \cfrac{a_1}{b_1} + \cfrac{a_2}{b_2} + \cdots$$

with convergents $C_n = A_n / B_n$ can be transformed, if $\forall n$, $B_n \neq 0$, into the equivalent one

$$C' = b_0 + \cfrac{a_1}{1} + \cfrac{a'_2}{1 - a'_2} + \cfrac{a'_3}{1 - a'_3} + \cdots$$

such that its convergents $C'_n = C_n = A'_n / B'_n$ are such that $\forall n$, $B'_n = 1$. Thus in that case the recurrence relation reduces to

$$\begin{aligned} & C'_0 = b_0\,, \quad C'_1 = b_0 + a_1 \\ C'_n \;=\; & (1 - a'_n)C'_{n-1} + a'_n C'_{n-2}\,, \qquad n = 2, 3, \ldots \end{aligned}$$

and we have

$$a'_n = -\frac{\Delta C_{n-1}}{\Delta C_{n-2}}\,, \qquad n = 2, 3, \ldots$$

Since $C_n = b_0 + a_1 - a_1 \cdot a_2 + \cdots + (-1)^{n-1} a_1 \cdot \ldots \cdot a_n$ then the usual convergence tests for series can be used for obtaining convergence conditions for the continued fraction. In particular d'Alembert's test says that if there exists $K < 1$ and N such that $\forall n \geq N, |a_n| \leq K$, then (C_n) converges.

As explained in section 3.2 these convergence tests can also be used for the construction of perfect estimations of the error and thus convergence acceleration methods.

Thus let us consider a continued fraction of the form

$$C = b_0 + \cfrac{a_1}{1} + \cfrac{a_2}{1 - a_2} + \cfrac{a_3}{1 - a_3} + \cdots$$

Let $0 < p_0 < p_1 < \cdots$ be an infinite strictly increasing sequence of positive integers and let

$$C' = b'_0 + \cfrac{a'_1}{1} + \cfrac{a'_2}{1 - a'_2} + \cfrac{a'_3}{1 - a'_3} + \cdots$$

be the continued fraction with convergents $C'_n = C_{p_n}$ for $n = 0, 1, \ldots$ Let r be a complex number. The main results are the following

Theorem 6.8

A necessary and sufficient condition that

$$\lim_{n\to\infty} (C_{p_{n+1}} - C_{p_n}) / (C_{p_n} - C_{p_{n-1}}) = r$$

is that $\lim_{n\to\infty} a'_n = -r$.

This result does not assume the convergence of the continued fractions C and C'. Using some results due to Delahaye [134] (see section 1.12) we have the

Theorem 6.9

We assume that $\forall n, C'_n \neq C'_{n+1}$.

i) If $|r| \neq 1$ *then there exists* $x \in \mathbb{C}$ *such that* $\lim_{n\to\infty} (C'_{n+1} - x)/(C'_n - x) = r$ *if and only if* $\lim_{n\to\infty} \Delta C'_{n+1} / \Delta C'_n = r$.

ii) If $|r| < 1$ *and if* $\lim_{n\to\infty} \Delta C'_{n+1} / \Delta C'_n = r$ *then* (C'_n) *converges to a limit* C' *and* $\lim_{n\to\infty} (C'_{n+1} - C')/(C'_n - C') = r$.

iii) If $|r| > 1$ *and if* $\lim_{n\to\infty} \Delta C'_{n+1} / \Delta C'_n = r$ *then* $\lim_{n\to\infty} |C'_n| = \infty$ *and* $\forall x \in \mathbb{C}$, $\lim_{n\to\infty} (C'_{n+1} - x)/(C'_n - x) = r$.

It must be noticed that the statement *i)* does not imply the convergence of (C'_n) to x.

Finally we have the following result on the continued fraction C itself

Theorem 6.10

A necessary and sufficient condition that

$$\lim_{n\to\infty} (C_{p_n+1} - C_{p_n}) / (C_{p_n} - C_{p_n-1}) = r$$

is that $\lim_{n\to\infty} a_{p_n} = -r$.

These results generalize those on limit periodic continued fractions (theorem 6.5).

Another interesting case is that of limit k-periodic continued fractions that is such that

$$\lim_{n\to\infty} a_{m+nk} = r_m\,, \qquad m = 0, 1, \ldots, k-1$$

where $r_0, \ldots, r_{k-1}$ are complex numbers. Of course $r_m = r_{m+k}$. Their asymptotic behaviour follows from the preceding results and we have the

Theorem 6.11

A necessary and sufficient condition that

$$\lim_{n\to\infty} (C_{m+nk} - C_{m+nk-1})/(C_{m+nk-1} - C_{m+nk-2}) = r_m$$

is that $\lim_{n\to\infty} a_{m+nk} = -r_m$ *for* $m = 0, \ldots, k-1$.

Other results of this type are given by Brezinski [78] who also obtained the acceleration result

Theorem 6.12

Let us assume that $|r_0 \cdot \ldots \cdot r_{k-1}| < 1$ *and that*

$$\begin{aligned}
&1 + r_0 + r_0 \cdot r_1 + \cdots + r_0 \cdot r_1 \cdot \ldots \cdot r_{k-2} \neq 0\\
&1 + r_1 + r_1 \cdot r_2 + \cdots + r_1 \cdot r_2 \cdot \ldots \cdot r_{k-1} \neq 0\\
&\vdots\\
&1 + r_{k-1} + r_{k-1} \cdot r_0 + \cdots + r_{k-1} \cdot r_0 \cdot \ldots \cdot r_{k-3} \neq 0.
\end{aligned}$$

If, for $m = 0, \ldots, k-1$, $\lim_{n\to\infty} (C_{m+nk+2} - C_{m+nk+1})/(C_{m+nk+1} - C_{m+nk}) = r_m$ *then* (C_n) *converges to a limit* C *and* $\lim_{n\to\infty} (C_{m+nk+1} - C)/(C_{m+nk} - C) = q_m$ *where*

$$\begin{aligned}
q_0 &= \frac{r_0 + r_0 \cdot r_1 + \cdots + r_0 \cdot r_1 \cdot \ldots \cdot r_{k-1}}{1 + r_0 + r_0 \cdot r_1 + \cdots + r_0 \cdot r_1 \cdot \ldots \cdot r_{k-2}}\\
q_1 &= \frac{r_1 + r_1 \cdot r_2 + \cdots + r_1 \cdot r_2 \cdot \ldots \cdot r_0}{1 + r_1 + r_1 \cdot r_2 + \cdots + r_1 \cdot r_2 \cdot \ldots \cdot r_{k-1}}\\
&\vdots \qquad \vdots\\
q_{k-1} &= \frac{r_{k-1} + r_{k-1} \cdot r_0 + \cdots + r_{k-1} \cdot r_0 \cdot \ldots \cdot r_{k-2}}{1 + r_{k-1} + r_{k-1} \cdot r_0 + \cdots + r_{k-1} \cdot r_0 \cdot \ldots \cdot r_{k-3}}.
\end{aligned}$$

Moreover the sequence $\left(T_n^{(k)}\right)_n$ *obtained by the* T_{+k} *transformation converges to* C *faster than* (C_n).

Other acceleration results for limit k-periodic continued fractions are due to Lembarki [281] and others (see the bibliographies of the papers contained in Brezinski [87]).

The case of k-periodic continued fractions is treated by Lembarki [279] in his thesis.

At the end of section 3.3 (theorem 3.17) we saw how to accelerate the convergence of sequences for which an asymptotic expansion of the inverse of the error is known. As shown by Matos [313] these results apply to some continued fractions.

If we consider the case of

$$C = 1 + \cfrac{a_1}{1} + \cfrac{a_2}{1} + \cdots$$

with $a_i = i^m \left(\alpha_0 + \alpha_1 i^{-1} + \alpha_2 i^{-2} + \cdots\right)$ where m is an integer, then it was proved by Hautot [215] that

$$C_n - C = (b_0 + b_1 g_1(n) + b_2 g_2(n) + \cdots)^{-1}$$

where the g_i's form an asymptotic sequence. Then the algorithm given in section 3.3 can be used.

For example for the continued fraction with $a_i = -1/4 - 0.009/i^2$ the number of exact digits is multiplied by 6.2. If $a_i = -(2i+3)^2/16(i+2)(i+3)$ the number of exact digits is multiplied by 6.6.

These results were used by Draux [144] for accelerating sequences of Padé approximants.

6.1.5 Vector sequences

In numerical analysis, vector sequences which need to be accelerated are usually produced by iterations of the form

$$x_{n+1} = Ax_n + b, \qquad n = 0, 1, \ldots$$

where x_0 and b are given arbitrary vectors of dimension p and A a $p \times p$ matrix.

The matrix $I - A$ is assumed to be regular and we shall denote by x the vector satisfying

$$x = Ax + b.$$

Let $\lambda_1, \ldots, \lambda_p$ be the eigenvalues of A and $v_1, \ldots, v_p$ the corresponding eigenvectors. If the vectors $v_1, \ldots, v_p$ are linearly independent, if $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_p|$, and if $(x_0, v_i) \neq 0$ and $(y, v_i) \neq 0$ for $i = 1, \ldots, p$ then Brezinski [49] proved the

Theorem 6.13

If the vector (or the topological) ε-algorithm is applied to the sequence (x_n) then, for $k = 0, \ldots, p-1$

$$\varepsilon_{2k}^{(n)} - x \sim \sum_{i=k+1}^{p} \lambda_i^{n+k} z_i \qquad (n \to \infty)$$

$$\varepsilon_{2k+1}^{(n)} \sim \frac{1}{\lambda_{k+1}^{n+k}} \cdot \frac{y_{k+1}}{(y_{k+1}, y_{k+1})} \qquad (n \to \infty)$$

where the y_i's and the z_i's are vectors related to the v_i's.

If $|\lambda_1| < 1$, the sequence (x_n) converges to x and the rate of convergence is $O(\lambda_1^n)$. When applying the ε-algorithm to (x_n) then, for $k = 0, \ldots, p-1$, each sequence $\left(\varepsilon_{2k}^{(n)}\right)_n$ also converges to x and the rate of convergence is $O\left(\lambda_{k+1}^n\right)$. This shows that the gain brought by the ε-algorithm depends on the distance between consecutive eigenvalues. Now if $|\lambda_1| > \cdots > |\lambda_r| > 1 > |\lambda_{r+1}| > \cdots > |\lambda_p|$, then (x_n) diverges. However we can still apply the ε-algorithm to (x_n). The sequences $\left(\varepsilon_{2k}^{(n)}\right)_n$ will diverge for $k = 0, \ldots, r-1$ but they will converge to x for $k = r, \ldots, p-1$ and the rate of convergence will be $O\left(\lambda_{k+1}^n\right)$. Thus the ε-algorithm can accelerate the convergence of such sequences and even transform a diverging sequence into converging ones. The same type of results can be obtained by the iterated vector Δ^2 process. Numerical results can be found in Wynn [473] and Brezinski [56].

The previous theorem was extended by Sidi, Ford and Smith [408] who proved the

Theorem 6.14

If $v_1, \ldots, v_p$ are linearly independent, if $|\lambda_1| \geq \cdots \geq |\lambda_r| > |\lambda_{r+1}| \geq |\lambda_{r+2}| \geq \cdots \geq |\lambda_p|$, and if $(y, v_i) \neq 0$ for $i = 1, \ldots, r$ then the vectors $\varepsilon_{2r}^{(n)}$ obtained by applying the topological ε-algorithm to (x_n) satisfy

$$\varepsilon_{2r}^{(n)} - x = \Lambda(n) \lambda_{r+1}^n (1 + o(1)), \qquad (n \to \infty)$$

where $\Lambda(n)$ is a nonzero and bounded vector for all sufficiently large n.

This result was still extended to the case where the matrix A is defective by Sidi and Bridger [407]. We recall that a matrix is said to be

defective if the algebraic multiplicity of one of its eigenvalues exceeds its geometric multiplicity (that is the number of linearly independent eigenvectors associated with it). In other words A is nondefective if and only if there exists a nonsingular matrix X such that $X^{-1}AX = \operatorname{diag}(\lambda_1, \ldots, \lambda_p)$, see Golub and van Loan [187]. If A is defective there exists a regular matrix X such that $X^{-1}AX = \operatorname{diag}(J_1, \ldots, J_\nu)$ where J_i is a Jordan block of dimension r_i of the form

$$J_i = \begin{pmatrix} \lambda_i & 1 & & & \\ & \ddots & \ddots & & 0 \\ & & \ddots & \ddots & \\ & 0 & & \ddots & 1 \\ & & & & \lambda_i \end{pmatrix}.$$

Let us denote by $v_{11}, v_{12}, \ldots, v_{1r_1}, v_{21}, v_{22}, \ldots, v_{2r_2}, \ldots, v_{\nu 1}, v_{\nu 2}, \ldots, v_{\nu r_\nu}$ the columns of X. Then v_{j1} is the eigenvector corresponding to λ_j and v_{ji} are the principal vectors corresponding to λ_j for $i = 2, \ldots, r_j$. These vectors are linearly independent and we have

$$x_0 - x = \sum_{j=1}^{\nu} \sum_{i=1}^{r_j} a_{ji} v_{ji}.$$

Let p_j be the largest integer for which $a_{j,p_j+1} \neq 0$. We have the

Theorem 6.15

If $|\lambda_r| > |\lambda_{r+1}| = \cdots = |\lambda_{r+t}| > |\lambda_{r+t+1}|$, *the vectors* $\varepsilon_{2k}^{(n)}$ *obtained by applying the topological* ε*-algorithm to* (x_n) *are such that*

$$\varepsilon_{2k}^{(n)} - x = \Lambda(n) n^{p_{r+1}} |\lambda_{r+1}|^n$$

where $k = \sum_{j=1}^{r} (p_j + 1)$ *and where* $\Lambda(n)$ *is a bounded vector, provided*

$$\prod_{j=1}^{r} \left(y, y_{jp_j}\right) \neq 0$$

where

$$y_{jq} = \left(\sum_{i=q+1}^{p_j+1} a_{ji} v_{j,i-q} \right) \lambda_j^{-q}, \qquad \text{for } q = 0, \ldots, p_j.$$

The proof of this result, which is quite difficult, and numerical examples can be found in Sidi and Bridger [407].

Since the vectors $\varepsilon_{2k}^{(n)}$ are usually better approximations of x than x_{n+2k}, a useful idea is that of *cycling*

- at the beginning of the $(n+1)$-th iteration x_n is known
- we set $u_0 = x_n$ and compute $u_{i+1} = Au_i + b$ for $i = 0, \ldots, 2k$ for some value of k
- then the ε-algorithm is applied to $u_0, \ldots, u_{2k}$ and it gives $\varepsilon_{2k}^{(0)}$
- then we set $x_{n+1} = \varepsilon_{2k}^{(0)}$ and begin the next iteration.

Such an idea is known for a long time. In particular, as we shall see below, if it is applied to a system of nonlinear equations instead of linear ones and if $k = p$, the dimension of the system, then the sequence (x_n) thus produced converges quadratically to x under some assumptions (see section 6.2.4).

For linear systems, cycling can be very effective if k is small compared to p. Gander, Golub and Gruntz [172] have recently applied this technique to systems arising from the discretization using centered differences of elliptic partial differential equations with Dirichlet boundary conditions. Up to 10 digits were gained during the extrapolation phase. The method also works even if the basic iterations $u_{i+1} = Au_i + b$ are not convergent. This technique have applications in statistics as shown by Gander and Golub [171]. It needs further theoretical research.

Numerical results showing the gain brought by the ε-algorithm (without cycling) were given by Espinoza [152] for a system of equations obtained by discretization of a partial differential equation by finite differences. The gain can be above 40%.

Another acceleration process was proposed by Iguchi [236]. It consists in constructing a new sequence (y_n) by

$$y_n = x_{n+2} + w_n (x_{n+2} - x_n)$$

with

$$w_n = -\frac{(\Delta x_{n+1}, \Delta x_{n+1})}{(\Delta^2 x_n, x_{n+2} - x_n)}.$$

Other generalizations of Aitken's process or the ε-algorithm for accelerating sequences generated by a matrix iterative method were given by Baron and Wajc [14] and by Miellou [321].

A general acceleration result for some vector sequences was recently given by Sadok [384]. We consider a vector sequence transformation $T : (S_n) \longmapsto (T_n)$ of the form

$$T_n = F(S_n, \ldots, S_{n+k}) , \qquad n = 0, 1, \ldots$$

where $S_n \in \mathbb{C}^p$ and $F : (\mathbb{C}^p)^{k+1} \longmapsto (\mathbb{C}^p)$. F is assumed to be quasi-linear (see chapter 4) and thus there exists f such that $F(x_0, \ldots, x_k) = [Df(x_0, \ldots, x_k)]^{-1} f(x_0, \ldots, x_k)$ with $D^2 f(x_0, \ldots, x_k)$ identically zero (theorem 4.1). We have the following acceleration result

Theorem 6.16

Let (S_n) be a sequence of vectors such that

$$S_{n+1} - S = (A + B_n)(S_n - S) , \qquad n = 0, 1, \ldots$$

where A is a square matrix with $\varrho(A) < 1$ and (B_n) a sequence of square matrices converging to zero. If $\exists \alpha > 0$ such that $\forall x \in \{v \in \mathbb{C}^p | \, \|v\| = 1\}$, $\left|\det Df\left(x, Ax, \ldots, A^k x\right)\right| \geq \alpha$ and $f\left(x, Ax, \ldots, A^k x\right) = 0$ then

$$\|T_n - S\| = o(\|S_n - S\|) , \qquad (n \to \infty).$$

This result generalizes theorem 1.8 to the vector case. It is true, for example, for Henrici's transformation (see section 4.5) but the vector generalization of the scalar E-algorithm given by Wimp [467] is not translative in general and thus it does not always accelerate the convergence of such a sequence (S_n).

A review of extrapolation methods for vector sequences can be found in Smith, Ford and Sidi [415] (see also Smith, Ford and Sidi [416]).

6.2 Systems of equations

As we shall see in this section some of the previous algorithms can be used for the solution of systems of equations. If the system is a linear one then the algorithms lead to the exact solution after a finite number of arithmetical operations that is they are direct methods for

systems of linear equations. Among them are the scalar, vector and topological ε-algorithms. In particular the topological ε-algorithm gives the same sequence of iterates as the biconjugate gradient method. Instead of using this algorithm, the bordering method can be used and it also provides the same iterates. Since in this method it is known how to avoid breakdown or near-breakdown we have in hands a process for avoiding this drawback in the biconjugate gradient method. If our algorithms are applied to a system of nonlinear equations they provide derivative-free methods with quadratic convergence under the usual assumptions. These methods are related to projection methods and a wide literature recently appeared on the subject. We shall also see in this section how extrapolation can be used in regularization and penalty techniques and in continuation methods for nonlinear equations. The connection between sequence transformations and fixed point iterations will be emphasized.

6.2.1 Linear systems

When solving a system of linear equations by an iterative method we usually construct a sequence of vectors by

$$x_{n+1} = Ax_n + b\,, \qquad n = 0, 1, \ldots$$

where x_0 is a given arbitrary vector, A a square matrix and b a vector.

If the spectral radius of A is strictly less than one, then the matrix $I - A$ is regular and the sequence (x_n) converges to $x = (I - A)^{-1} b$ the unique solution of the system. We have

$$x_{n+1} - x = A(x_n - x) = A^{n+1}(x_0 - x).$$

Let P be the characteristic polynomial of A and let p be its degree (the dimension of the system). Then, by the Cayley-Hamilton theorem, $\forall n$

$$A^n\, P(A)(x_0 - x) = 0$$

that is, if we set $P(t) = a_0 + a_1 t + \cdots + a_p t^p$, $\forall n$

$$a_0(x_n - x) + a_1(x_{n+1} - x) + \cdots + a_p(x_{n+p} - x) = 0.$$

Moreover, since $I - A$ is regular, 1 is not a zero of P, that is

$$a_0 + \cdots + a_p \neq 0.$$

These results hold even if the spectral radius of A is not strictly less than 1 but under the only condition that $I - A$ be non-singular. Thus if the scalar, the vector or the topological ε-algorithm is applied to the sequence (x_n) then, by theorems 2.18, 4.2 or 4.3, we have $\forall n, \varepsilon_{2p}^{(n)} = x = (I - A)^{-1} b$ which shows that these three algorithms are direct methods for solving systems of linear equations. The above result can be refined and we obtain the more complete one due to Brezinski [48]

Theorem 6.17

Let us apply the (scalar, vector or topological) ε-algorithm to the sequence of vectors

$$x_{n+1} = Ax_n + b, \qquad n = 0, 1, \ldots$$

where x_0 is a given arbitrary vector.

If $I - A$ is regular let us denote by m the degree of the minimal polynomial of A for the vector $x_0 - x$ and by r the eventual multiplicity of its null zero. Then, $\forall n \geq 0$

$$\varepsilon_{2(m-r)}^{(n+r)} = x.$$

If $I - A$ is singular and if b belongs to the range of $I - A$ (which means that the system has infinitely many solutions) let us denote by m the degree of the minimal polynomial of A for the vector $x_0 - x$, by r the eventual multiplicity of its null zero and by q the multiplicity of its zero equal to 1. Then if $q = 1, \forall n \geq 0$

$$\varepsilon_{2(m-r)-2}^{(n+r)} = x$$

and if $q = 2, \forall n$

$$\varepsilon_{2(m-r)-3}^{(n+r)} = z$$

where z is a constant vector.

If $I - A$ is singular and if b does not belong to the range of $I - A$ (which means that the system has no solution) let us denote by m the degree of the minimal polynomial of A for the vector $x_1 - x_0$, by r the eventual multiplicity of its null zero and by q the multiplicity of its zero equal to 1. Then if $q = 1, \forall n \geq 0$

$$\varepsilon_{2(m-r)-1}^{(n+r)} = z$$

where z is a constant vector.

Let us recall that the minimal polynomial of A for an arbitrary vector u is the polynomial P of smallest degree such that $P(A)u = 0$. This polynomial is a divisor of the characteristic polynomial. This theorem shows that the ε-algorithms are direct methods for solving systems of linear equations even when the sequence (x_n) diverges and even if $I - A$ is singular and the system has infinitely many solutions. In that case however it is not known which solution is obtained nor is not known if the constant vector z have some connection with x. Similar results concerning the case where (x_n) is obtained by a k step iterative process were given by Brezinski [48].

Since the ε-algorithms are direct methods for linear systems it is interesting to study their possible links with other direct methods. It was proved by Wynn [478] that the vector ε-algorithm has no relation with the conjugate gradient method except in a very special case. As showed by Brezinski [63] such a connection exists for the topological ε-algorithm and we have the

Theorem 6.18

Let (y_k) be the sequence of vectors obtained by the conjugate or the biconjugate gradient method for solving $(I - A)x = b$ with $y_0 = 0$ and let $\left(\varepsilon_{2k}^{(0)}\right)$ be the vectors obtained by applying the topological ε-algorithm with $y = b$ to the sequence $x_{n+1} = Ax_n + b, n = 0, 1, \ldots$ with $x_0 = 0$ then

$$y_k = \varepsilon_{2k}^{(0)}, \qquad k = 0, 1, \ldots, p$$

where p is the dimension of the system. Moreover

$$y_k = -\frac{\begin{vmatrix} 0 & b & \cdots & B^{k-1}b \\ c_0 & c_1 & \cdots & c_k \\ \vdots & \vdots & & \vdots \\ c_{k-1} & c_k & \cdots & c_{2k-1} \end{vmatrix}}{\begin{vmatrix} c_1 & \cdots & c_k \\ \vdots & & \vdots \\ c_k & \cdots & c_{2k-1} \end{vmatrix}}$$

with $c_i = (b, B^i b)$ and $B = I - A$.

The conjugate gradient method is a well known method for symmetric positive definite systems. It can be found in many textbooks, see, for

example, Golub and van Loan [187]. The biconjugate gradient method is a generalization which applies to arbitrary matrices, see Fletcher [159].

A severe limitation in the use of the biconjugate gradient method is the possible occurrence of a breakdown (and in this case the algorithm stops) or of a near-breakdown (and in this case the propagation of rounding errors causes failure of the convergence). For example, Joubert [252] reported a breakdown in this algorithm for

$$B = I - A = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 1 & 3 \end{pmatrix}$$

and almost every starting vector y_0. If we use the ε-algorithms such a breakdown (or near-breakdown) is curable. In the scalar and the vector ε-algorithms we already saw that particular rules in which rounding errors propagate less exist and that these rules can be extended to the case when a division by zero occurs. The corresponding subroutines (see chapter 7) use these rules (which are only valid if no more than two adjacent vectors in the same column are equal or near-equal). Using these subroutines with the particular rules allows to solve the preceding system. For the topological ε-algorithm (and thus for the biconjugate gradient method) no such particular rules are known. However we saw that the sequence $\left(\varepsilon_{2k}^{(0)}\right)$ can be recursively computed by the bordering method (introduction of chapter 4) and that breakdown or near-breakdown can be avoided in the bordering method (section 1.8). Thus we now known particular rules for avoiding breakdown or near-breakdown in the biconjugate gradient method, see Brezinski, Redivo Zaglia and Sadok [98, 99].

When the matrix B is symmetric the topological ε-algorithm is uneconomical since it computes $(b, B^i b)$ for $i = 0, \ldots, 2p-1$ without using the property that $(B^j b, B^i b) = (b, B^{i+j} b)$. This drawback can be avoided by using the recursive projection algorithm or the compact recursive projection algorithm (section 4.4) as explained by Brezinski [68]. The solution of a system of linear equations can also be obtained via the H-algorithm (section 4.5). Since these methods are projection methods on a subspace, a more detailed study will be given in section 6.2.2.

To end this section let us mention that the ε-algorithms are very uneconomical for solving a linear system of dimension p since the com-

putation of $\varepsilon_{2p}^{(0)} = x$ needs the storage of $2p + 1$ vectors. However the numerical stability is sometimes better than in other direct methods. As we shall see in section 6.2.4, the main interest of these algorithms mostly lies in the possibility of obtaining quadratic derivative-free methods for systems of nonlinear equations. Such algorithms can be used for accelerating the convergence of matrix iterative processes by cycling with $k < p$ the dimension of the system. We set (see section 6.1.5)

$$\begin{aligned} u_0 &= x_n \\ u_1 &= Au_0 + b \\ &\vdots \quad\quad \vdots \\ u_{2k} &= Au_{2k-1} + b \end{aligned}$$

then the ε-algorithm is applied to the vectors $u_0, u_1, \ldots, u_{2k}$ and we take $x_{n+1} = \varepsilon_{2k}^{(0)}$. Methods of this type were studied by Jennings [246], Hafez and Cheng [210], Hafez et al. [211] and Lawther [276]. Realistic applications can be found in Espinoza [152] who obtained a gain of time of 37 per cent (with $k = 1$) for the Dirichlet problem for the Laplacian.

6.2.2 Projection methods

It is not our purpose in this book to develop projection methods. However since the ε-algorithm, the H-algorithm and some variants of the CRPA can be viewed as projection methods when applied to the solution of systems of linear and nonlinear equations we shall devote a section to this topics but we shall restrict ourselves to the material relevant to our subject.

Due to their use in the solution of systems of equations, projection methods have recently received much attention. They can be divided into two classes namely the ε-algorithms and the polynomial methods. Polynomial methods include the minimal polynomial extrapolation method (MPE) of Cabay and Jackson [109], Vorobyev [450] and Germain-Bonne [182], the reduced rank extrapolation method (RRE) due to Eddy [149], Mešina [319], and Kaniel and Stein [255] and the modified minimal polynomial extrapolation method (MMPE) of Brezinski [51], Pugachev [368], and Sidi, Ford and Smith [408].

The convergence and the stability properties of these methods were studied by Pugachev [368], Sidi [399], and Sidi, Ford and Smith [408].

The methods are showed to be bona fide acceleration methods when applied to vector sequences produced by a matrix iterative method whose matrix is diagonalizable. The case of defective iteration matrices was studied by Sidi and Bridger [407]. It was proved by Sidi [401] that the MPE, the RRE, the MMPE and the topological ε-algorithm, when applied to vectors generated by a matrix iterative method, are Krylov subspace methods and are thus equivalent to the conjugate gradient, the conjugate residual and the generalized conjugate gradient methods and that the error analysis of these methods can be unified. Another framework for their unified treatment was provided by Beuneu [28] and Jbilou [244]. Recursive algorithms for the implementation of the MPE, RRE and MMPE were given by Ford and Sidi [163], Sidi [406] and by Brezinski and Sadok [101] for the MMPE.

Let us now present these methods as in Jbilou and Sadok [245]. Let (S_n) be a sequence of vectors of $\mathbb{C}^p$. We shall denote by $\Delta^i_{n,k}$ the $p \times k$ matrix whose columns are $\Delta^i S_n, \ldots, \Delta^i S_{n+k-1}$ and by $Y_{n,k}$ the $p \times k$ matrix whose columns are the given vectors $y_1^{(n)}, \ldots, y_k^{(n)}$. $Y^*_{n,k}$ will be the transpose conjugate matrix of $Y_{n,k}$. We set

$$T_{n,k} = S_n - \Delta^1_{n,k} \left(Y^*_{n,k} \Delta^2_{n,k} \right)^{-1} Y^*_{n,k} \Delta S_n.$$

If $y_i^{(n)} = \Delta S_{n+i-1}$ we obtain the MPE. If $y_i^{(n)} = \Delta^2 S_{n+i-1}$ we obtain the RRE and for $y_i^{(n)} = y_i$ we have the MMPE. In the case where $k = p$ and y_i is the i-th vector of the canonical basis of $\mathbb{C}^p$ we recover Henrici's transformation (section 4.5). Using the Schur complement it can be proved that

$$T_{n,k} = \frac{\begin{vmatrix} S_n & \cdots & S_{n+k} \\ \left(y_1^{(n)}, \Delta S_n\right) & \cdots & \left(y_1^{(n)}, \Delta S_{n+k}\right) \\ \vdots & & \vdots \\ \left(y_k^{(n)}, \Delta S_n\right) & \cdots & \left(y_k^{(n)}, \Delta S_{n+k}\right) \end{vmatrix}}{\begin{vmatrix} 1 & \cdots & 1 \\ \left(y_1^{(n)}, \Delta S_n\right) & \cdots & \left(y_1^{(n)}, \Delta S_{n+k}\right) \\ \vdots & & \vdots \\ \left(y_k^{(n)}, \Delta S_n\right) & \cdots & \left(y_k^{(n)}, \Delta S_{n+k}\right) \end{vmatrix}}, \qquad k \leq p.$$

Of course the matrix $Y_{n,k}^* \Delta_{n,k}^2$ is assumed to be regular. Thus the vectors $T_{n,k}$ can be recursively computed by the H-algorithm with $g_i(n) = \left(y_i^{(n)}, \Delta S_{n+i-1}\right)$ or, when $y_i^{(n)} = y_i$ which corresponds to the MMPE, by the $S\beta$-algorithm (see section 4.2) or by the second variant of the CRPA (see section 4.4). The other methods can be implemented by the algorithms of Ford and Sidi [163] which are given in section 4.6. As showed by Lembarki [278] many other projection methods can be implemented by the RPA.

All these methods are direct methods for solving the system of linear equations $x = Ax + b$ when applied to the sequence (x_n) generated by the matrix iterative process

$$\begin{aligned} &x_0 \text{ given} \\ &x_{n+1} = Ax_n + b, \qquad n = 0, 1, \ldots \end{aligned}$$

As we shall see in section 6.2.4 they provide quadratic derivative-free methods for systems of nonlinear equations.

To end this section let us mention that the ε-algorithm can be extended for constructing intersection projection matrices by using generalized inverses instead of the inverse of a vector (which is in fact a generalized inverse) and that it has applications in the solution of systems of linear equations in the least squares sense, as studied by Pyle [370].

6.2.3 Regularization and penalty techniques

When solving a system of linear equations

$$Ax = b$$

the ill-conditioning of the matrix A can amplify errors on the data thus leading to a poor accuracy in the computed solution. A technique, called regularization and due to Tikhonov [434], enables to partly avoid this drawback. It consists in solving

$$(A + \varepsilon B)\, x(\varepsilon) = b$$

where B is a given matrix. If $A + \varepsilon B$ is better conditioned than B then the computed solution of this system can be less sensitive to perturbations on the data and a better accuracy can often be achieved.

We have

$$x(\varepsilon) = (I + \varepsilon M)^{-1} x$$

where $M = A^{-1}B$. Thus formally

$$x(\varepsilon) = \left(I - \varepsilon M + \varepsilon^2 M^2 - \varepsilon^3 M^3 + \cdots\right) x$$

the series converging if $|\varepsilon|\varrho(M) < 1$, and polynomial extrapolation can be used to obtain better estimates of x from $(x(\varepsilon_n))$ where (ε_n) is a sequence converging to zero. Such an extrapolation can be performed by using Richardson process on each component of the vectors $x(\varepsilon_n)$ or equivalently by the H-algorithm with $g_i(n) = \varepsilon_n^i$ and the theoretical results of section 2.2 apply.

Let us consider the Hilbert matrix given by $a_{ij} = 1/(i + j - 1)$ for $i, j, = 1, \ldots, p$ and $b_i = \sum_{j=1}^{p} a_{ij}$. The solution of the system $Ax = b$ is $x_i = 1$ for $i = 1, \ldots, p$. Taking $B = I$ and extrapolating we obtain for the maximum norm of the error and $p = 4$

$\varepsilon = 10^{-6}$	$0.178 \cdot 10^{-3}$			
		$0.190 \cdot 10^{-6}$		
$\varepsilon = 10^{-7}$	$0.180 \cdot 10^{-4}$		$0.192 \cdot 10^{-10}$	
		$0.191 \cdot 10^{-8}$		$0.474 \cdot 10^{-12}$
$\varepsilon = 10^{-8}$	$0.180 \cdot 10^{-5}$		$0.454 \cdot 10^{-12}$	
		$0.187 \cdot 10^{-10}$		
$\varepsilon = 10^{-9}$	$0.180 \cdot 10^{-6}$			

We see that, as stated by the theoretical results of theorem 2.15, the error behaves as $(0.1)^n$ in the first column and as $(0.1)^{2n}$ in the second column. After, the errors due to the conditioning and the computer's arithmetic become preponderant.

For $p = 10$ we have

$$\begin{array}{lllll}
\varepsilon = 10^{-1} & 0.330 & & & \\
 & & 0.102 & & \\
\varepsilon = 10^{-2} & 0.125 & & 0.324 \cdot 10^{-1} & \\
 & & 0.331 \cdot 10^{-1} & & 0.100 \cdot 10^{-1} \\
\varepsilon = 10^{-3} & 0.422 \cdot 10^{-1} & & 0.101 \cdot 10^{-1} & \\
 & & 0.103 \cdot 10^{-1} & & 0.289 \cdot 10^{-2} \\
\varepsilon = 10^{-4} & 0.135 \cdot 10^{-1} & & 0.290 \cdot 10^{-2} & \\
 & & 0.297 \cdot 10^{-2} & & \\
\varepsilon = 10^{-5} & 0.402 \cdot 10^{-2} & & &
\end{array}$$

Similar techniques can be applied to the solution of systems of linear equations in the least squares sense. If A is either singular or rectangular we have to solve

$$A^* A x = A^* b$$

where A^* is the adjoint of A (that is the conjugate transpose). Instead of solving this system, the regularized system

$$(A^* A x + \varepsilon B)\, x(\varepsilon) = A^* b$$

is solved for various values of ε and then extrapolated.

Let us consider the rectangular system given by Ribière [374]

$$\begin{pmatrix} 60 & 30 & 20 & 15 & 12 \\ 30 & 20 & 15 & 12 & 10 \\ 140 & 105 & 84 & 70 & 60 \\ 210 & 168 & 140 & 120 & 105 \\ 504 & 420 & 360 & 315 & 280 \\ 420 & 360 & 315 & 280 & 252 \end{pmatrix} \begin{pmatrix} 18 \\ 10 \\ 0 \\ -11 \\ -23 \end{pmatrix} = \begin{pmatrix} 939 \\ 378 \\ 1420 \\ 1725 \\ 3367 \\ 2284 \end{pmatrix}.$$

With $B = I$ we obtain

$$\begin{array}{lllll}
\varepsilon = 10^{-6} & 0.723 \cdot 10^{-1} & & & \\
 & & 0.825 \cdot 10^{-3} & & \\
\varepsilon = 10^{-7} & 0.797 \cdot 10^{-2} & & 0.504 \cdot 10^{-5} & \\
 & & 0.460 \cdot 10^{-4} & & 0.123 \cdot 10^{-5} \\
\varepsilon = 0.5 \cdot 10^{-7} & 0.401 \cdot 10^{-2} & & 0.126 \cdot 10^{-5} & \\
 & & 0.574 \cdot 10^{-5} & & 0.252 \cdot 10^{-6} \\
\varepsilon = 10^{-8} & 0.807 \cdot 10^{-3} & & 0.303 \cdot 10^{-6} & \\
 & & 0.847 \cdot 10^{-6} & & \\
\varepsilon = 0.5 \cdot 10^{-8} & 0.404 \cdot 10^{-3} & & &
\end{array}$$

As explained by Marchuk and Shaidurov [306, 307], $x(\varepsilon)$ is the solution of the minimization problem

$$\min_u \left(\|Au - b\|^2 + \varepsilon \|Bu\|^2 \right)$$

where $\|v\|^2 = (v, v)$. Thus the techniques of regularization and extrapolation previously described can be extended to the solution of the functional equation $Ax = b$ in an Hilbert space, in the least squares sense. The solution of

$$\min_u \|Au - b\|^2$$

is replaced by the solution of the regularized problem

$$\min_u \left(\|Au - b\|^2 + \varepsilon \|Bu\|^2 \right)$$

where B is some linear operator. Under some assumptions, whose details can be found in Morozov [327], this minimization problem is equivalent to solving

$$(A^*A + \varepsilon B^*B)\, x(\varepsilon) = A^*b.$$

This problem can be solved for various values of ε and then extrapolated as explained above. This technique has applications in differential and integral equations, such as the Laplace transform inversion (see section 6.4.4), in numerical integration and differentiation, in smoothing processes, in the solution of nonlinear equations by continuation methods. By changing ε into $1 - \varepsilon$, perturbation methods can also be put into this framework and extrapolated (see Bender and Orszag [25] for a description of perturbation methods) and Shanks' transformation, that is Padé approximation, can also be much useful. Several applications are given by Ribière [374].

From the practical point of view, the method of regularization presents two problems: the choice of B and that of ε. The ε_n's must be taken so that the conditions of theorem 2.15 are satisfied. Of course this case can be achieved by taking the ε_n's sufficiently small but, in that case, the regularized equation will be almost as ill-conditioned as the initial equation. If large values of the ε_n's are taken, the conditioning will be better but the $x(\varepsilon_n)$'s will be bad approximations of x and extrapolation will not improve their accuracy. Thus the choice of the ε_n's must take into account these two opposed facts, an almost heuristic procedure. An analysis can be found in Gastinel [176].

Let us now say a few words about the regularization of other operator equations. We consider the numerical solution of Fredholm integral equations of the first kind

$$\int_a^b K(x,y)\,f(y)\,dy = g(x).$$

K and g are known and the problem is to find f. This problem is usually ill-conditioned and a regularization technique was proposed by Bakushinskii [12]. It consists in solving

$$\varepsilon f(x) + \int_a^b K(x,y)\,f(y)\,dy = g(x).$$

If small values of ε are chosen then the problem is as ill-conditioned as the initial one. Thus one has to take larger values of ε. However, as mentioned above, the solution of the regularized problem will not be a good approximation of f in that case. Thus a possibility is to take several values of ε (not too small) and then to extrapolate, a technique proposed by Caldwell [110] and used to solve potential problems of the Dirichlet type.

Regularization techniques different from that of Tikhonov can be used. They are described by Engl [151] and can be extended to nonlinear equations.

Let us now consider the nonlinear constrained optimization problem

$$\text{Minimize } f(x) \text{ subject to } g_i(x) \geq 0 \text{ for } i = 1,\ldots,m$$

where $x \in \mathbb{R}^n, f : \mathbb{R}^n \longmapsto \mathbb{R}, g_i : \mathbb{R}^n \longmapsto \mathbb{R}$. The inequality constraints can be replaced by equality constraints $g_i(x) = 0$ for $i = 1,\ldots,m$.

If the constraints cannot be eliminated it is necessary to make a compromise between the minimization of the objective function f and the feasible region for x. This conflict leads to penalty function methods whose simplest one is

$$\text{Minimize } \phi(x,\varepsilon) = f(x) + \frac{1}{2\varepsilon}\sum_{i=1}^{m} g_i^2(x).$$

Let $x(\varepsilon)$ be the solution of this unconstrained minimization problem. Under some assumptions $\lim_{n\to\infty} x(\varepsilon) = x^*$ the solution of the initial constrained optimization problem, see for example Ciarlet [115].

The penalized problem can be solved for various values of ε and the solutions can be extrapolated by using Richardson's process.

For example let us take the problem considered by Fletcher [160]

$$\text{Minimize } -x_1 - x_2 \text{ subject to } 1 - x_1^2 - x_2^2 = 0$$

whose solution is $x_1^* = x_2^* = 1/\sqrt{2} = 0.70710678\ldots$.

We obtain the following results with extrapolation

ε	$x_1(\varepsilon) = x_2(\varepsilon)$	$k=1$	$k=2$	$k=3$
1	0.8846462			
10^{-1}	0.7308931	0.7138094		
10^{-2}	0.7095936	0.7072269	0.7071605	
10^{-3}	0.7073566	0.7071080	0.7071068	0.7071068
10^{-4}	0.7071318	0.7071068	0.7071068	0.7071068
10^{-5}	0.7071093	0.7071068	0.7071068	0.7071068

They show the improvement obtained by extrapolation. On penalty function methods see Minoux [322] for their theory and Haslinger and Neittaanmäki [214] for their application in optimal shape design.

As for regularization the choice of the ε's is the result of a balance. If ε is chosen small, in order that $x(\varepsilon)$ be close to x^*, then the problem could be ill-conditioned thus leading to numerical difficulties in the solution of the unconstrained problem. If ε is too large then $x(\varepsilon)$ will be a bad approximation of x^* and no gain could be brought by extrapolation. Heuristic techniques for the choice of the ε' s are given by Minoux [322] among others.

The generalized reduced gradient (GRG) is a method due to Abadie [1] for solving nonlinear constrained optimization problems. For finding the direction of displacement at each step use is made of the conjugate gradient procedure of Fletcher and Reeves [161] which is an iterative method whose convergence is often quite slow. Its convergence has been accelerated by Aboun [3] using the vector ε-algorithm and by Rahmani [372] by the topological ε-algorithm. The numerical tests performed on eight problems show that a gain in the number of iterations between 5.4% and 60.8% can be brought by the ε-algorithms. For some problems the computing time is divided by 2.8. The parallelization of the procedure is also possible.

6.2.4 Nonlinear equations

There is a very strong connection between sequence transformations and fixed point methods for solving $x = g(x)$ where $g : \mathbb{R}^p \longmapsto \mathbb{R}^p$. The most well known example of this connection is that between Aitken's Δ^2 process and Steffensen's method in the case $p = 1$. We have

$$T_n = S_n - \frac{(S_{n+1} - S_n)^2}{S_{n+2} - 2S_{n+1} + S_n}, \qquad n = 0, 1, \ldots \text{ for Aitken's process}$$

and

$$x_{n+1} = x_n - \frac{(g(x_n) - x_n)^2}{g(g(x_n)) - 2g(x_n) + x_n}, \quad n = 0, 1, \ldots \text{ for Steffensen's method.}$$

More generally let $T : (S_n) \longmapsto (T_n)$ be a quasi-linear sequence transformation defined by

$$T_n = F(S_n, \ldots, S_{n+k}), \qquad n = 0, 1, \ldots$$

For solving the fixed point problem $x = g(x)$ we can associate to it the iterative method

$$x_{n+1} = F(x_n, g(x_n), \ldots, g_k(x_n)), \qquad n = 0, 1, \ldots$$

where $g_{i+1}(t) = g(g_i(t))$ and $g_0(t) = t$. Conversely to any fixed point iteration of this form, we can associate a sequence transformation of the previous form.

In the case $p = 1$ the properties of the sequence transformation and the fixed point method can be related. Assuming that F is quasi-linear and that g is differentiable in the neighborhood of the fixed point x we have

$$x_{n+1} - x = (x_n - x) \cdot F\left(1, \frac{g_1(x_n) - x}{x_n - x}, \ldots, \frac{g_k(x_n) - x}{x_n - x}\right)$$

and

$$g_i(x_n) - x = [g'(x)]^i (x_n - x) + o(x_n - x).$$

Thus gathering the results of Wimp [467] (p. 113) and Brezinski [81] we have the

Theorem 6.19

Let (S_n) be a sequence converging to S and such that $\exists a \neq 1$, $\lim_{n\to\infty} (S_{n+1} - S)/(S_n - S) = a$. Let x be such that $x = g(x)$ and $g'(x) \neq 1$. Then a necessary and sufficient condition that (T_n) converges to S faster than (S_n) for all (S_n) satisfying the previous assumptions, is that $\lim_{n\to\infty} (x_{n+1} - x)/(x_n - x) = 0$ for all g satisfying the previous assumptions.

Thus the acceleration of linear sequences and the super-linear convergence of the fixed point method are closely related.

We also recall that a necessary and sufficient condition for a sequence transformation to accelerate linear sequences is to be exact for geometric progressions (a consequence of theorem 1.8) and that a necessary and sufficient condition for fixed point methods to converge super-linearly is to be exact for $g(x) = ax + b$ (which means $x_1 = x$, $\forall x_0$). The conditions of the above theorem are satisfied for Aitken's process and it is known that Steffensen's method has order 2 if $g'(x) \neq 1$.

As explained above the use of extrapolation methods for constructing fixed point iterations consists in a procedure called *cycling* since the process can be written, as

$$\begin{aligned} u_0 &= x_n \\ u_1 &= g(u_0) \\ &\vdots \quad\quad \vdots \\ u_k &= g(u_{k-1}) \\ x_{n+1} &= F(u_0, u_1, \ldots, u_k). \end{aligned}$$

Overholt's process (section 2.8) was built for that purpose. In that case if g is sufficiently differentiable in a neighborhood of x, if $g'(x) \neq 1$ and if $x_{n+1} = V_{k-1}^{(0)}$ then the sequence (x_n) thus obtained has order k. For $k = 2$ this procedure is identical to Steffensen's method.

There are various other ways of using extrapolation methods for improving fixed point iterations. They can be found in Brezinski [44], Nourein [338], Claessens, Loizou and Wuytack [116], Brezinski [56], Cuyt and Wuytack [131].

Cycling with the ε-algorithm and $k = 1$ is exactly Steffensen's method. Cycling with a higher value of k was studied by Achakir [5] and Ortloff [344] (see also Ortloff and Böhm [345]). The complex dynamics of such a procedure was studied by Iserles [238]. It has order $k + 1$.

Steffensen's method has order 2 and each iteration needs two function evaluations. Thus its efficiency index is $2^{1/2} = 1.414\ldots$

A variant was proposed by King [262]. It is as follows

$$\begin{aligned} & x_1 = g(x_0), \quad x_2 = g(x_1) \\ & y_0 = x_0, \quad y_1 = x_1 \\ x_{n+2} &= g(y_{n+1}) \\ y_{n+2} &= y_{n+1} - \frac{y_{n+1} - x_{n+2}}{1 - K_{n+1}} \end{aligned}$$

with $K_{n+1} = (x_{n+2} - x_{n+1})/(y_{n+1} - y_n)$.

This process has order $\left(1 + \sqrt{5}\right)/2 = 1.618\ldots$ but it requires only one function evaluation per iteration. Thus its efficiency index is $1.618\ldots$ which is better than Steffensen's method and identical to that of Regula Falsi (secant method).

If the basic iterations $u_{i+1} = g(u_i)$ have an order k greater than one, then Aitken's process has to be replaced by

$$T_n = S_n - \frac{S_{n+1} - S_n}{d - 1}$$

where d is a particular zero of

$$t^k + t^{k-1} + \cdots + t - a = 0$$

and $a = \Delta S_{n+1}/\Delta S_n$. Then, as proved by Jones [247], the sequence (x_n) will have order k^2 (instead of $2k - 1$ as showed by Ostrowski [350]).

If a is positive then the polynomial $t^k + t^{k-1} + \cdots + t - a$ has only one real positive zero and d must be taken as this zero. Moreover a is positive if the sequence (u_i) is monotone. If (u_i) is not monotone it is alternate and then we can replace it by (u_{2i+2}) and k by k^2.

For sequences of order strictly greater than one, the procedures for controlling the error presented in section 3.8 are particularly effective. If we set $T_n = x_{n+1}$ then we can write $T_n = x_n + \Delta x_n$ and $T_n(b) = x_n + (1 - b)\Delta x_n$. (T_n) converges to x faster than (x_n) since its order is greater than 1. Thus, by theorem 3.40, $\exists N$ such that $\forall n \geq N$, x belongs to the intervals whose endpoints are $T_n(-b)$ and $T_n(b)$. Moreover the sequences $(T_n(\pm b))$ have the same order than (x_n).

Let us now come to the case where $g'(x) = 1$ always for one equation in one unknown. This is the case when $f(x) = x - g(x)$ has a multiple zero. We shall present the results obtained by Sablonnière [379].

We consider the sequence $x_{n+1} = g(x_n)$ with

$$g \in A_q = \left\{ g(x) = x + \sum_{i=1}^{\infty} a_{q+i} x^{q+i} \text{ with } a_{q+1} < 0 \right\}.$$

In that case $x_n - x = O\left(n^{-1/q}\right)$. For some subsets of A_1 and A_2 it is possible to obtain more terms in the asymptotic expansion of $x_n - x$ which allows to construct suitable extrapolation processes.

For each $g \in A_q$ there exists a unique function ψ

$$\psi(x) = \sum_{i=1}^{q} d_i x^{-i} + c_0 \ln x + \sum_{i=1}^{\infty} c_i x^i$$

satisfying the relation

$$\psi(g(x)) = 1 + \psi(x).$$

Thus

$$\psi(x_n) = \psi(g_n(x_0)) = n + \psi(x_0).$$

The coefficients of ψ can be obtained from those of g and then the asymptotic expansion of $x_n - x$ is deduced from the last relation.

Let us consider the following subsets

$$A_1' = \{g \in A_1, c_0 \neq 0\}, \qquad A_1'' = \{g \in A_1, c_0 = 0\},$$
$$A_2' = \{g \in A_2, a_4 \neq 0, c_0 \neq 0\}, \qquad A_2'' = \{g \in A_2, g \text{ odd}, c_0 \neq 0\},$$
$$A_2''' = \{g \in A_2, g \text{ odd}, c_0 = 0\}.$$

Let us set $D(u_n) = \Delta u_n \Delta u_{n+1} / \Delta^2 u_n$. Thus Aitken's Δ^2 process can be written as

$$T_n = u_{n+1} - D(u_n).$$

We consider the algorithms

$$A_1 : \quad x_n^{(0)} = x_n, \quad x_n^{(k)} = x_{n+1}^{(k-1)} - \left(\frac{k+q}{k}\right) D\left(x_n^{(k-1)}\right)$$

$$A_2 : \quad x_n^{(0)} = x_n, \quad x_n^{(k)} = x_{n+1}^{(k-1)} - \left(\frac{2k+q-1}{2k-1}\right) D\left(x_n^{(k-1)}\right)$$

$$A_3 : \quad x_n^{(0)} = x_n, \quad x_n^{(k)} = x_{n+1}^{(k-1)} - \left(\frac{4k-1}{4k-3}\right) D\left(x_n^{(k-1)}\right).$$

We have the

Theorem 6.20

For $k \geq 1$ and when n tends to infinity

i) if $g \in A'_q$ for $q = 1$ or 2 and if the algorithm A_1 is used then

$$x_n^{(k)} - x = O\left(n^{-(k+1)/q}\right)$$

ii) if $g \in A''_q$ for $q = 1$ or 2 and if the algorithm A_2 is used then

$$x_n^{(k)} - x = O\left(n^{-(2k+1)/q}\right)$$

iii) if $g \in A'''_2$ and if the algorithm A_3 is used then

$$x_n^{(k)} - x = O\left(n^{-(2k+1)/2}\right).$$

The same results hold if, instead of the previous algorithms A_1, A_2 and A_3 the k-th iterate of the Θ_2-algorithm is used. Both procedures can also be combined to obtain a slightly better rate of convergence. Let us denote by $\Theta_2(u_n)$ the application of the Θ_2-algorithm (that is the second column of the Θ-algorithm) to the sequence (u_n). We consider the following algorithms

$$A_4 \quad : \quad x_n^{(0)} = x_n, \quad y_n^{(k)} = x_{n+1}^{(k-1)} - \left(\frac{2k+q-1}{2k-1}\right) D\left(x_n^{(k-1)}\right),$$
$$z_n^{(k)} = \Theta_2\left(x_n^{(k-1)}\right), \quad x_n^{(k)} = \left(y_n^{(k)} + (2k-1)z_n^{(k)}\right) \Big/ (2k).$$

$$A_5 \quad : \quad x_n^{(0)} = x_n, \quad y_n^{(k)} = x_{n+1}^{(k-1)} - \left(\frac{4k-1}{4k-3}\right) D\left(x_n^{(k-1)}\right),$$
$$z_n^{(k)} = \Theta_2\left(x_n^{(k-1)}\right), \quad x_n^{(k)} = \left(2y_n^{(k)} + (4k-3)z_n^{(k)}\right) \Big/ (4k-1).$$

We have the

Theorem 6.21

For $k \geq 1$ and when n tends to infinity

i) if $g \in A'_q$ for $q = 1$ or 2 and if the algorithm A_4 is used then

$$x_n^{(k)} - x = O\left(n^{-(2k+1)/q}\right)$$

ii) if $g \in A_2''$ *and if the algorithm* A_5 *is used then*

$$x_n^{(k)} - x = O\left(n^{-(2k+1)/2}\right).$$

Thus a better convergence rate is obtained for A_1'. These results were extended by Sedogbo [389] to a modification of the ε-algorithm. He obtained a better result for A_2''' by considering the modified ε-algorithm

$$\begin{aligned} \varepsilon_{-1}^{(n)} &= 0\,, \quad \varepsilon_0^{(n)} = x_n\,, \quad n = 0, 1, \ldots \\ \varepsilon_{k+1}^{(n)} &= \varepsilon_{k-1}^{(n+1)} + \frac{A_k}{\varepsilon_k^{(n+1)} - \varepsilon_k^{(n)}}\,, \quad k, n = 0, 1, \ldots \end{aligned}$$

with $A_{2k} = 1/(4k+1)$, $A_{2k+1} = 4k+3$ and we have the

Theorem 6.22

If $g \in A_2'''$ *then when* n *tends to infinity*

$$\varepsilon_{2k}^{(n)} - x = O\left(n^{-(4k+1)/2}\right).$$

Let us now consider the following algorithm

$$\begin{aligned} A_6 \;:\; & x_n^{(0)} = x_n, \quad y_n^{(k)} = x_{n+1}^{(k-1)} - c_k D\left(x_n^{(k-1)}\right), \\ & z_n^{(k)} = \Theta_2\left(x_n^{(k-1)}\right), \quad x_n^{(k)} = \gamma_k y_n^{(k)} + \lambda_k z_n^{(k)}. \end{aligned}$$

We have the

Theorem 6.23

For $k \geq 1$ *and when* n *tends to infinity*

i) if $g \in A_q'$ *for* $q = 1$ *or 2 and if the algorithm* A_6 *is applied with* $c_k = (2k+q-1)/(2k-1)$, $\gamma_k = 1/2k$ *and* $\lambda_k = (2k-1)/2k$ *then*

$$x_n^{(k)} - x = O\left(n^{-(2k+1)/q}\right)$$

ii) if $g \in A_1''$ *and if the algorithm* A_6 *is applied with* $c_k = (3k-1)/(3k-2)$, $\gamma_k = 2/3k$ *and* $\lambda_k = (3k-2)/3k$ *then*

$$x_n^{(k)} - x = O\left(n^{-(3k+1)}\right)$$

iii) if $g \in A_2''$ *and if the algorithm* A_6 *is applied with* $c_k = (4k-1)/(4k-3)$, $\gamma_k = 2/(4k-1)$ *and* $\lambda_k = (4k-3)/(4k-1)$ *then*

$$x_n^{(k)} - x = O\left(n^{-(2k+1)/2}\right)$$

iv) if $g \in A_2'''$ *and if the algorithm* A_6 *is applied with* $c_k = (5k-1)/(5k-3)$, $\gamma_k = 4/(5k+1)$ *and* $\lambda_k = (5k-3)/(5k+1)$ *then*

$$x_n^{(k)} - x = O\left(n^{-(5k+2)/2}\right).$$

Let us consider the case $g(x) = x + \cos x - 1$, $g \in A_1'$. We obtain

n	$k=0$	$k=1$	$k=2$	$k=3$
0	0.300	$-0.171 \cdot 10^{-2}$	$-0.182 \cdot 10^{-5}$	$-0.249 \cdot 10^{-8}$
1	0.255	$-0.103 \cdot 10^{-2}$	$-0.945 \cdot 10^{-6}$	$-0.908 \cdot 10^{-9}$
2	0.223	$-0.671 \cdot 10^{-3}$	$-0.538 \cdot 10^{-6}$	$-0.333 \cdot 10^{-9}$
3	0.198	$-0.464 \cdot 10^{-3}$	$-0.328 \cdot 10^{-6}$	$-0.112 \cdot 10^{-9}$
4	0.179	$-0.336 \cdot 10^{-3}$	$-0.211 \cdot 10^{-6}$	$-0.334 \cdot 10^{-10}$
5	0.163	$-0.251 \cdot 10^{-3}$	$-0.141 \cdot 10^{-6}$	$-0.316 \cdot 10^{-11}$
6	0.149	$-0.193 \cdot 10^{-3}$	$-0.976 \cdot 10^{-7}$	
7	0.138	$-0.152 \cdot 10^{-3}$	$-0.696 \cdot 10^{-7}$	
8	0.129	$-0.122 \cdot 10^{-3}$	$-0.508 \cdot 10^{-7}$	

For the class A_1' still more interesting algorithms and precise results were given by Sablonnière [380].

He considered the four algorithms ($n \geq 1$)

i)

$$\varepsilon_{-1}^{(n)} = 0, \quad \varepsilon_0^{(n)} = x_n$$

$$\varepsilon_{2k-1}^{(n)} = \varepsilon_{2k-3}^{(n+1)} + k \Big/ \left(\varepsilon_{2k-2}^{(n+1)} - \varepsilon_{2k-2}^{(n)}\right)$$

$$\varepsilon_{2k}^{(n)} = \varepsilon_{2k-2}^{(n+1)} + (k+1) \Big/ \left(\varepsilon_{2k-1}^{(n+1)} - \varepsilon_{2k-1}^{(n)}\right)$$

ii)

$$d_0^{(n)} = x_n, \quad d_k^{(n)} = \delta_k\left(d_{k-1}^{(n)}\right) \text{ where}$$

$$\delta_\alpha(u_n) = u_{n+1} - \frac{\alpha+1}{\alpha} \cdot \frac{\Delta u_n \cdot \Delta u_{n+1}}{\Delta^2 u_n}$$

iii)

$$t_0^{(n)} = x_n, \quad t_k^{(n)} = \hat{\theta}\left(t_{k-1}^{(n)}\right) \text{ where}$$

$$\hat{\theta}(u_n) = u_{n+1} - \frac{\Delta u_{n+1}\left(u_{n+1} - \delta(u_n)\right)}{\Delta u_{n+1} - \Delta\delta(u_n)} \text{ and}$$

$$\delta(u_n) = u_{n+1} - \Delta u_n \Delta u_{n+1} \Big/ \Delta^2 u_n$$

iv) the Θ-algorithm.

For these algorithms we have the following results, assuming that (x_n) tends to zero (which does not restrict the generality)

Theorem 6.24

If $g \in A_1'$ then

i)

$$\lim_{n\to\infty}\left(n\,\varepsilon_{2k}^{(n)} \Big/ \varepsilon_{2k-2}^{(n)}\right) = k\,c_0/(k+1)$$
$$\lim_{k\to\infty} \varepsilon_{2k}^{(n)} \Big/ \varepsilon_{2k-2}^{(n)} = c_0/n$$

ii)

$$\lim_{n\to\infty}\left(n\,d_k^{(n)} \Big/ d_{k-1}^{(n)}\right) = c_0/(k+1)$$
$$\lim_{k\to\infty}\left(k\,d_{k+1}^{(n)} \Big/ d_k^{(n)}\right) = c_0/n$$

iii)

$$\lim_{n\to\infty}\left(n\,t_k^{(n)} \Big/ t_{k-1}^{(n)}\right) = -c_0/k(k+1)$$
$$\lim_{k\to\infty}\left(k(k+1)\,t_k^{(n)} \Big/ t_{k-1}^{(n)}\right) = -c_0/n$$

iv)

$$\lim_{n\to\infty}\left(n\Theta_{2k}^{(n)} \Big/ \Theta_{2k-2}^{(n)}\right) = -c_0\omega_{k+1}/\omega_k \text{ where}$$
$$\omega_1 = 1,\ \omega_2 = 0.5 \text{ and } \omega_{k+1} = \frac{\omega_k}{k+1}\left(\frac{1}{k} - \frac{k}{k-1}\cdot\frac{\omega_k}{\omega_{k-1}}\right)$$
$$\lim_{k\to\infty}\left(k(k+1)\Theta_{2k}^{(n)} \Big/ \Theta_{2k-2}^{(n)}\right) = -c_0/n.$$

Let us notice that the sequence (ω_k) tends to zero very rapidly. More precisely $k!\,(k-1)!\omega_k = O(1/k)$ and $\lim_{k\to\infty} (k-1)!\omega_k = 0$.

If these four algorithms are compared together then we obtain the

Theorem 6.25

If $g \in A_1'$ *then*

$$\lim_{n\to\infty} d_k^{(n)} \Big/ \varepsilon_{2k}^{(n)} = 1/k!$$
$$\lim_{n\to\infty} t_k^{(n)} \Big/ d_k^{(n)} = (-1)^k/k!$$
$$\lim_{n\to\infty} \Theta_{2k}^{(n)} \Big/ t_k^{(n)} = k!/(k+1)!\omega_{k+1} = O(1/k).$$

Thus the best algorithm among the four is the Θ-algorithm.

When $f(x) = x - g(x)$ has a zero of known multiplicity m, this multiplicity can be introduced in the algorithm to have quadratic convergence as for a simple zero. For example, Steffensen's method becomes, see Brezinski [36]

$$u_0 = x_n$$
$$u_1 = g(u_0)$$
$$u_2 = g(u_1)$$
$$x_{n+1} = u_0 - mb_0 \quad \text{with} \quad b_0 = (\Delta u_0)^2 \Big/ \Delta^2 u_0.$$

When the multiplicity m is unknown, Achakir [5] proposed a method with quadratic convergence in which a sequence (m_n) converging to m is built such that $m_{n+1} = m + O(x_n - x)$. It is as follows

$$u_0 = x_n$$
$$u_1 = g(u_0)$$
$$u_2 = g(u_1)$$
$$u_3 = g(u_2)$$
$$x_{n+1} = u_0 - \frac{\Delta u_0}{\Delta b_0} \cdot b_0 \quad \text{with} \quad b_i = (\Delta u_i)^2 \Big/ \Delta^2 u_i \text{ and } m_{n+1} = \frac{\Delta u_0}{\Delta b_0}.$$

Let us consider, for example, the case

$$x = \sin x$$

which has a zero of multiplicity 3 at zero. We obtain with $x_0 = 0.5$

n	Modified Steffensen's method	Achakir's method	m_n
1	$-0.25 \cdot 10^{-1}$	$0.41 \cdot 10^{-1}$	2.62
2	$-0.14 \cdot 10^{-4}$	$0.28 \cdot 10^{-4}$	2.9969
3	$-0.45 \cdot 10^{-14}$	$0.87 \cdot 10^{-10}$	2.9999908

Then a division by zero occurs. These results were obtained on a VAX computer with 33 decimal digits in extended precision.

Let us consider again iterations of the form

$$x_{n+1} = F(x_n), \qquad n = 0, 1, \ldots$$

We have

$$x_{n+1} = x + (x_n - x)F'_n$$

with $F'_n = F'(x + \theta_n(x_n - x))$ and $\theta_n \in [0, 1]$. Subtracting x_n to both sides we obtain

$$x_n = \frac{x - \Delta x_n - x\, F'_n}{1 - F'_n}.$$

If $|F'_n|$ is small compared to $|\Delta x_n|$ (which is true if $F''(x) = 0$) then

$$x_n \simeq \frac{x - \Delta x_n}{1 - F'_n}$$

which shows that the sequence (x_n) can be extrapolated with the rational form of the E-algorithm that is with $g_1(n) = \Delta x_n$ and $g_2(n) = x_n\, F'_n$. Since F'_n is unknown we shall approximate it by

$$f'_n = \frac{F(S_n) - F(S_{n-1})}{S_n - S_{n-1}} = \frac{\Delta S_n}{\Delta S_{n-1}}.$$

Let us consider again the case $F(x) = \sin x$. We obtain the following numbers of exact digits

n	x_n	$E_1^{(n)}$	$E_2^{(n)}$
0	0.07		
1	0.13	0.22	
2	0.17	0.28	0.67
3	0.20	0.33	0.97
4	0.23	0.37	1.19
5	0.26	0.40	1.37
6	0.28	0.42	1.51
7	0.30	0.45	1.64
8	0.32	0.47	1.75
9	0.33	0.49	1.85
10	0.35	0.50	1.94
12	0.38	0.53	2.10
14	0.40	0.56	2.24
19	0.45	0.62	2.52

$E_1^{(n)}$ corresponds to Aitken's Δ^2 process. We tried other algorithms on this example and the best one seems to be the Θ-algorithm.

We obtain for the last value computed by the algorithms

	$n=4$	$n=7$	$n=12$	$n=15$	$n=16$
ε-algorithm	0.42	0.55	0.72	0.79	0.82
ϱ-algorithm	0.78	1.01	1.32	1.43	1.54
Θ-algorithm	0.85	1.76	4.28	4.49	6.20
Richardson	0.46	0.61	0.76	0.82	0.84
1^{st} gener. ε	0.02	0.03	0.00	0.01	0.00
2^{nd} gener. ε	0.09	0.10	0.09	0.09	0.09
Iterated Δ^2	0.49	0.72	1.24	1.34	1.28
Overholt	0.48	0.64	0.80	0.87	0.89
Levin t	0.40	0.64	0.80	0.88	0.90

Then the precision of the Θ-algorithm deteriorates due to cancellations errors.

If the condition $F''(x) = 0$ is not satisfied (as for $F(x) = e^{-x}$) then the preceding method does not work well. For this example we obtain

n	x_n	$E_1^{(n)}$	$E_2^{(n)}$
0	0.45		
1	0.66	1.58	
2	0.93	2.09	0.17
5	1.66	3.57	0.18
10	2.89	6.03	0.18
15	4.12	8.50	0.18

In both cases the computation of $E_3^{(n)}$ with $g_3(n) = f_n'$ does not improve the results and it can even deteriorate them.

We shall now consider the case where $p > 1$ that is systems of nonlinear equations written as $x = g(x)$. For such systems the following cycling procedure is proposed

- choose x_0
- $(n+1)$-th iteration

$$u_0 = x_n$$
$$u_{i+1} = g(u_i), \qquad i = 0, \ldots, d_n$$

  where d_n is the degree of the minimal polynomial of the matrix $g'(x)$ for the vector $x_n - x$ (see section 6.2.1 for the definition).

- Then set

$$x_{n+1} = T_{0,d_n}$$

  where $T_{n,k}$ denotes one of the projection methods studied in section 6.2.2 that is the MPE and the RRE.

Such procedures for solving a system of nonlinear equations have quite a long history. They were first proposed by Brezinski [33, 41] and Gekeler [177] who both used the scalar or the vector ε-algorithm. They studied the convergence of the sequence (x_n), which is obviously quadratic for most of the examples, but there was a gap in their proofs. The same gap can be found in the works of Skelboe [410] for the MPE and Beuneu [28] for another class of extrapolation methods. A full satisfactory proof of the quadratic convergence of (x_n) was given for the first time by Jbilou and Sadok [245] who proved the

Theorem 6.26

If $I - g'(x)$ is regular, if $\|g'(u) - g'(v)\| \leq L\|u - v\|$ for all $u, v \in D \subseteq \mathbb{R}^p$, if $\exists \alpha > 0$ such that $\forall u \in D_0 - \{x\}$, $\alpha_n(u) > \alpha$ where $D_0 \subseteq D$ and $\alpha_n(u) = \sqrt{\det\left(H_n^*(u)H_n(u)\right)}$, $H_n(u)$ being the matrix whose columns are

$$\frac{g^i(u) - g^{i-1}(u)}{\|g^i(u) - g^{i-1}(u)\|} \quad \text{for } i = 1, \ldots, d_n$$

with $g^0(u) = u$ and $g^i(u) = g\left(g^{i-1}(u)\right)$ and d_n the degree of the minimal polynomial of $g'(x)$ for the vector $x_n - x$, then there exists $U \subseteq D_0$ such that $\forall x_0 \in U$, the sequence (x_n) constructed above converges quadratically to x that is

$$\|x_{n+1} - x\| = O\left(\|x_n - x\|^2\right).$$

This theorem is also true when $T_{0,d_n} = \varepsilon_{2d_n}^{(0)}$ obtained by the topological ε-algorithm (see Le Ferrand [277]). It seems to be true for the vector ε-algorithm although an entirely satisfactory proof has not yet been obtained. The case of Henrici's method was studied by Ortega and Rheinboldt [343], section 11.3.5, page 373. This theorem is still true for a wider class of methods under a supplementary assumption.

If the matrix $I - g'(x)$ is singular, a regularization technique due to Yamamoto [486] can be used.

It must be noticed that we have not assumed that g is a contraction or, in other words, we have not assumed that the sequence (u_i), computed by the basic iterations $u_{i+1} = g(u_i)$, converges. If (u_i) diverges rapidly then numerical instability can occur and this instability can cause the loss of the quadratic character of the convergence or even of the convergence itself. In that case the basic iterations have to be replaced by

$$u_{i+1} = u_i + \alpha_n \left(g(u_i) - u_i\right), \qquad i = 0, \ldots, d_n \ (\text{or } 2d_n).$$

Let $\lambda_1, \ldots, \lambda_p$ be the eigenvalues of $g'(x)$ and let C be the open disc of the complex plane with center at -1 and radius 1. Then, as proved by Brezinski [53], these new basic iterations are numerically stable if α_n is chosen such that

$$\alpha_n(\lambda_i - 1) \in C \quad \text{for} \quad i = 1, \ldots, p.$$

Such an α_n exists only if all the $(\lambda_i - 1)$'s have their real parts of the same sign. If it is not the case it is possible to minimize the numerical instability by taking α_n such that

$$\alpha_n(\lambda - 1) \in C \quad \text{with} \quad |\lambda - 1| = \max_i |\lambda_i - 1|.$$

Of course such a procedure is difficult to implement since the λ_i's are, in general, unknown. To avoid this drawback Wynn [485] proposed to choose α_n as follows after setting $\alpha_0 = 1$. At the beginning of the $(n+1)$-th iteration we set

$$\begin{aligned} v_0 &= x_n \\ v_1 &= v_0 + \alpha_{n-1}\,(g(v_0) - v_0) \end{aligned}$$

and then α_n is computed by

$$\alpha_n = -\frac{\|v_1 - v_0\|}{\|g(v_1) - v_1 - g(v_0) + v_0\|}$$

where the norm of the vector y is taken as

$$\|y\| = \max_{1 \le i \le p} |y_i|$$

y_i being the i-th component of y. After having determined α_n by this method the modified basic iteration are computed again from $u_0 = x_n$. Many numerical results with the ε-algorithm are given by Brezinski [56].

Let us give one more example. We consider the system

$$\begin{cases} x = axy^3 + b \\ y = a'e^{1+xy} + b' \end{cases}$$

with $a - b = a' + b' = 1$. Its solution is $x = -1$ and $y = 1$. The eigenvalues of the Jacobian matrix at the solution are

$$\lambda = \frac{a - a' \pm \sqrt{(a' - a)^2 - 8aa'}}{2}.$$

If $a = a' = 1/2$ than $\lambda_i - 1 \in C$ for $i = 1$ and 2. Thus we can take $\alpha_n = 1$ in the basic iterations which are in this case $u_{i+1} = g(u_i)$. We obtain

n	x_n	y_n
0	0.0	0.0
1	−1.32	0.917
2	−1.011	1.0014
3	−1.000017	0.9999984
4	−1.000000000035	1.0000000000020
5	−0.9999999999999999	1.000000000000000

Let us now take $a = 1$ and $a' = 10$. We have $\lambda_1 - 1 = -5$ and $\lambda_2 - 1 = -6$ and the stability condition is satisfied by the choice $\alpha_n = 0.25$ since we must have $0 < \alpha_n < 1/3$. We obtain

n	x_n	y_n
0	−0.9	0.9
1	−1.073	1.012
2	−1.041	0.984
3	−1.0050	0.9968
4	−1.000041	0.999968
5	−1.0000000020	0.9999999985
6	−0.9999999999999999	1.000000000000000

If we take $\alpha_n = 0.5$ then the stability condition is not satisfied and we have

n	x_n	y_n
0	−0.9	0.9
3	−1.060	0.953
6	−1.043	1.015
10	−1.003	1.003

For higher values of α_n the program stops by an overflow.

If the above automatic determination of α_n is used then we obtain for $a = 1$, $a' = 10$, $x_0 = -0.9$ and $y_0 = 0.9$

n	α_n	x_{n+1}	y_{n+1}
0	−0.1154	−0.8886	1.0866
1	−0.0471	−0.9170	0.9548
2	−0.0928	−0.9273	1.0773
3	−0.1262	−0.9791	0.9918
8	−0.1055	−0.99878	0.99878
9	−0.1042	−0.99956	0.99923
10	−0.1085	−0.999945	0.99981
11	−0.1131	−0.99999942	0.9999931
12	−0.1181	−0.99999999983	0.9999999940
13	−0.1199	−1.000000000000004	0.9999999999999959

As we can see from these results the values of α_n found are outside the disc C. However we obtain quite a rapid, although not quadratic, convergence.

6.2.5 Continuation methods

The main problem for solving nonlinear equations $f(x) = 0$ by an iterative method is to find an initial approximation belonging to the domain of convergence of the solution. A possibility for obtaining such a starting point (or, in other words, to widen the domain of convergence) is to use a continuation method (also called homotopy method).

Let $g(x) = 0$ be another equation whose solution x_0 is known. We consider the equation

$$H(x,t) = (1-t)f(x) + t\,g(x) = 0.$$

We have

$$H(x,1) = g(x)$$
$$\text{and} \quad H(x,0) = f(x).$$

We assume that the solution $x(t)$ of $H(x,t) = 0$ depends continuously on t. We have $x(1) = x_0$.

A continuation method, as described for example by Ortega and Rheinboldt [343], consists in partitioning the interval $[0,1]$ as

$$t_0 = 1 > t_1 > t_2 > \cdots > t_N = 0$$

and solving the equations

$$H(x,t_i) = 0\,, \qquad i = 0,\ldots,N$$

by some iterative method where the solution of the i-th problem is used as the starting point for solving the $(i+1)$-th problem. If t_i and t_{i+1} are sufficiently close such a starting point will belong to the domain of convergence.

Instead of partitioning the interval $[0,1]$ into a finite number of subintervals we can also take a sequence (t_n) with

$$t_0 = 1 > t_1 > t_2 > \cdots > 0$$

and $\lim_{n\to\infty} t_n = 0$. Let us set $x_n = x(t_n)$, the solution of $H(x,t_n) = 0$. We shall now show how to extrapolate the sequence (x_n) which converges to the solution x of $f(x) = 0$ and make use of four strategies

1. Exact solution of $H(x,t_n) = 0$ (to the computer precision).

2. Inexact solution of $H(x,t_n) = 0$ by making only one iteration.

3. Knowing x_{n-1} and x_{n-2}, exact solution of $H(x,t_n) = 0$ with the starting point $y_n = x_{n-2} - \dfrac{\Delta x_{n-2}}{\Delta t_{n-2}} \cdot t_{n-2}$.

4. Same as 3 but inexact solution with one iteration.

Let us give the results obtained (number of exact digits) with $g(x) = 1 - e^{-x}$ whose solution is $x = 0.5671432904097838\ldots$. The iterative method used was Regula Falsi (since this method needs two starting points we took x_n and $x_n + 0.1$). The sequence (x_n) was then accelerated by

$$T_n = x_n - \frac{\Delta x_n}{\Delta t_n} \cdot t_n\,, \qquad n = 0, 1, \ldots$$

and $t_n = 0.1^n$.

	Strategy 1			Strategy 2		Strategy 3			Strategy 4	
n	x_{n+1}	T_n	ν	x_{n+1}	T_n	x_{n+1}	T_n	ν	x_{n+1}	T_n
0	1.29	1.27	8	0.84	1.32	1.29	1.27	8	0.84	1.32
1	2.31	3.52	7	2.36	1.96	2.31	3.52	6	2.26	2.01
2	3.31	5.54	6	3.38	4.78	3.31	5.54	5	3.32	4.11
3	4.31	7.54	6	4.38	6.91	4.31	7.54	5	4.31	6.26
4	5.31	9.54	6	5.38	8.42	5.31	9.54	4	5.31	9.50
5	6.31	11.54	6	6.38	10.29	6.31	11.54	4	6.31	12.02
6	7.31	13.54	5	7.38	11.97	7.31	13.54	4	7.31	14.17
7	8.31	15.62	5	8.38	13.75	8.31	15.62	4	8.31	15.66
8				9.38	15.30					

where ν is the number of iterations for obtaining the exact solution.

As we can see from these results the four strategies work quite well since the number of exact digits is roughly doubled. The number of iterations needed for solving exactly the problem is only slightly decreased by using strategy 3. The results obtained by strategy 4 are a little bit better than those given by strategy 2.

Other homotopy methods are described by Keller [259].

These results are given here only to show a new application of extrapolation methods but no theory exists so far.

6.3 Eigenelements

In this section we shall see how the ε-algorithms (scalar, vector and topological) can be used in eigenelement problems. Of course since algorithms for computing eigenelements are iterative, they can first be used for their acceleration. However it will not be our aim to develop this point of view here, but we shall present another more interesting feature of the ε-algorithms: as we shall see the ε-algorithms give us new methods for computing the eigenvalues of a matrix. This is not a surprising fact since, as we saw in section 6.2.1, these algorithms are direct methods for solving systems of linear equations and such methods (as Gaussian elimination or Householder factorization) lead to iterative methods for computing eigenvalues. Thus the ε-algorithms will give us an extension of the Rayleigh quotient (or power) method for computing simultaneously several eigenelements. Another interesting problem, having many applications, is the computation of the derivatives of the

eigenelements of a matrix whose elements depend on parameters. In this case, the ε-algorithms will give a direct method for solving the problem.

6.3.1 Eigenvalues and eigenvectors

The Rayleigh quotient (or power) method is a well known method for computing the dominant eigenvalue and the corresponding eigenvector of a matrix. It is as follows. We consider the sequence of vectors

$$x_{n+1} = Ax_n\,, \qquad n = 0, 1, \ldots$$

where x_0 is a given arbitrary vector. Let y be an arbitrary vector. We set $y_n = (y, x_n)$ and we consider the sequence

$$S_n = y_{n+1}/y_n\,, \qquad n = 0, 1, \ldots$$

Under some assumptions (S_n) converges to the dominant eigenvalue of A. Let $\lambda_1, \ldots, \lambda_p$ (p is the dimension of A) and $v_1, \ldots, v_p$, be the eigenvalues of A and the corresponding eigenvectors. If we assume that $v_1, \ldots, v_p$ are linearly independent, that $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_p|$, that $(x_0, v_1) \neq 0$ and that $(y, v_1) \neq 0$ then (S_n) converges to λ_1 and we have

$$S_n - \lambda_1 = O\left[(\lambda_2/\lambda_1)^n\right].$$

After having obtained λ_1 either the matrix A or the method can be modified in order to obtain the sub-dominant eigenvalue λ_2. But, of course, the drawback of this procedure is that it can only be started after achieving convergence to λ_1 since the knowledge of the value of λ_1 is needed for performing a deflation of A or a λ-difference.

Thanks to the results given in section 6.1.5, the ε-algorithms can be used to compute simultaneously several eigenvalues thus giving an extension of the Rayleigh quotient method. There are two possibilities: to apply the vector or the topological ε-algorithm to the vector sequence (x_n) or to apply the scalar ε-algorithm to the scalar sequence (y_n). In the first case we shall obtain sequences of scalars converging (under some assumptions) to the eigenvalues and, at the same time, sequences of vectors converging to the eigenvectors while, with the scalar ε-algorithm, we shall only produce scalar sequences converging to the eigenvalues. Of course, using the scalar ε-algorithm, saves much time and storage but,

on the other hand, the eigenvectors will have to be computed separately afterwards (by solving systems of linear equations).

The algorithm is the following

- Choose x_0 and y such that $(x_0, v_i) \neq 0$ and $(y, v_i) \neq 0$ for $i = 1, \ldots, m \leq p$.
- Compute $x_{n+1} = Ax_n$ and $y_n = (y, x_n)$ for $n = 0, 1, \ldots$
- Apply the vector ε-algorithm to (x_n) and set

$$a_k^{(n)} = \left(y, \varepsilon_{2k}^{(n+1)}\right) \Big/ \left(y, \varepsilon_{2k}^{(n)}\right), \quad k = 0, \ldots, m-1;\ n = 0, 1, \ldots$$
$$b_k^{(n)} = \left(y, \varepsilon_{2k+1}^{(n)}\right) \Big/ \left(y, \varepsilon_{2k+1}^{(n+1)}\right), \quad k = 0, \ldots, m-1;\ n = 0, 1, \ldots$$

or apply the scalar ε-algorithm to (y_n) and set

$$c_k^{(n)} = \varepsilon_{2k}^{(n+1)} \Big/ \varepsilon_{2k}^{(n)}, \quad k = 0, \ldots, m-1;\ n = 0, 1, \ldots$$
$$d_k^{(n)} = \varepsilon_{2k+1}^{(n)} \Big/ \varepsilon_{2k+1}^{(n+1)}, \quad k = 0, \ldots, m-1;\ n = 0, 1, \ldots$$

We have the following result, which is a direct consequence of theorem 6.13

Theorem 6.27

If $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_m| \geq |\lambda_{m+1}| \geq \cdots \geq |\lambda_p|$ *then, for* $k = 0, \ldots, m-1$, *the sequences* $\left(a_k^{(n)}\right)$, $\left(b_k^{(n)}\right)$, $\left(c_k^{(n)}\right)$ *and* $\left(d_k^{(n)}\right)$ *converge to* λ_{k+1} *when* n *tends to infinity. Moreover, with the vector* ε*-algorithm*

$$\lim_{n\to\infty} \varepsilon_{2k}^{(n)} \Big/ \left(y, \varepsilon_{2k}^{(n)}\right) = \lim_{n\to\infty} \left(y, \varepsilon_{2k+1}^{(n)}\right) \varepsilon_{2k}^{(n)} = v_{k+1}.$$

Instead of the vector ε-algorithm, the topological ε-algorithm can be used. We can also make use of the iterated vector or scalar Δ^2 process. The rate of convergence of the four previous sequences is $O\left[(\lambda_{k+2}/\lambda_{k+1})^n\right]$ and thus they can be accelerated by the scalar ε-algorithm producing sequences $\left(\varepsilon_{2q}^{(n)}\right)$ such that

$$\varepsilon_{2q}^{(n)} = \lambda_{k+1} + O\left[(\lambda_{k+q+2}/\lambda_{k+1})^n\right] \qquad (n \to \infty).$$

Variants of the preceding methods exist and can be implemented. As shown by the examples given in Brezinski [56], it seems that the preceding methods can suffer from severe numerical instability which comes

from the many scalar products needed. Thus they must be programmed very carefully and, in particular, the scalar products must be computed with the correction proposed by Pichat [358] which is explained in section 7.2. These methods deserve further studies both on the theoretical and practical levels.

Let us return to the initial Rayleigh quotient method for computing λ_1. If the scalar ε-algorithm is applied to the sequence (S_n) thus, as seen above, we shall have

$$\varepsilon_{2q}^{(n)} = \lambda_1 + O\left[(\lambda_{q+2}/\lambda_1)^n\right] \qquad (n \to \infty).$$

An important application of this procedure is the determination of the optimal value w_{opt} of the parameter w in the over-relaxation method.

We have to solve the system $Ax = b$. Let D be the matrix formed by the diagonal of A, $-E$ its strictly lower triangular part and $-F$ its strictly upper triangular part.

We set

$$\begin{aligned} &M = (D - wE)/w, \qquad w \neq 0 \\ &N = [(w-1)D + wF]/w \\ &L_w = M^{-1}N \\ &c = M^{-1}b \end{aligned}$$

and we consider the iterations

$$x_{n+1} = L_w x_n + c$$

where x_0 is arbitrarily chosen. This is the so-called over-relaxation method (SOR).

When the matrix A is symmetric positive definite and block tridiagonal (a situation which arises often when discretizing boundary value problems) the optimal value w_{opt} of w (that is the value leading to the fastest convergence) is given by (see Varga [448])

$$w_{opt} = 2\left(1 + \sqrt{1 - |\lambda_1|}\right)^{-1/2}$$

where λ_1 is the dominant eigenvalue of L_1. Thus we have to determine λ_1. We first begin the iterations with $w = 1$ and use the Rayleigh quotient method for obtaining λ_1, that is we compute

$$S_n = (y, \Delta x_{n+1})/(y, \Delta x_n), \qquad n = 0, 1, \ldots$$

which converges to λ_1 as $O[(\lambda_2/\lambda_1)^n]$. In order to obtain faster convergence, the scalar ε-algorithm is applied to (S_n), thus leading to a better estimation of λ_1. After having obtained λ_1, the iterations are continued with w_{opt} as given by the formula above.

Let us apply this procedure to the 200×200 tridiagonal matrix with $a_{ii} = 2, a_{i,i+1} = a_{i+1,i} = -1$, the other elements being zero.

We have $\lambda_1 = 0.999755728800233\ldots$, $w_{opt} = 1.969221\ldots$, $S_{100} = 0.999367119297927$ and $\varepsilon_{80}^{(19)} = 0.999693452381627$.

With the exact value of w_{opt} we add one exact digit every 71 iterations. If λ_1 is estimated by S_{100} then $w = 1.950919$ and we only gain one exact digit every 200 iterations. If λ_1 is estimated by $\varepsilon_{80}^{(19)}$ then $w = 1.965582$ and we shall gain one digit every 111 iterations. Thus an important improvement can be brought by ε-algorithm in this case.

6.3.2 Derivatives of eigensystems

Many problems of engineering and physics require the computation of the partial derivatives of the eigenvalues and eigenvectors of matrices whose elements depend on parameters. This is, for example, the case in optimal design of aircrafts, system identification of structures, and reanalysis of structures when some parameters are slightly changed.

This problem can be solved by direct methods. We have

$$A\boldsymbol{x}_i = \lambda_i \boldsymbol{x}_i\,, \qquad i = 1, \ldots, p$$

where A is a $p \times p$ matrix.

We assume that the elements of the matrix A depend on parameters $p_1, p_2, \ldots, p_m$ and we shall denote respectively by $\lambda_{i,j}$, $\boldsymbol{x}_{i,j}$ and A_j the partial derivatives with respect to p_j of λ_i, $\boldsymbol{x}_i$, and the matrix A (that is A_j will be the matrix whose elements are the partial derivatives with respect to p_j of the corresponding elements of A). We have

$$A_j \boldsymbol{x}_j + A\boldsymbol{x}_{i,j} = \lambda_{i,j}\, \boldsymbol{x}_i + \lambda_i\, \boldsymbol{x}_{i,j}.$$

If $\boldsymbol{x}_i$ is normalized such that $\boldsymbol{x}_i^* \boldsymbol{x}_i = 1$ (where the star denotes the conjugate transpose) then, by differentiation, we shall have

$$\boldsymbol{x}_i^* \boldsymbol{x}_{i,j} = 0$$

and we obtain

$$\begin{aligned} \lambda_{i,j} &= x_i^* (A - \lambda_i I) x_{i,j} + x_i^* A_j x_i \\ x_{i,j} &= [(A_j - \lambda_{i,j} I) x_i + A x_{i,j}] / \lambda_i . \end{aligned}$$

Thus $\lambda_{i,j}$ and $x_{i,j}$ can be directly obtained by solving this linear system of $p+1$ equations in $p+1$ unknowns. However the solution of this system raises many numerical problems due to the ill-conditioning which leads Rudisill and Chu [377] to propose the following iterative method

$$\begin{aligned} \mu_k &= x_i^* (A - \lambda_i I) u_k + x_i^* A_j x_i \\ u_{k+1} &= [(A_j - \mu_k I) x_i + A u_k] / \lambda_i , \qquad k = 0, 1, \ldots \end{aligned}$$

It was proved by Andrew [9] that if A is nondefective and if u_0 is chosen to satisfy

$$x_i^* u_0 = 0$$

then, $\forall k$

$$u_k = x_{i,j} + \sum_{\substack{r=1 \\ r \neq i}}^{p} (\lambda_r / \lambda_i)^k y_r$$

where the y_r's are some vectors. Thus if $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_p|$ and if $i = 1$, (u_k) and (μ_k) tend respectively to $x_{1,j}$ and $\lambda_{1,j}$ when k tends to infinity and the rate of convergence is $O\left(|\lambda_2/\lambda_1|^k\right)$. For $i \neq 1$, the sequences (u_k) and (μ_k) fail to converge due to the presence of terms in $(\lambda_r/\lambda_i)^k$ for $r < i$ which grow exponentially.

However, as remarked by Tan [423, 424, 425, 426] in a series of papers, if the ε-algorithm (scalar, vector or topological) is applied to (u_k) then, thanks to theorems 2.18, 4.2 and 4.3 on its kernel, we shall have

$$\varepsilon_{2p-2}^{(m)} = x_{i,j} , \qquad m = 0, 1, \ldots$$

Thus the ε-algorithm has transformed the preceding iterative method into a direct one. The performances of these algorithms in the presence of rounding errors was investigated and a refinement procedure suggested by Tan [425].

Another extrapolation method was also proposed by Tan [425]. He sets $i = 1$ and

$$w_0^{(k)} = u_k , \qquad k = 0, 1, \ldots$$

and then computes

$$w_r^{(k)} = \frac{\lambda_1 w_{r-1}^{(k+1)} - \lambda_{r+1} w_{r-1}^{(k)}}{\lambda_1 - \lambda_{r+1}}, \qquad r = 1, 2, \ldots, p-2.$$

If $\lambda_j \neq \lambda_1$ for $j = 2, \ldots, r+2$ then

$$w_r^{(k)} = x_{1,j} + \sum_{p=r+2}^{n} (\lambda_p/\lambda_1)^k z_p, \qquad r \leq p-2$$

where the z_p's are some vectors.

This procedure can be extended to the non-dominant eigenelements. Tan [427, 429] also used the minimal polynomial and the reduced rank extrapolation methods which are shown to be effective. When the partial derivatives of several eigenvalues and eigenvectors are needed, simultaneous iteration can be used as described by Tan and Andrew [430]. It consists in generating a sequence of $r \times r$ matrices, (M_k), and a sequence of $p \times r$ matrices, (U_k) as follows

$$\begin{aligned} M_k &= (X^*X)^{-1} X^* (A_j X + AU_k - U_k \Lambda) \\ U_{k+1} &= (A_j X + AU_k - XM_k) \Lambda^{-1}, \qquad k = 0, 1, \ldots \end{aligned}$$

where $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_r)$ and X is the $p \times r$ matrix with columns $x_1, \ldots, x_r$. It is assumed that $x_i^* x_i = 1$ for $i = 1, \ldots, p$. Application of the vector or the topological ε-algorithm to this method again yields the exact solution.

Numerical results can be found in the above mentioned references.

6.4 Integral and differential equations

Extrapolation algorithms have several applications in the solution of integral and differential (ordinary and partial) equations.

When using implicit methods for integrating ordinary differential equations one has to solve, at each step, a system of nonlinear equations. This solution can be obtained via the derivative-free methods described in section 6.2.4. The solution of boundary value problems can be transformed into the solution of initial value problems by shooting methods. Such a procedure also involves solving systems of nonlinear equations. The solution of boundary value problems on a semi-infinite interval can be improved by extrapolation.

New methods (nonlinear and A-stable) for the integration of ordinary differential equations can be constructed from the confluent form of the ϱ-algorithm or from Padé approximation.

The inversion of the Laplace transform, which has so many important applications, can be obtained by rational interpolation and approximation of the image and then inversion of this rational function.

Finally extrapolation procedures can be used for improving the accuracy of the solution of partial differential equations obtained by discretization methods. The E-algorithm can also be used in the minimization of functionals obtained in variational methods.

Of course when integrating numerically differential equations, it is possible to use several step sizes $h_0, h_1, \ldots$ and then to extrapolate the approximate values of the solution at the coincident points thus obtained. Such an application of extrapolation methods was studied by Gragg [189, 190], Bulirsch and Stoer [107] and more recently by Deuflhard [142].

6.4.1 Implicit Runge-Kutta methods

When integrating stiff differential equations numerically it is mandatory to use A-stable methods. It is well known that linear explicit A-stable methods do not exist and thus one has to use implicit one step methods of the form

$$y_{n+1} = y_n + h\phi(x_n, y_n, x_{n+1}, y_{n+1}, h)$$

(where y_n is an approximation of the solution at x_n) or implicit multistep methods of the form

$$\alpha_k y_{n+1} + \cdots + \alpha_0 y_{n-k+1} = h(\beta_k f(x_{n+1}, y_{n+1}) + \cdots + \beta_0 f(x_{n-k+1}, y_{n-k+1})).$$

In both cases y_{n+1} is given implicitely as the solution of a system of (usually nonlinear) equations and thus the derivarive-free methods described in section 6.2.4 are very useful.

For example, the vector ε-algorithm was used by Alt [8] in semi-

implicit Runge-Kutta methods. Such method is defined by

$$\begin{aligned} k_1 &= f(x_n + \theta_1 h, y_n + a_{11} h k_1) \\ k_2 &= f(x_n + \theta_2 h, y_n + a_{21} h k_1 + a_{22} h k_2) \\ &\vdots \qquad\qquad \vdots \\ k_r &= f(x_n + \theta_r h, y_n + a_{r1} h k_1 + \cdots + a_{rr} h k_r) \\ y_{n+1} &= y_n + h(c_1 k_1 + \cdots + c_r k_r) \end{aligned}$$

where the θ_i's, the a_{ij}'s and the c_i's are constants chosen so that the method has the highest possible order.

Thus, at each step, r systems of (nonlinear) equations have to be solved for obtaining $k_1, k_2, \ldots, k_r$. These solutions can be obtained by the derivative free methods described in section 6.2.4, where the starting initial value for the iterations is given by an explicit method thus leading to very fast and efficient processes.

6.4.2 Boundary value problems

Let us consider a system of p differential equations of the first order

$$y'(t) = f(t, y(t))$$

where $y : \mathbb{R} \longmapsto \mathbb{R}^p$ and $f : \mathbb{R} \times \mathbb{R}^p \longmapsto \mathbb{R}^p$, f satisfying a Lipschitz condition.

Let $a \leq t_1 < t_2 < \cdots < t_k \leq b$, be k distinct points in the interval of integration and let $g : (\mathbb{R}^p)^k \longmapsto \mathbb{R}^p$. The solution of the preceding systems is subject to satisfy the multipoint boundary conditions

$$g(y(t_1), \ldots, y(t_k)) = 0.$$

In order to integrate this system, we have to transform it into an initial value problem, that is we have to find the missing initial conditions $x = y(a) \in \mathbb{R}^p$ such that

$$g(y(t_1, x), \ldots, y(t_k, x)) = 0$$

where $y(t, x)$ is the solution of the differential equation satisfying

$$y(a, x) = x.$$

Thus the problem of finding x reduces to the solution of a system of p equations in p unknowns which are the components of x. This system of

equations can be very efficiently solved by the derivative free methods explained in section 6.2.4.

We choose arbitrarily a value x_0 for u and we set $u_0 = x_0$. We integrate (numerically) the system with u_0 as an initial condition and we obtain approximate values $y_i(u_0)$ of $y(t_i, u_0)$ for $i = 1, \ldots, k$. Then we compute

$$u_1 = u_0 + \alpha\, g\,(y_1(u_0), \ldots, y_k(u_0))$$

where α is some non-zero parameter.

We integrate numerically the system with u_1 as an initial condition. We obtain approximate values $y_i(u_1)$ of $y(t_i, u_1)$ for $i = 1, \ldots, k$ and we set

$$u_2 = u_1 + \alpha\, g\,(y_1(u_1), \ldots, y_k(u_1))\,.$$

And so an up to u_{2p} where p is the dimension of the system.

Then we apply to vector ε-algorithm or the topological ε-algorithm to the vectors $u_0, u_1, \ldots, u_{2p}$ and we obtain a vector denoted by $\varepsilon_{2p}^{(0)}$. We set

$$x_1 = \varepsilon_{2p}^{(0)}$$

and we start again the whole business with $u_0 = x_1$ and so on.

If the conditions of theorem 6.26 are satisfied then the sequence (x_n) converges quadratically to x.

The advantage of this procedure on shooting methods using finite differences is that the dimension of the system to be solved is equal number of unknown initial conditions and not to the number of discretization points in the interval of integration.

Instead of using the ε-algorithm for solving the system, we can use Henrici's method and the H-algorithm thus saving much computational time since only $u_0, u_1, \ldots, u_{p+1}$ are needed at each step.

Let us consider the example given by Brezinski and Rieu [100]

$$\begin{aligned} y_1'(t) &= y_2(t) - y_3(t) \\ y_2'(t) &= y_1^2(t) + y_2(t) \\ y_3'(t) &= y_1^2(t) + y_3(t) \end{aligned}$$

with the conditions

$$y_1(0) = 1, \quad y_2(1/2) = e - 4, \quad y_3(1) = -4 - e - e^2.$$

This boundary value problem has a unique solution which is

$$\begin{aligned} y_1(t) &= 2 - e^t \\ y_2(t) &= -4 - (4t-2)e^t + e^{2t} \\ y_3(t) &= -4 - (4t-3)e^t + e^{2t} \end{aligned}$$

and the missing initial conditions are

$$y_1(0) = 1, \quad y_2(0) = -1, \quad y_3(0) = 0.$$

Using a Runge-Kutta method with an error less than $0.5 \cdot 10^{-9}$ and starting with $x_0 = 0$, the vector ε-algorithm gives

$$\begin{aligned} x_1 &= (0.921428452, -0.951029784, -0.100612601) \\ x_2 &= (0.999698042, -1.002885341, -7.49 \cdot 10^{-4}) \\ x_3 &= (0.999999542, -0.999997855, -3.70 \cdot 10^{-7}) \\ x_4 &= (0.999999995, -0.999999996, -1.10 \cdot 10^{-10}). \end{aligned}$$

The next iterations give the same values as x_4 and thus the boundary value problem has been solved with the same accuracy as the numerical integration procedure used. The Runge-Kutta method was used 24 times (4 iterations and $p = 3$) for obtaining these results.

A problem related to the preceding one and which can be treated by a similar method is that of free boundary. Let us give a very simple example for illustrating it. We consider the differential equation

$$y''(x) = f(x, y(x)), \quad x \in [a, b]$$

with the boundary conditions

$$\begin{aligned} y'(a) &= y_0 \\ y'(b) &= z_0 \end{aligned}$$

where y_0 and z_0 are given but where b is unknown but satisfies

$$G(b, y(b)) = 0.$$

The problem consists in finding b.

If we set

$$z(x) = y'(x)$$

the problem becomes

$$\begin{aligned} y'(x) &= z(x) \\ z'(x) &= f(x, y(x)) \\ z(a) &= y_0 \\ z(b) &= z_0 \end{aligned}$$

and thus we have to find the missing initial condition $y(a)$ and the value of b, that is we have to solve the system

$$\begin{aligned} z(y(a), b) &= z(b) = z_0 \\ G(b, y(b)) &= 0. \end{aligned}$$

This system can be solved by one of the derivative free methods explained in section 6.2.4.

For a survey on free and moving boundary problems see Cryer [124].

Let us now consider the case of second-order two-point boundary value problems on the semi-infinite interval $[a, \infty)$. A technique for dealing with such problems is to replace the boundary condition at infinity by the same condition at $x = R$ for a sufficiently large value of R. The practical difficulty is to decide what *large* means.

Croft [122] proposed to use several values of R and to extrapolate the computed solutions at the various points in the intervals of integration.

We consider the problem

$$y''(x) - (1 + f(x))y(x) = 0, \quad y(0) = 1, \quad y(\infty) = 0$$

with $\lim_{x \to \infty} f(x) = 0$ and $\int_0^\infty |f(x)|\, dx < \infty$.

The general solution of this problem is

$$y(x) = Ay_1(x) + By_2(x)$$

with $y_1(x) = e^x (c_1 + o(1))$ and $y_2(x) = e^{-x} (c_2 + o(1))$.

Imposing the boundary conditions $y(0) = 1$ and $y(R) = 0$ we obtain a solution $y_R(x)$ given by

$$y_R(x) = \frac{y_2(R)y_1(x) - y_1(R)y_2(x)}{y_2(R)y_1(0) - y_1(R)y_2(0)}$$

while the exact solution is $y(x) = y_2(x)/y_2(0)$.

Let us now take a sequence (R_n) of values of R tending to infinity and set $E_n(x) = y(x) - y_{R_n}(x)$. We have

$$E_n(x) = \frac{y_2(x)y_1(0) - y_1(x)y_2(0)}{y_2(R_n)y_1(0) - y_1(R_n)y_2(0)} \cdot \frac{y_2(R_n)}{y_2(0)}$$

and

$$\lim_{n\to\infty} \frac{E_{n+1}(x)}{E_n(x)} = \lim_{n\to\infty} e^{-2(R_{n+1}-R_n)}.$$

Thus if $R_{n+1} = R_n + c$ the convergence is linear and can be accelerated by many algorithms such as Aitken's Δ^2 process, Levin's u-transformation, the ε-algorithm, the Θ-algorithm,....

If $R_n = c^n$ with $c > 1$ then $\lim_{n\to\infty} E_{n+1}(x)/E_n(x) = 0$. The convergence is super-linear and does not need, in general, to be accelerated. When needed the processes given by Jones [247] can be used.

If $\lim_{n\to\infty} (R_{n+1} - R_n) = 0$ (which is the case if, for example, $R_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$) then $\lim_{n\to\infty} E_{n+1}(x)/E_n(x) = 1$. In this case it will be difficult to accelerate the convergence since the set of logarithmic sequence is not accelerable.

Another alternative is to use the E-algorithm since

$$E_n \sim C(x) \cdot \frac{e^{-R_n}}{y_1(0)e^{-R_n} - y_2(0)e^{R_n}} = \frac{C(x)}{y_2(0)} \cdot e^{-2R_n}\left[1 + \frac{y_1(0)}{y_2(0)} \cdot e^{-2R_n} + \cdots\right].$$

Since the asymptotic expansion of the error is known we can use the E-algorithm with the auxiliary sequences

$$g_j(n) = e^{-2jR_n}, \qquad j = 1, 2, \ldots$$

We have

$$g_{j+1}(n)/g_j(n) = e^{-2R_n} \quad \text{and} \quad \lim_{n\to\infty} g_{j+1}(n)/g_j(n) = 0$$

$$g_j(n+1)/g_j(n) = e^{-2j(R_{n+1}-R_n)}.$$

Thus if $\lim_{n\to\infty} (R_{n+1} - R_n) = b \neq 0$ then $\lim_{n\to\infty} g_{j+1}(n)/g_j(n) = b_j$ with $b_j \neq 1$ and $\forall i \neq j$, $b_i \neq b_j$ which shows that all the conditions of theorem 2.10 are satisfied and it follows that, $\forall k \geq 1$

$$\lim_{n\to\infty} \left(E_k^{(n)} - y(x)\right) \Big/ \left(E_{k-1}^{(n+1)} - y(x)\right) = 0.$$

Let us consider the example

$$y''(x) - \left(1 + 2/x^2\right) y(x) = 0$$

with $y(1) = 2e^{-1}$ and $y(\infty) = 0$. Let us choose $R_n = 2.1 + n \cdot 0.1$ for $n = 0, 1, \ldots$. We shall extrapolate the approximate solutions at $x = 1.5$. We have $y(1.5) = 0.371884$ and we obtain

$y_{R_n}(1.5)$	Δ^2	Θ_2	u
0.308751			
0.322974			0.324417
0.333765	0.367695		0.301638
0.342026	0.368996	0.373826	0.326916
0.348397	0.369875	0.373190	0.426446
0.353341	0.370475	0.372770	0.368589
0.357198	0.370888	0.372490	0.372489
0.360221	0.371176	0.372301	0.371911
0.362600	0.371377	0.372173	0.371909
0.364478	0.371519	0.372086	0.371890

With the E-algorithm and $R_n = 2.4 + n \cdot 0.4$ for $n = 0, 1, \ldots$ we have

$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
.342026							
.360221	.375068						
.367145	.372794	.372219					
.369910	.372167	.372008	.371987				
.371048	.371977	.371929	.371921	.371918			
.371526	.371916	.371900	.371897	.371896	.371896		
.371729	.371895	.371890	.371889	.371888	.371888	.371888	
.371816	.371888	.371886	.371886	.371885	.371885	.371885	.371885

Other examples can be found in the paper by Croft [122].

Semilinear singular perturbation problems can be extrapolated by Richardson process, see Herceg, Vulanović and Petrović [227].

Richardson extrapolation of multi-grid methods for the solution of boundary value problems is described by Hackbusch [209].

6.4.3 Nonlinear methods

Let us consider the initial value problem

$$y'(x) = f(x, y(x)), \qquad x \in [a, b]$$
$$y(a) = y_0.$$

We set

$$g(t) = y(x + h - 1/t).$$

Thus

$$y(x + h) = \lim_{t \to \infty} g(t).$$

Let us take the confluent form of the ϱ-algorithm (see section 5.4) to estimate this limit. We have

$$\varrho_2(t) = g(t) - \frac{2g'^2(t)}{g''(t)}$$

with $g'(t) = t^{-2}y'(x + h - 1/t)$ and $g''(t) = -2t^{-3}y'(x + h - 1/t) + t^{-4}y''(x + h - 1/t)$.

Taking $\varrho_2(t)$ as an approximation of $y(x + h)$ and $t = 1/h$, Brezinski [47] obtained the following one step explicit formula

$$y_{n+1} = y_n + \frac{2hy_n'^2}{2y_n' - hy_n''}$$

with $y_n' = f(x_n, y_n)$ and $y_n'' = f_x'(x_n, y_n) + y_n' f_y'(x_n, y_n)$, y_n being an approximate value of the exact solution at the point $x_n = a + nh$.

The great advantage of this method is that it is explicit and A-stable. Moreover it has order 2.

Let us apply it to

$$y'(x) = 1 + y(x)/x \quad \text{with} \quad y(x) = 1$$

whose solution is $y(x) = x(1 + \ln x)$. We obtain the following relative errors

x	$h = 0.01$	$h = 0.02$	$h = 0.04$
1.10	$-0.23 \cdot 10^{-5}$	$-0.92 \cdot 10^{-5}$	
1.20	$-0.37 \cdot 10^{-5}$	$-0.15 \cdot 10^{-4}$	$-0.60 \cdot 10^{-4}$
1.30	$-0.46 \cdot 10^{-5}$	$-0.18 \cdot 10^{-4}$	
1.40	$-0.52 \cdot 10^{-5}$	$-0.21 \cdot 10^{-4}$	$-0.84 \cdot 10^{-4}$

If this scheme is applied to $y'(x) = -10\,y(x)$ with $y(0) = 1$ and compared with the improved Euler method which has the same order but is not A-stable we obtain the following relative errors

h	x	Above A-stable method	Improved Euler method
0.01	0.3	$0.25 \cdot 10^{-2}$	$-0.54 \cdot 10^{-2}$
	0.6	$0.50 \cdot 10^{-2}$	$-0.11 \cdot 10^{-1}$
	1.0	$0.83 \cdot 10^{-2}$	$-0.18 \cdot 10^{-1}$
0.04	0.6	$0.79 \cdot 10^{-1}$	-0.24
	1.0	0.13	-0.43
0.16	0.96	0.97	$-0.15 \cdot 10^{4}$

In the preceding formula hy_n'' can be replaced by its approximation $y_n' - y_{n-1}'$. We now have a two-step formula with the same order 2 which is also A-stable.

This formula can be derived in a different way which allows generalization. We have

$$y(x) = y(x_n) + (x - x_n) f(x_n, y(x_n)) + \frac{(x - x_n)^2}{2} \cdot f'(x_n, y(x_n)) + \cdots$$

Since $y(x_n)$ is unknown, we shall replace it by y_n and consider the Taylor series

$$g(x) = y_n + (x - x_n) f(x_n, y_n) + \frac{(x - x_n)^2}{2} \cdot f'(x_n, y_n) + \cdots$$

Let $[m/k]_g$ be the Padé approximant of this series with a numerator of degree m and a denominator of degree k. Then we shall take

$$y_{n+1} = [m/k]_g (x_{n+1}).$$

Our preceding method corresponds to $m = k = 1$.

Such methods have been studied by Wambecq [454]. They have been extended to systems of differential equations either by treating separately each function of the system or by more vectorial methods, see Wambecq [455]. Replacing the derivatives of f by approximations leads to multistep methods. Prediction-correction methods based on similar ideas can also be derived. A survey of these nonlinear methods can be found in Cuyt and Wuytack [131]. As in the linear case, they can be

coupled with mesh refinements and extrapolation techniques similar to those described by Gragg [190]. However it must be remarked that the presence of a denominator in such methods can give some difficulties due to a possible division by zero.

6.4.4 Laplace transform inversion

The inversion of the Laplace transform is a very important problem due to the many applications of the Laplace transform and it is the subject of a vast literature. It is not our purpose here to give a survey of the numerical methods for solving this problem but only to emphasize on methods based on extrapolation techniques.

The inversion of the Laplace transform consists in finding the function f when F is known and

$$F(p) = \int_0^\infty e^{-pt} f(t) dt.$$

We shall consider two cases for our knowledge of F: some of the coefficients of its Taylor expansion around zero are known or its values at some points of the complex plane.

In the first case f can be replaced by a rational approximation, namely a Padé or a Padé-type approximation, and this rational function is then inverted, an idea originally due to Longman [296]. For inverting a rational function it must be first decomposed into its partial fractions which means that its poles must be computed. In the case of Padé type-approximation these poles are known. In the case of Padé approximation their computation can be replaced by a device, involving the summation of an infinite series, which will be described below. Although there is a connection between Padé approximation and the ε-algorithm (see section 2.3), we shall not describe this method here because its properties, and in particular its convergence, are strongly related to properties of Padé and Padé-type approximants and we refer the interested reader to van Iseghem [444] and Brezinski and van Iseghem [102].

Let us now assume that $F(p_n)$ is known for $n = 0, \ldots, 2k+1$ where the p_n's are distinct points in the complex plane. A method due to Fouquart [164] consists in constructing an interpolating rational function with a numerator of degree k and a denominator of degree $k+1$ in p since we must have $\lim_{p \to \infty} F(p) = 0$. This rational function can be

obtained by the algorithm for rational interpolation described in section 2.5 (pp. 102–104) where the $\varrho_k^{(n)}$'s are computed by the ϱ-algorithm with the initializations

$$\varrho_0^{(n)} = 1/F(p_n) , \qquad n = 0, \ldots, 2k+1$$

(instead of $F(p_n)$ since the numerator has degree k and the denominator $k+1$).

Now let us describe how to invert a rational function without decomposing it into partial fractions. It is an algorithm due to Longmam and Sharir [299] which is as follows. Let F have the form

$$F(p) = A \cdot \frac{p^m + a_1 p^{m-1} + \cdots + a_m}{p^n + b_1 p^{n-1} + \cdots + b_n}$$

with $m < n$ and let f be the function of which F is the Laplace transform. Then f has the form

$$f(t) = A \sum_{k=0}^{\infty} \frac{v_k}{k!} \cdot t^k$$

with

$$v_k = u_{k+m} + a_1 u_{k+m-1} + \cdots + a_m u_k , \qquad k = 0, 1, \ldots$$

and

$$\begin{aligned} u_k &= 0 \quad \text{for} \quad k = 0, \ldots, n-2 \\ u_{n-1} &= 1 \\ u_k &= -(b_1 u_{k-1} + \cdots + b_n u_{k-n}) , \qquad k = n, n+1, \ldots \end{aligned}$$

The series giving f is convergent and usually its convergence is fast enough. However if f converges too slowly, its convergence can be accelerated by the ε-algorithm.

The subroutine INVLAP performs this method for the Laplace transform inversion.

Let us first give a numerical example showing the propagation of rounding errors in the procedure. In section 2.5 we already saw that rounding errors can affect rational interpolation. Moreover the inversion of the Laplace transform is known to be an ill-conditioned problem.

Let us take

$$F(p) = \frac{2p^2 + 11p + 6}{p^3 + 9p^2 + 26p + 24}$$

which is the Laplace transform of

$$\begin{aligned} f(t) &= e^{-2t} - e^{-3t} + 2e^{-4t} \\ &= 2 - 7t + \frac{27}{2!}t^2 - \frac{109}{3!}t^3 + \frac{447}{4!}t^4 - \frac{1837}{5!}t^5 + \frac{7527}{6!}t^6 - \frac{30709}{7!}t^7 \\ &\quad + \frac{124767}{8!}t^8 - \frac{505117}{9!}t^9 + \frac{2039127}{10!}t^{10} - \dots \end{aligned}$$

We obtain the following coefficients for f

$$\begin{aligned} &0.2000000000000182 \cdot 10^1 \\ -&0.7000000000003372 \cdot 10^1 \\ &0.2700000000003619 \cdot 10^2 \\ -&0.1090000000002967 \cdot 10^3 \\ &0.4470000000020648 \cdot 10^3 \\ -&0.1837000000012881 \cdot 10^4 \\ &0.7527000000074291 \cdot 10^4 \\ -&0.3070900000040410 \cdot 10^5 \\ &0.1247670000021011 \cdot 10^6 \\ -&0.5051170000105436 \cdot 10^6 \\ &0.2039127000051420 \cdot 10^7 \end{aligned}$$

Let us now take

$$F(p) = \ln\left(1 + a^2/p^2\right), \qquad f(t) = 2(1 - \cos at)/t.$$

For $k = 7$, $p_n = 0.1 + nh$ for $n = 0, \dots, 2k+1$, $h = 2/(2k+1)$ and $a = 1.0$ we obtain the following results for the partial sums f_i of the series

$$\begin{aligned} t = -1.0 \quad f(t) &= -9.19395388 \cdot 10^{-1} \\ f_5(t) &= -9.19518557 \cdot 10^{-1} \\ t = -0.6 \quad f(t) &= -5.82214617 \cdot 10^{-1} \\ f_5(t) &= -5.82235377 \cdot 10^{-1} \\ t = -0.2 \quad f(t) &= -1.99334221 \cdot 10^{-1} \\ f_5(t) &= -1.99336600 \cdot 10^{-1} \\ t = \ 0.2 \quad f(t) &= \ 1.99334221 \cdot 10^{-1} \\ f_5(t) &= \ 1.99334361 \cdot 10^{-1} \\ t = \ 0.6 \quad f(t) &= \ 5.82214617 \cdot 10^{-1} \\ f_5(t) &= \ 5.82215743 \cdot 10^{-1} \\ f_9(t) &= \ 5.82214468 \cdot 10^{-1} \end{aligned}$$

A better accuracy was not achieved by using more terms of the series.

The inversion of the Laplace transform can also be achieved by computing the Bromwich integral

$$f(x) = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^{pt} F(p)\, dp$$

where c is such that F is regular in the half plane Re $(p) \geq c$. For this purpose Levin [284] proposed a modification of its t-transformation, called the P-transformation, which consists in the computation of

$$P_k = \frac{e^{ct}}{\pi} \operatorname{Re} \frac{\sum_{j=0}^{k} (-1)^j \binom{k}{j} (j+1)^{k-1} I_{j+1}(t) e^{-i(j+1)t} / F(c+i(j+1))}{\sum_{j=0}^{k} (-1)^j \binom{k}{j} (j+1)^{k-1} e^{-i(j+1)t} / F(c+i(j+1))}$$

where $I_{j+1}(t) = \int_0^{j+1} e^{ixt} F(c+ix)\, dx$. These integrals can be evaluated by any quadrature formula, for example by the Gauss-Legendre procedure.

The P-transformation is justified when F is such that there exists s so that $p^s F(p)$ is analytic and has no branch point at infinity but in fact the numerical examples show that this transformation is also efficient for other types of singularities of F.

Let us give some examples

- $F(p) = (p^2+1)^{-1/2}$, $f(t) = J_0(t)$, $c = 0.1$

t	$\lvert P_{14} - f(t) \rvert$
1	$2.0 \cdot 10^{-14}$
2	$1.2 \cdot 10^{-13}$
4	$1.9 \cdot 10^{-13}$
8	$5.0 \cdot 10^{-14}$
16	$6.0 \cdot 10^{-14}$

- $F(p) = p^{-1/2} \exp\left(-p^{-1/2}\right)$, $c = 0.001$,

$$f(t) = \frac{1}{2t\sqrt{\pi t}} \int_0^{\infty} u \exp\left(-\frac{u^2}{4t}\right) J_0(2\sqrt{u})\, du,$$

t	$\lvert P_{14} - f(t)\rvert$
1	$2.6 \cdot 10^{-12}$
10	$1.0 \cdot 10^{-15}$
100	$4.3 \cdot 10^{-12}$

- $F(p) = \dfrac{p \ln p}{p^2 + 1}$, $f(t) = -\sin t\ \mathrm{Si}(t) - \cos t\ \mathrm{Ci}(t)$, $c = 1$

t	$\lvert P_{14} - f(t)\rvert$
0.1	$0.9 \cdot 10^{-5}$
1.0	$3.0 \cdot 10^{-12}$
2.0	$1.0 \cdot 10^{-14}$
4.0	$1.0 \cdot 10^{-14}$

In these examples the integrals $I_{j+1}(t)$ were computed by the Gauss-Legendre quadrature formula up to 14 exact digits.

For the regularization of integral equations see section 6.2.3.

Richardson extrapolation of the iterated–collocation method for integral equations of the second kind on an arbitrary mesh is described by Lin, Sloan and Xie [289].

6.4.5 Partial differential equations

When solving partial differential equations by finite differences or finite elements methods *the problem of finding the most efficient methods ... is therefore of paramount importance* as stated by Marchuk and Shaidurov [307] who devoted a whole monograph to the extrapolation of finite difference methods by Richardson process.

It is not possible to give here even a survey of this book; let us only say that it studies the application of Richardson method to first-order ordinary differential equations, to the one-dimensional stationary diffusion equation, to elliptic equations, to nonstationary problems, to integral equations, to quasilinear equations, to eigenvalue problems and to boundary layer problems.

The initial Russian version of this book was written in 1979, that is before the discovery of the E-algorithm, and this is why the authors said that *the problem of finding more efficient algorithms* [than Richardson extrapolation process] *is a very pressing one.*

Thus, in this section, we shall mainly emphasize on extrapolation by techniques different from Richardson's and refer the reader interested by Richardson's process to the previously mentioned book (French translation, Marchouk and Shaydurov [306]).

Solving partial differential equations by finite differences or finite elements methods produces a sequence of vectors and thus any vector extrapolation process (chapter 4) can be used for their acceleration. In particular, some examples involving the vector ε-algorithm are given by Wynn [473] when he first proposed this algorithm.

A general approach to the extrapolation of discretization techniques was given by Qun and Tao [371]. We consider the linear operator equation

$$Lu = f$$

where $L : X \longmapsto Y$, $u \in X$, $f \in Y$, X and Y being Banach spaces. After discretization of this problem, it reduces to

$$L_h\, u(h) = f_h$$

where h is the step size and $\lim_{h \to 0} u(h) = u$. If the problem is multidimensional and if p denotes its dimension then $h = (h_1, \ldots, h_p)$ and, under some assumptions, we have

$$u(h) = u + \sum_{1 \leq |k| \leq m} c_k h^{2k} + O\left(|h|^{2m+1}\right)$$

with $k = (k_1, \ldots, k_p)$, $|k| = k_1 + \cdots + k_p$ and $h^k = h_1^{k_1} \cdot \ldots \cdot h_p^{k_p}$.

Let N_m be the number of elements in the index set $I_m = \{k, 1 \leq |k| \leq m\}$. We consider the N_m equations

$$u\left(h/2^n\right) = v + \sum_{1 \leq k \leq m} c_k\, h^{2k} \Big/ 2^{2(k,n)}, \qquad \forall n \in I_m$$

with $h/2^n = (h_1/2^{n_1}, \ldots, h_p/2^{n_p})$ and $(k, n) = \sum_{i=1}^{p} k_i n_i$.

Solving this system gives v which is a approximation of order $|h|^{2m+1}$ of u. This solution can be obtained by using the E-algorithm.

Another application of the E-algorithm was given by Morandi Cecchi and Redivo Zaglia [325] who considered the Cauchy problem for the biharmonic operator, an ill conditioned problem in the sense of Hadamard.

If the initial data are not analytical but only differentiable a certain number of times, one can think of approximating these data by polynomials and then solving the problem. However, due to the ill-conditioning of the problem this procedure leads to a bad result. The difficulty can be avoided by reducing the problem to the minimization of a functional subject to linear inequalities. This problem can be solved by linear programming or by least squares with normal equations as shown by Cannon and Morandi Cecchi [111]. It consists in approximating a function $f(\theta)$ by the linear combination

$$f^*(\theta) = c_0\phi_0(\theta) + \cdots + c_k\phi_k(\theta)$$

where the ϕ_i's are given functions. The constants c_i's are determined such that the system $f^*(\theta_i) = f(\theta_i)$, $i = 0, \ldots, m-1$ is satisfied in the least squares sense. It is an underdetermined problem which becomes overdetermined by adding the tangential and normal derivatives at the points θ_i. The solution of this problem was obtained by Morandi Cecchi and Redivo Zaglia [325] with the help of the E-algorithm. It compares very well with the other methods for the accuracy if the algorithm is totally reapplied on each test point. If only few points are needed, the accuracy obtained by the E-algorithm is greater than for the other methods.

The use of the E-algorithm in partial differential equations seems to be promising but it needs further studies.

6.5 Interpolation and approximation

It is not the purpose of this book to discuss the questions of interpolation and approximation. However since the solution of these problems can make use of some of the algorithms described above we shall now give, for the sake of completeness, a list of such applications with references.

The E and MNA algorithms play a central rôle in interpolation, extrapolation and rational approximation problems. First they were used by Håvie [221, 220, 222] and by Håvie and Powell [223] for interpolation by generalized rational functions. The MNA-algorithm was extended by Loi and McInnes [294] for generalized rational interpolation and Loi [291, 292] showed how to jump over an isolated singularity in order to avoid division by zero and breakdown of the algorithm and also

how to use it for computing separately the numerators and the denominators of the classical Padé approximants. An algorithm, based on the MNA-algorithm, is given by Loi and McInnes [295] for constructing the quadratic approximants of Shafer [390].

The case of multivariate interpolation by Neville-Aitken type formulæ is treated by Gasca and Lebron [174], Mühlbach [330] and Gasca and Mühlbach [175]. In a series of works, Cuyt [126, 127], Cuyt and Verdonk [129, 130], and Verdonk [449] show that the E-algorithm can be used in multivariate rational Hermite interpolation since these functions can be expressed as ratios of determinants similar to those defining the E-algorithm. A number of interesting particular cases also fit into this framework such as univariate rational interpolants, univariate and multivariate Padé approximants, and partial Padé approximants due to Brezinski [80] which, as shown by Prévost [364], can be obtained by the E-algorithm.

Vector Padé approximants, which are rational functions with a common denominator approximating simultaneously several power series in the sense of Padé, were introduced by van Iseghem [442]. They are related to vector orthogonal polynomials which can be recursively computed by an algorithm, due to van Iseghem [445], which is similar to the MNA-algorithm. Vector Padé approximants can also be obtained by the RPA and the CRPA and they can be used for accelerating the convergence of vector sequences as done by van Iseghem [443].

Finally let us mention that the product $P_k(x)V_{k+1}(x)$, where P_k and V_{k+1} are respectively the orthogonal polynomial of degree k and the Stieltjes polynomial of degree $k+1$ with respect to an arbitrary definite functional, can be recursively computed by the E-algorithm as stated by Brezinski [74]. These polynomials are useful in Kronrod's method for estimating the error in Gaussian quadrature methods, a procedure extended to Padé approximation by Brezinski [76].

Among the interesting questions about the approximation of functions is that of the computation of the poles and zeros of meromorphic functions. Let f be a meromorphic function for $|z| < R$, $0 < R \leq +\infty$. We have, for z sufficient small

$$f(z) = c_0 + c_1 z + c_2 z^2 + \cdots$$

Let $p_1, p_2, \ldots, p_N$ be the poles of f counted with their proper multiplicity ($0 \leq N \leq +\infty$) and such that

$$0 < |p_1| \leq |p_2| \leq \cdots < R.$$

It is well known that the qd-algorithm (see section 2.4) provides sequences tending to the p_k's under some assumptions, see Henrici [226]. The speed of convergence of these sequences was investigated by Prévost [363] who also showed how to use the ε-algorithm for treating this problem. Let us give his results.

Theorem 6.28

If the qd-algorithm is applied to the sequence (c_n) *(that is with* $q_1^{(n)} = c_{n+1}/c_n$ *for* $n = 0, 1, \ldots$*) and if, for* $k < N - 1$

$$|p_{k-1}| < |p_k| < |p_{k+1}|, \qquad (\textit{with } p_0 = 0)$$

then $\lim_{n\to\infty} q_k^{(n)} = p_k^{-1}$.

Moreover if, for $k < N - 2$

$$|p_{k-2}| < |p_{k-1}| < |p_k| < |p_{k+1}| < |p_{k+2}| \ (\textit{or } 0 < |p_1| < |p_2| < |p_3| \textit{ for } k = 1)$$

and $|p_k/p_{k+1}| \neq |p_{k-1}/p_k|$ *then*

$$\lim_{n\to\infty} \left(q_k^{(n+1)} - p_k^{-1}\right) \Big/ \left(q_k^{(n)} - p_k^{-1}\right) = a_k$$

with $a_k = p_k/p_{k+1}$ *if* $|p_k/p_{k+1}| > |p_{k-1}/p_k|$ *and* $a_k = p_{k-1}/p_k$ *otherwise.*

Due to the connection between the qd and rs-algorithms (see section 2.4) the same results are valid for the sequences $\left(r_k^{(n+1)} \Big/ r_k^{(n)}\right)$ obtained by this algorithm with $r_1^{(n)} = c_n$ for $n = 0, 1, \ldots$

Due to the connection between the qd and ε-algorithms the same results still hold (with the supplementary assumption $|p_i| \neq 1$ for $i = 1, \ldots, k-1$) for the sequences $\left(\varepsilon_{2k-2}^{(n+1)} \Big/ \varepsilon_{2k-2}^{(n)}\right)$ and $\left(\varepsilon_{2k-1}^{(n)} \Big/ \varepsilon_{2k-1}^{(n+1)}\right)$ obtained from this algorithm with $\varepsilon_0^{(n)} = c_n$ for $n = 0, 1, \ldots$

Since the convergence of the sequences converging to p_k^{-1} obtained by these three algorithms is linear they can be accelerated by Aitken's Δ^2 process.

If we want to compute the zeros of f we set

$$g(z) = 1/f(z)$$

and apply the previous algorithms to the computation of the poles of g which are the zeros of f. For that purpose the progressive form of the ε-algorithm is appropriate. Such a process can be, in particular, used to obtain the zeros of a polynomial f.

It is possible to compute simultaneously the poles of several meromorphic functions by an extension of the qd-algorithm to the vector case due to van Iseghem [446].

6.6 Statistics

There are many connections between statistics and the methods proposed in the previous sections. For example the link with least squares is well known. In section 3.6 we saw how to compute recursively the multiple correlation coefficient and, conversely, how this coefficient can be used for an automatic selection between several sequence transformations. Slowly convergent and even divergent series occur in statistics in the computation of the moments of estimators. For summing these series it is possible to use extrapolation methods or continued fractions as done by Bowman and Shenton [31, 32]. These methods proved to be very effective. As we shall see below, the ratio of determinants involved in the jackknife statistics is exactly the ratio defining the E-algorithm and thus this algorithm provides a recursive method for its implementation. The determination of the parameters in an ARMA model makes use of Hankel determinants which are the same as those appearing in the rs and ε-algorithms and thus these algorithms can be used for solving the problem. One can also think about accelerating the convergence of Monte-Carlo methods by extrapolation, a possibility never studied before.

On the other hand, it is also possible to use some techniques developed in statistics (in linear filtering and time series analysis, for example) to built new methods for predicting the unknown limit of a sequence and thus accelerating its convergence. An account of such procedures is given in chapter 12 of Wimp [467] based on a previous work by Higgins [228]. These possibilities have not yet been fully exploited.

6.6.1 The jackknife

The jackknife is a well known statistical procedure for bias reduction. It was described by Gray and Schucany [199].

Let $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_{k+1}$ be $k+1$ estimators of θ such that

$$E\left[\hat{\theta}_j - \theta\right] = \sum_{i=1}^{\infty} a_{ij} b_i(\theta), \qquad \text{for } j = 1, \ldots, k+1$$

where the a_{ij}'s are given and the b_i's are unknown functions of θ ($E[\cdot]$ denotes the expectation that is the mean value).

The jackknife consists in considering the following ratio of determinants

$$G\left(\hat{\theta}_1, \ldots, \hat{\theta}_{k+1}; a_{ij}\right) = \frac{\begin{vmatrix} \hat{\theta}_1 & \hat{\theta}_2 & \cdots & \hat{\theta}_{k+1} \\ a_{11} & a_{12} & \cdots & a_{1,k+1} \\ \vdots & \vdots & & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{k,k+1} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ a_{11} & a_{12} & \cdots & a_{1,k+1} \\ \vdots & \vdots & & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{k,k+1} \end{vmatrix}}.$$

$G\left(\hat{\theta}_1, \ldots, \hat{\theta}_{k+1}; a_{ij}\right)$ is called the k-th order generalized jackknife. If $b_i(\theta) = 0$ for $i > k$, then taking the expectation of both sides, shows that $G\left(\hat{\theta}_1, \ldots, \hat{\theta}_{k+1}; a_{ij}\right)$ is an unbiased estimator of θ. If it is not the case, the jackknife produces an estimator of lower order bias.

Of course the ratio in the definition of $G\left(\hat{\theta}_1, \ldots, \hat{\theta}_{k+1}; a_{ij}\right)$ is exactly the ratio considered in the E-algorithm and applying the E-algorithm with the initial values

$$E_0^{(n)} = \hat{\theta}_n \quad \text{and} \quad g_{0,i}^{(n)} = a_{in} \quad \text{for } n, i = 1, 2, \ldots,$$

gives

$$E_k^{(n)} = G\left(\hat{\theta}_n, \ldots, \hat{\theta}_{n+k}; a_{in}\right), \qquad \text{for } k, n = 1, 2, \ldots$$

Thus the E-algorithm provides a simple and efficient method for implementing the jackknife. If $a_{ij} = x_j^i$ then Richardson extrapolation method can be used while the case $a_{ij} = c_{i+j-1}$ can be implemented by

the G-transform. Another possibility for computing recursively $G\left(\hat{\theta}_1, \ldots, \hat{\theta}_{k+1}; a_{ij}\right)$ for increasing values of k is to use the bordering method as described in section 1.8.

The properties of the E-algorithm given in section 2.1 can be translated into properties for the jackknife. In particular, we immediately have, from theorems 2.1 and 2.2

Theorem 6.29

If $E\left[\hat{\theta}_j - \theta\right] = \sum_{i=1}^{\infty} a_{ij} b_i(\theta)$ *for* $j = 1, 2, \ldots$ *then, for all* n $E\left[G\left(\hat{\theta}_n, \ldots, \hat{\theta}_{n+k}; a_{in}\right)\right] = \theta + b_{k+1}(\theta) g_{k,k+1}^{(n)} + b_{k+2}(\theta) g_{k,k+2}^{(n)} + \cdots$.

If $b_i(\theta) = 0$ *for* $i > k$, *then* $E\left[G\left(\hat{\theta}_n, \ldots, \hat{\theta}_{n+k}; a_{in}\right)\right] = \theta$.

If we set

$$\varepsilon_n = \sum_{i=k+1}^{\infty} a_{in} b_i(\theta)$$

then we have

$$E\left[G\left(\hat{\theta}_n, \ldots, \hat{\theta}_{n+k}; a_{in}\right)\right] = \theta + \overline{E}_k^{(n)}$$

where $\overline{E}_k^{(n)}$ denotes the quantity obtained by applying the E-algorithm to $\overline{E}_0^{(n)} = \varepsilon_n$ and $g_{0,i}^{(n)} = a_{in}$. Thus we obtained, as a by-product of the E-algorithm, some results which were proved for the jackknife by other techniques, see Gray [194]. These questions are developed in Brezinski [90].

In practice the estimators $\hat{\theta}_i$ are obtained by deleting some values from the sample as explained by Efron [150].

6.6.2 ARMA models

A time series is a sequence of observations ordered in time. It can be considered as realizations of a stochastic process X_t whose value, which depends on the time t, is a random variable. If the observations are made at equally spaced intervals of time we shall denote them by X_i for $i \in \mathbf{Z}$.

(X_i) is said to be a second order stationary process if

$$\forall i \in \mathbf{Z}, \quad E[X_i] = 0$$
$$\forall i, j \in \mathbf{Z}, \quad E[X_i X_j] \text{ exists and depends only on } i - j$$

(where E designates the mean value that is the expectation operator).

Let us assume that $E[X_0^2] \neq 0$. We define the autocorrelation function of the process by, $\forall i \in \mathbf{Z}$

$$\varrho_i = E[X_0 X_i] \Big/ E\left[X_0^2\right].$$

A process such that $\varrho_0 = 1$ and $\forall i \neq 0$, $\varrho_i = 0$ is called a white noise. A second order stationary process is said to be an autoregressive moving average process of order (p, q) if, $\forall i \in \mathbf{Z}$

$$X_i + \sum_{j=1}^{p} a_j X_{i-j} = \varepsilon_i + \sum_{j=1}^{q} b_j \varepsilon_{i-j}$$

where (ε_i) is a white noise. We shall speak of an ARMA(p, q) model. This model is said to be minimal if the polynomials

$$A(z) = 1 + \sum_{i=1}^{p} a_i z^i \quad \text{and} \quad B(z) = 1 + \sum_{i=1}^{q} b_i z^i$$

have no common zero and if their zeros are outside the unit disc.

An important problem in statistics is to find the values p and q of the minimal ARMA(p, q) model of (X_i).

As proved by Beguin, Gouriéroux and Monfort [18] a second order stationary process has a minimal ARMA(p, q) representation if and only if

$$\forall n \geq q - p + 1 \quad \sum_{i=0}^{p} a_i \varrho_{n+i} = 0 \qquad \text{with } a_0 \neq 0 \text{ and } a_p = 1$$
$$\text{and} \quad \sum_{i=0}^{p} a_i \varrho_{q-p+i} \neq 0.$$

Of course, due to the properties of Hankel determinants, this is equivalent to

$$H_k(\varrho_{n-k+1}) = 0 \quad \forall n \geq q + 1 \text{ and } \forall k \geq p + 1$$
$$H_p(\varrho_{n-p+1}) \neq 0 \quad \forall n \geq q$$
$$H_k(\varrho_{q-k+1}) \neq 0 \quad \forall k \geq p.$$

Thus, in order to find the values of p and q of the ARMA model, one can compute the Hankel determinants by their recurrence relation and check if they are zero or not.

Another possibility, which was studied by Berlinet [27], is to apply the rs-algorithm or the ε-algorithm to the sequence (ϱ_i). Since the quantities computed by these algorithms can be expressed as ratios of determinants whose numerators are precisely the determinants whose equality to zero is to be checked we immediately have the

Theorem 6.30

A second order stationary process admits a minimal ARMA(p,q) *representation if and only if one of the following equivalent conditions is satisfied*

i) $\forall n \geq q-p+1,\ s_p^{(n)} = c \neq 0$ *and* $s_p^{(q-p)} \neq c$.

ii) $\forall n \geq q-p+1,\ r_{p+1}^{(n)} = 0$ *and* $r_{p+1}^{(q-p)} \neq 0$.

iii) $\forall n \leq -q-p,\ s_p^{(n)} = c \neq 0$ *and* $s_p^{(-q-p+1)} \neq c$.

iv) $\forall n \leq -q-p-1,\ r_{p+1}^{(n)} = 0$ *and* $r_{p+1}^{(-q-p)} \neq 0$.

v) $\forall n \geq q-p+1,\ \varepsilon_{2p}^{(n)} = 0$ *and* $\varepsilon_{2p}^{(q-p)} \neq 0$.

The case of multivariate ARMA models was treated by Berlinet [26] by using the matrix and vector ε-algorithms.

6.6.3 Monte-Carlo methods

Monte-Carlo methods were introduced by S. M. Ulam around 1945. They were developed by J. von Neumann and N. Metropolis when working on the neutron transport equation in Los Alamos. The early history of this method is given in Grant Cooper [191]. It consists in solving a deterministic mathematical problem by means of a probabilistic analogy using random numbers. Monte-Carlo methods have very many applications in physical problems when modelling is too difficult. We refer the interested reader to the classical treatise of Hammersley and Handscomb [212] and to Novak [339] where more recent references can be found. Monte-Carlo methods also have applications in numerical analysis and, in particular, for optimization and numerical integration. To

illustrate the process we shall describe how they can be used for computing an approximate value of a definite integral. Of course, since there exist many very powerful quadrature methods, the use of a Monte-Carlo method is not justified in this case and it will only serve as an illustration. However, for computing multiple integrals, Monte-Carlo methods become an attractive tool as explained by Lambert [271].

Let us assume that we want to compute an approximate value of

$$I = \int_0^1 f(x)\,dx.$$

Let x be a random variable uniformly distributed in $[0,1]$. We consider the random variable y defined by

$$y = f(x).$$

The mean value of y is

$$E[y] = \int_0^1 f(x)\,dx = I$$

and its variance (the square of the standard deviation σ) is

$$\sigma^2 = \mathrm{var}(y) = E\left[(y-I)^2\right] = \int_0^1 f^2(x)\,dx - I^2.$$

Let $x_1, \ldots, x_n$ be n realizations of the random variable x, that is n random numbers uniformly distributed in $[0,1]$ which are obtained, in practice, by using a random number generator. We set $y_i = f(x_i)$ for $i = 1, \ldots, n$. An estimator for $E[y]$ is given by the arithmetical mean $\overline{y}$

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

Of course, since the y_i's are random variables, $\overline{y}$ is also a random variable and it is easy to see that $E[\overline{y}] = I$ and that $\mathrm{var}(\overline{y}) = \sigma^2/n$.

In order to check whether or not $\overline{y}$ is a good estimation of I we can make use of the inequality of Bienaymé-Chebyshev which says that

$$\Pr\left(|\overline{y} - I| > a\,\sigma/\sqrt{n}\right) < 1/a^2$$

where $\Pr(u)$ designates the probability for the event u to be true. This inequality tells us that the probability that $\overline{y}$ does not belong to the interval $[I - a\,\sigma/\sqrt{n}\,, I + a\,\sigma/\sqrt{n}]$ is smaller than $1/a^2$. Thus, when n is large, this probability is quite small and $\overline{y}$ is a good estimation of I with a high probability.

Before giving an example let us mention a computational problem due to the computer's arithmetic. For obtaining $\overline{y}$ we shall have to compute $y_1 + y_2 + \cdots + y_n$ for a large value of n. It is well known that, due to the possible cancellation errors, such a computation is numerically unstable.

A procedure for correcting the error due to the computer's arithmetic in such a sum was proposed by Pichat [358]. It will be described in section 7.2.

Let us now give a numerical example. The random number generator was taken from Neelamkavil [333].

We want to compute

$$I = \int_0^1 x \sin x^{-1}\, dx = \int_1^\infty x^{-3} \sin x\, dx.$$

We have

$$\int x^{-3} \sin x\, dx = -\frac{1}{2x} \cdot \left(\cos x + \frac{\sin x}{x}\right) - \frac{1}{2} \int \frac{\sin x}{x}\, dx$$

and thus

$$I = \int_1^\infty x^{-3} \sin x\, dx = \frac{1}{2} \cdot (\sin 1 + \cos 1) - \frac{1}{2} \cdot \left(\frac{\pi}{2} - \mathrm{Si}(1)\right)$$

where $\mathrm{Si}(t)$ is the sine integral defined by

$$\mathrm{Si}(t) = \int_0^t x^{-1} \sin x\, dx.$$

Taking the value of $\mathrm{Si}(1) = 0.9460830704$ from Abramowitz and Stegun [4] we obtain

$$I \simeq 0.3785300171.$$

Let S_n be the result obtained by the Monte-Carlo method described above with a sample of size k_n, that is

$$S_n = \frac{1}{k_n} \sum_{i=1}^{k_n} f(x_i)$$

where the x_i's are uniformly distributed in $[0,1]$.

By the Bienaymé-Chebyshev inequality we know that

$$\Pr\left(S_n \in \left[I - \frac{a\sigma}{\sqrt{k_n}}, I + \frac{a\sigma}{\sqrt{k_n}}\right]\right) > 1 - \frac{1}{a^2}.$$

For accelerating the convergence of the sequence (S_n) we shall use the transformation studied in section 3.4 given by

$$T_n = S_n - \frac{\Delta S_n}{\Delta g_n} \cdot g_n, \qquad n = 0, 1, \ldots$$

It is quite easy to prove that

$$\Pr\left(T_n \in [I - a\sigma h_n, I + a\sigma h_n]\right) > 1 - \frac{1}{a^2}$$

with $h_n = \left(\frac{g_{n+1}}{\sqrt{k_n}} + \frac{g_n}{\sqrt{k_{n+1}}}\right) \Big/ |g_{n+1} - g_n|$.

The question is to know whether or not $\forall n, h_n < 1/\sqrt{k_n}$. The answer to this question depends on the choice of (g_n). It seems suitable to choose $g_n = k_n^{-1/2}$. In that case the preceding inequality is satisfied if $k_{n+1} > 9\,k_n$. We shall also try $g'_n = k_n^{-1}$ for which the inequality is satisfied if $k_{n+1} > 4\,k_n$ and $g''_n = k_n^{-3/2}$ for which the inequality is satisfied if $k_{n+1} \geq 3\,k_n$.

We obtain the following results (number of exact digits) with $k_{n+1} = 5k_n$

n	S_n	T_n with g_n	T_n with g'_n	T_n with g''_n
10	1.49	1.04	1.28	1.37
50	1.45	0.78	0.90	0.94
250	0.97	1.39	2.26	1.75
1250	1.58	1.60	1.60	1.59
6250	1.59	2.12	2.58	2.27
31250	2.14	2.35	3.08	3.84
156250	3.11	3.10	3.51	3.75
781250	4.03			

In these results the conditions on (k_n) are satisfied only for (g'_n) and (g''_n) and we see that, for n large enough, we obtain a better accuracy with (g'_n) and (g''_n) but not with (g_n). Let us now take $k_{n+1} = 10\,k_n$ which satisfies the conditions in all the cases.

n	S_n	T_n with g_n	T_n with g'_n	T_n with g''_n
10	1.49	0.75	0.89	0.93
100	0.95	1.53	1.31	1.27
1000	1.26	2.13	1.72	1.67
10000	1.65	2.17	3.54	2.73
100000	2.60	3.37	3.55	3.36
1000000	3.30			

The results obtained with T_n are better (for n large enough) than those obtained by S_n but not than those obtained by S_{n+1}. With $g_n = k_n^{-1/2}$ the condition $h_n < 1 \Big/ \sqrt{k_{n+1}}$ is satisfied if $\sqrt{k_{n+1}} < -\sqrt{k_n}$ which is obviously impossible.

For this example $\sigma = 1.0843656$ which explains why the results obtained are not very decisive for the values of k_n considered.

The generator of pseudo-random numbers used was that described by Neelamkavil [333] which is reported to have been successfully implemented on 8-bit and 16-bit microcomputers. It consists in generating the sequence

$$r_0 = 13$$
$$r_{n+1} = 5^{13} r_n \ (\text{mod. } 2^{35})$$
$$x_{n+1} = r_{n+1} / 2^{35} .$$

6.7 Integration and differentiation

Numerical methods for the computation of definite integrals depend on the number of points in the domain of integration. Thus they produce a sequence of approximations converging to the exact value of the integral. The same is true in methods for numerical differentiation. Extrapolation processes are often very useful for accelerating the convergence of such methods. Historically it is well known that Romberg's method for accelerating the trapezoidal rule was the starting point for further developments and the rebirth of extrapolation techniques. The numerical methods for the computation of Cauchy's principal values and improper integrals depend on a parameter tending to zero or infinity. In these cases, extrapolation by continuous prediction algorithms can be much useful. Acceleration methods can also be used for deriving new (nonlinear) quadrature formulæ.

6.7.1 Acceleration of quadrature formulæ

The trapezoidal rule is a method for computing approximate values of the definite integral

$$I = \int_a^b f(x)dx.$$

It consists in computing

$$T_n = T(h) = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(a+ih) + f(b)\right]$$

where $h = (b-a)/n$.

If f is continuous in $[a,b]$ then the sequence (T_n) converges to I when n tends to infinity. Moreover if f is $2k+2$ times continuously differentiable in $[a,b]$ then, by the Euler-Maclaurin formula, we have

$$T(h) = I + \sum_{i=1}^{k} a_i h^{2i} + R_k(h)$$

where $R_k(h) = O\left(h^{2k+2}\right)$.

If, for some function f, $R_k(h)$ is identically zero, then $T(h)$ is a polynomial of degree k in h^2 and I is the constant term of this polynomial (that is its value at the point zero). This constant term can be computed by interpolation without knowing the coefficients a_i's as follows. First we compute $T(h_n)$ for $n = 0, 1, \ldots, k$ and then the value at the point zero of the interpolation polynomial passing through the points $(h_n^2, T(h_n))$ for $n = 0, 1, \ldots, k$. Let us denote by $T_k^{(0)}$ this value. If $R_k(h)$ is identically zero then $T_k^{(0)}$ is equal to I. If $R_k(h)$ is not identically zero then $T_k^{(0)}$ is a much better approximation of I than $T(h_0), \ldots, T(h_k)$. This is the idea behind the well known method due to Romberg [375]. It became really widely used after its rigorous error analysis by Bauer [15] and the study of the convergence of the sequence $\left(T_k^{(0)}\right)$ to I when k tends to infinity by Laurent [274]. In particular Laurent proved that the sequence (h_n) cannot be chosen arbitrarily for insuring the convergence but that we must have for all n, $h_n/h_{n+1} \geq \alpha > 1$. This is the so-called condition (α) which is satisfied, for example, by taking $h_{n+1} = h_n/2$ which was the choice made by Romberg for reducing the number of evaluations of the function f. The values of the preceding interpolation polynomials at

the point zero can be recursively computed by the Neville-Aitken scheme (that is by Richardson extrapolation scheme as described in section 2.2) which becomes since $h_n = h_0/2^n$ the well known Romberg's method

$$\begin{aligned} T_0^{(n)} &= T\left(h_0/2^n\right), & n &= 0,1,\ldots \\ T_{k+1}^{(n)} &= \frac{4^{k+1}T_k^{(n+1)} - T_k^{(n)}}{4^{k+1}-1}, & k,n &= 0,1,\ldots \end{aligned}$$

This method was derived heuristically by Romberg and its interpretation by extrapolation at zero by polynomials in h^2 was only given later by Laurent [275] in his thesis. We have the

Theorem 6.31

i) $\forall k$ *fixed,* $\lim_{n\to\infty} T_k^{(n)} = I.$

ii) $\forall n$ *fixed,* $\lim_{k\to\infty} T_k^{(n)} = I.$

iii) $\forall k$ *fixed,* $T_k^{(n)} - I = O\left(h_n^{2k+2}\right)$ *when* $n \to \infty$ *and* $\lim_{n\to\infty} \left(T_k^{(n)} - I\right) \Big/ \left(T_{k-1}^{(n)} - I\right) = 0.$

Of course if f is $2m+2$ times continuously differentiable in $[a,b]$ the results *iii)* are valid only for $k \leq m$. The results *i)* and *ii)* are true for any continuous function in $[a,b]$ (by theorems 2.16 and 2.17).

Let us apply Romberg's method to the computation of

$$I = \int_0^1 \frac{dx}{x+0.01} = 4.615120516841260\ldots$$

with $h_0 = 1/3$.
We obtain (number of exact digits that is $-\log_{10}\left(\left|T_k^{(n)} - I\right| \Big/ I\right)$ which explains that some values can be less than one and even negative when there is no exact digit)

$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
−0.47								
	0.13							
−0.11		0.62						
	0.57		1.17					
0.28		1.16		1.86				
	1.08		1.85		2.74			
0.71		1.83		2.74		3.89		
	1.72		2.73		3.89		5.37	
1.20		2.69		3.89		5.37		7.22
	2.50		3.87		5.37		7.22	
1.74		3.78		5.36		7.22		
	3.45		5.31		7.21			
2.32		5.13		7.18				
	4.54		7.07					
2.91		6.70						
	5.70							
3.51								

Since each column converges faster than the preceding one, we can apply the procedure described in section 3.8 for controlling the error. Let us give the intervals with endpoints $V_k^{(n)}(-b)$ and $V_k^{(n)}(b)$ where

$$V_k^{(n)}(b) = T_k^{(n)} - b\left(T_k^{(n)} - T_{k-1}^{(n)}\right).$$

For $k = 8$ and $n = 0$ we have $T_8^{(0)} = 4.615120792632863$ and we obtain for various values of b the following intervals

$$\begin{aligned}
b &= 0.5 \quad [\,4.615113048839706, 4.615128536426023] \\
b &= 0.3 \quad [\,4.615114984787995, 4.615126604777330] \\
b &= 0.1 \quad [\,4.615118856684575, 4.615122728581154] \\
b &= 0.05 \quad [\,4.615119824658719, 4.615121760607010] \\
b &= 0.03 \quad [\,4.615120211848377, 4.615121373417352]
\end{aligned}$$

For smaller values of b, I does not belong to the interval.

Instead of extrapolating at zero by polynomials in h^2, the trapezoidal rule can be accelerated by extrapolating at infinity by rational functions in $1/h^2$. This can be done by the ϱ-algorithm with $x_n = 1/h_n^2$ as suggested by Brezinski [34] who proved that it has a slight advantage

over Romberg's method. For the same numerical example we obtain

$$\begin{array}{llll}
\varrho_2^{(0)}=5.58 & & & \\
\varrho_2^{(1)}=4.89 & \varrho_4^{(0)}=4.65 & & \\
\varrho_2^{(2)}=4.67 & \varrho_4^{(1)}=4.6199 & \varrho_6^{(0)}=4.61537 & \\
\varrho_2^{(3)}=4.62 & \varrho_4^{(2)}=4.6154 & \varrho_6^{(1)}=4.6151273 & \varrho_8^{(0)}=4.615120586 \\
\varrho_2^{(4)}=4.6157 & \varrho_4^{(3)}=4.6151293 & \varrho_6^{(2)}=4.615120593 & \\
\varrho_2^{(5)}=4.615155 & \varrho_4^{(4)}=4.6151206 & & \\
\varrho_2^{(6)}=4.6151212 & & &
\end{array}$$

Rational extrapolation of the trapezoidal was also considered by Bulirsch and Stoer [108].

When the function f to be integrated has singularities in the interval of integration then the Euler-Maclaurin formula no more holds and thus the error $T(h)-I$ of the trapezoidal rule no more has a series expansion in h^2. In that case it is first necessary to obtain the asymptotic expansion of the error and then to find a suitable algorithm for extrapolation which eliminates progressively the successive terms in this expansion. Such a technique was used by Håvie [216, 217, 218] in several situations. Of course all these extrapolation algorithms, based on an asymptotic expansion of the error, are particular cases of the E-algorithm (see section 2.1) and they led Schneider [388] and Håvie [219] to their derivation of this algorithm. This is also the case of the methods derived by Fox [165] and J. A. Shanks [393, 394]. There is a wide literature on Romberg's method, which is now classical and is discussed in any textbook of numerical analysis without giving reference to Romberg's original paper (a proof of genuine celebrity), and we shall not enter here into all the details and variants. The interested reader is referred to the survey paper by Joyce [253] or to Piessens et al. [360] or to Lyness [304].

It was showed by Kahaner [254] that when the integrand f has an integrable endpoint singularity of algebraic or logarithmic type (that is, for example, $f(x)=\sqrt{x}\,g(x)$ or $f(x)=x^{\alpha}(1-x)^{\beta}g(x)$ with $\alpha,\beta>-1$ and g smooth enough) then extrapolation by the ε-algorithm provides better results than Romberg's unmodified method but that it requires more function evaluations than a properly modified Romberg scheme. This shows again the superiority of the E-algorithm (which was not yet found at that time) in which the successive terms of the asymptotic expansion of the error can be introduced when they are known. In the

case of an algebraic endpoint singularity we have

$$T(h) - I = a_1 h^{\alpha_1} + a_2 h^{\alpha_2} + \cdots \quad \text{with} \quad 0 < \alpha_1 < \alpha_2 < \cdots$$

If the α_i's are known, the E-algorithm can be used with $g_i(n) = h_n^{\alpha_i}$. For $h_n = h_0/2^n$ error bounds were obtained by Walz [452], see also Walz [451]. If the α_i's are not known the procedure of Christiansen and Petersen [114] described in section 2.2 can be used or that of Werner and Wuytack [461].

For a logarithmic endpoint singularity we have

$$T(h) - I = (a_1 + b_1 \ln h)\, h^{\alpha+1} + (a_2 + b_2 \ln h)\, h^{\alpha+2} + \cdots$$

and the E-algorithm can be used if α is known. When α or the α_i's are unknown then the ε-algorithm provides an interesting alternative.

When the weight function has algebraic and logarithmic endpoint singularities, that is when computing

$$I = \int_0^1 f(x)(1-x)^{\alpha} x^{\beta} (-\ln x)^{\nu}\, dx \quad \text{with} \quad \alpha + \nu > -1\,, \beta > -1$$

then Levin's transformation can be used as showed by Sidi [396]. Some cases can be treated by Romberg's method after a change of variable, see Hunter [231].

In Romberg's scheme as described above, $T_1^{(n)}$ is exactly Simpson's rule and thus the subsequent columns in the table accelerate this rule. Similarly other quadrature methods can be accelerated by extrapolation procedures as the rectangular rule, see Laurent [275].

Let us now discuss the possible extrapolation of Gaussian quadrature methods. For such methods the asymptotic expansion of the error is usually unknown and thus the E-algorithm cannot be used. The only information available has the form

$$S_n - I = O(g_n)$$

where S_n is the approximation of

$$I = \int_a^b f(x)\, w(x)\, dx$$

obtained by the Gaussian quadrature formula with $n+1$ points corresponding to the weight function w on $[a, b]$ and where g_n is a known

function of n. Thus we shall try to extrapolate the sequence (S_n) by the method described in section 3.4 which is in fact the first column $\left(E_1^{(n)}\right)$ of the E-algorithm, namely

$$T_n = S_n - \frac{\Delta S_n}{\Delta g_n} \cdot g_n\,, \qquad n = 0, 1, \ldots$$

Let us, for example, consider the computation of

$$I = \int_{-1}^{1} \frac{e^{2x}}{\sqrt{1-x^2}}\,dx = \pi I_0(2).$$

We have $I_0(2) = J_0(2i)$ whose value can be obtained by the algorithm given by Moshier [328] which achieves a precision of $8.6 \cdot 10^{-8}$. We have

$$I = 7.1615284390499388380.$$

By the IMSL library subroutines we have $I = 7.1615284390502566548$.

If we set $w(x) = (1-x^2)^{-1/2}$ then the Gauss-Chebyshev quadrature method can be used, that is

$$S_n = \sum_{i=0}^{n} A_i f(x_i)$$

with $A_i = \dfrac{\pi}{n+1}$ and $x_i = \cos\dfrac{2i+1}{2n+2} \cdot \pi$.

It can be proved that

$$S_n - I = \frac{2\pi f^{(2n+2)}(\xi_n)}{4^{n+1}(2n+2)!} \quad \text{with} \quad \xi_n \in [-1, +1].$$

If we take $g_n = \dfrac{1}{4^{n+1}(2n+2)!}$ then $\lim\limits_{n\to\infty} g_{n+1}/g_n = 0$. Moreover

$$2^i e^{-2} \le f^{(i)}(\xi) \le 2^i e^2\,, \qquad \forall \xi \in [-1, +1]$$

and thus

$$4e^{-4} \le f^{(2n+4)}(\xi_{n+1}) \Big/ f^{(2n+2)}(\xi_n) \le 4e^4$$

which shows that the conditions of theorem 3.14 *i)* are satisfied and it follows that (T_n) will converge to I faster than (S_n). It is impossible to check whether the condition of theorem 3.14 *iii)* is true or not (and, in general, it is not) and we obtain the following results

n	S_n	T_n
1	6.842965459611574	7.151477124036244
2	7.151474267446758	7.161354399030198
3	7.161354398779038	7.161526543435953
4	7.161526543435945	7.161528424887133
5	7.161528424887133	7.161528438973225
6	7.161528438973225	7.161528439049938
7	7.161528439049938	7.161528439050256
8	7.161528439050256	

The subsequent results are equal to the last values which seems to confirm that Moshier's algorithm does not achieve full accuracy.

6.7.2 Nonlinear quadrature formulæ

In section 6.4.3 we gave nonlinear methods, based on the confluent form of the ϱ-algorithm or on Padé approximants, for integrating differential equations.

Of course the problem of computing a definite integral

$$I = \int_a^b f(t)\,dt$$

can be turned into the initial value problem

$$\begin{aligned} y'(x) &= f(x) \\ y(a) &= 0 \end{aligned}$$

since $y(x) = \int_a^x f(t)\,dt$ and $y(b) = I$. Thus this problem can be solved by one of the methods described in section 6.4.3. We set $h = (b-a)/N$, $y_0 = 0$ and then we compute

$$y_{n+1} = y_n + \frac{2hf^2(a+nh)}{2f(a+nh) - h\,f'(a+nh)}$$

for $n = 0, \ldots, N-1$. y_N will be an approximation of I. In this formula $h\,f'(a+nh)$ can be replaced by the approximate value $f(a+(n+1)h) - f(a+nh)$. As showed by Wuytack [469], these two methods have an error proportional to h^2. Thus extrapolation can be used for improving the accuracy.

Let us give some numerical examples with the two preceding methods. These examples show that when f is a smooth function the classical linear methods, such as the trapezoidal rule or that of Simpson, give better results than the nonlinear ones. However when f has a pole near the interval of integration then the nonlinear methods provide better results. Let us first take

$$\int_0^1 \frac{e^x}{(3-e^x)^2}\,dx = 3.0496468\ldots$$

We obtain the following values

N	1^{st} method	2^{nd} method
4	3.1679644	−63.753354
14	3.0573798	76.984314
24	3.0519517	3.6495883
34	3.0507182	3.2750184
44	3.0502605	3.1694384

For $\int_0^1 \frac{3-e^x+xe^x}{(3-e^x)^2}\,dx = 3.5496468\ldots$ we have

N	1^{st} method	2^{nd} method
4	3.0308452	1.9979022
14	3.5070085	39.378940
24	3.5367515	4.1958131
34	3.5436232	3.7937413
44	3.5461874	3.6796189

The second method seems to suffer from instability.

6.7.3 Cauchy's principal values

Let us now assume that the integrand f has a singularity at point $c \in [a,b]$ but that

$$\lim_{\varepsilon\to 0+}\left(\int_a^{c-\varepsilon} f(x)\,dx + \int_{c+\varepsilon}^b f(x)\,dx\right)$$

exists. This limit is called the Cauchy's principal value of the integral and it is denoted by

$$I = v.p.\int_a^b f(x)\,dx.$$

For computing an approximate value of this integral it is possible to use the trapezoidal rule or the midpoint approximation if none of the points of the quadrature formula coincides with c or a suitable modification of these methods in the other case. Then the Euler-Maclaurin formula for the error can be adapted and Romberg's method can be modified accordingly for extrapolating the results thus leading to a particular case of the E-algorithm. This was the approach followed, for example, by Hunter [233], Lyness [302] and Håvie and Simonsen [224].

We shall now propose two new approaches to this problem. The first one consists in taking a sequence $\varepsilon_0 > \varepsilon_1 > \cdots > 0$ with $\lim_{n\to\infty} \varepsilon_n = 0$ and computing (by any quadrature rule) an approximation S_n of

$$I_n = \int_a^{c-\varepsilon_n} f(x)\,dx + \int_{c+\varepsilon_n}^b f(x)\,dx$$

and then to accelerate the convergence of (S_n) by any of the extrapolation methods studied above.

The second approach consists in setting $\varepsilon = 1/t$ and computing (by any quadrature rule) an approximation $g(t)$ of

$$I(t) = \int_a^{c-t^{-1}} f(x)\,dx + \int_{c+t^{-1}}^b f(x)\,dx$$

and then using the confluent form of any extrapolation algorithm (see chapter 5) to obtain an approximation of

$$I = \lim_{t\to\infty} I(t)$$

from $g(t)$ and the derivatives of $I(t)$.

These two procedures have not yet been studied from the theoretical point of view but let us illustrate them by a numerical example. We consider the principal value

$$I = v.p. \int_{-1}^1 \frac{e^x}{x}\,dx = Ei(1) + E_1(1).$$

Using the tables of Abramowitz and Stegun [4] we obtain to 10 digits

$$I = 2.114501750$$

and with the IMSL library subroutines we get

$$I = 2.114501750751457.$$

In the first procedure the two integrals in I_n have been computed by Romberg's method with a precision of $1.6 \cdot 10^{-9}$ at least. The sequence (S_n) thus obtained have been accelerated by Aitken's Δ^2 process, the first column of the ϱ-algorithm and linear extrapolation (that is we considered the sequences $\left(\varepsilon_2^{(n)}\right)$, $\left(\varrho_2^{(n)}\right)$ and $\left(T_n = S_n - \dfrac{\Delta S_n}{\Delta \varepsilon_n} \cdot \varepsilon_n\right)$). We get the following results (number of exact digits)

- with $\varepsilon_n = 0.7^{n+1}$, $n = 0, 1, \ldots$

n	S_n	$\varepsilon_2^{(n)}$	$\varrho_2^{(n)}$	T_n
0	0.17	1.63	0.37	1.66
1	0.33	2.08	0.51	2.13
2	0.49	2.53	0.65	2.60
3	0.64	2.99	0.80	3.06
4	0.80	3.45	0.96	3.53
5	0.95	3.92	1.11	3.99
6	1.11	4.38	1.26	4.46
7	1.26	4.85	1.42	4.92
8	1.42	5.31	1.57	5.39
9	1.57	5.78	1.73	5.85
10	1.73	6.24	1.88	6.32
11	1.88	6.70	2.04	6.78
12	2.04	7.17	2.19	7.25
13	2.19	7.63	2.35	7.71
14	2.35	8.10	2.50	8.17
15	2.50	8.56	2.66	8.64
16	2.66	9.03	2.81	9.10
17	2.81	9.49	2.97	9.57
18	2.97			

$\left(\varepsilon_2^{(n)}\right)$ and (T_n) converge faster than (S_{n+2}) but not $\left(\varrho_2^{(n)}\right)$.

- with $\varepsilon_n = \dfrac{1}{n+2}$, $n = 0, 1, \ldots$

n	S_n	$\varepsilon_2^{(n)}$	$\varrho_2^{(n)}$	T_n
0	0.32	0.80	2.68	2.13
1	0.50	0.92	3.07	2.59
2	0.62	1.02	3.36	2.93
3	0.72	1.10	3.61	3.19
4	0.80	1.17	3.81	3.41
5	0.87	1.23	3.98	3.60
6	0.93	1.28	4.14	3.76
7	0.98	1.32	4.28	3.91
8	1.02	1.37	4.40	4.04
9	1.07	1.40	4.51	4.16
10	1.10	1.44	4.62	4.27
11	1.14	1.47	4.72	4.37
12	1.17	1.50	4.81	4.46
13	1.20	1.53	4.89	4.55
14	1.23	1.56	4.97	4.63
15	1.25	1.58	5.04	4.71
16	1.28	1.60	5.11	4.78

$\left(\varrho_2^{(n)}\right)$ and (T_n) converge faster than (S_{n+2}) but not $\left(\varepsilon_2^{(n)}\right)$.

The second procedure described above is more difficult to use since it needs the computation of the derivatives of $I(t)$. We have

$$I'(t) = t^{-2}\left[f\left(c-t^{-1}\right)+f\left(c+t^{-1}\right)\right]$$

$$I''(t) = -2t^{-3}\left[f\left(c-t^{-1}\right)+f\left(c+t^{-1}\right)\right]+t^{-4}\left[f'\left(c-t^{-1}\right)-f'\left(c+t^{-1}\right)\right].$$

Using the confluent forms of Overholt's process, the ε-algorithm, the ϱ-algorithm and the Taylor expansion we have respectively

$$V_1(t) = I(t)-\frac{I'^2(t)}{I''(t)} = \varepsilon_2(t)$$

$$\varrho_2(t) = I(t)-\frac{2I'^2(t)}{I''(t)}$$

$$T_2(t) = I(t)-tI'(t)+\frac{t^2}{2}\cdot I''(t).$$

For the previous example we obtain by computing again $g(t)$ by the same procedure (number of exact digits)

t	$g(t)$	$V_1(t) = \varepsilon_2(t)$	$\varrho_2(t)$	$T_2(t)$
10	1.02	1.32	4.28	0.55
100	2.02	2.33	7.28	1.55
1000	3.02	3.33	10.28	2.55
10000	4.02	4.33	13.28	3.55
100000	5.02	5.33	16.41	4.55

Since the exact value of I was known to us only to 16 digits it was not possible to know if the precision of $\varrho_2(t)$ still increased but for the other methods we have

t	$g(t)$	$V_1(t) = \varepsilon_2(t)$	$T_2(t)$
1000000	6.02	6.33	5.55
10000000000	9.27	10.18	8.61

These results can be compared with those of Hunter [232] for the same example.

The sequences $(g(t_n))$, $(\varepsilon_2(t_n))$, $(\varrho_2(t_n))$ and $(T_2(t_n))$ can be accelerated by Aitken's Δ^2 process, the first column of the ϱ-algorithm and linear extrapolation with $\varepsilon_n = 1/t_n$ (denoted by (T'_n)).

With $t_n = 10^{n+1}$ for $n = 0, 1, \ldots$ we obtain (number of exact digits)

- Extrapolation of $(g(t_n))$

n	$g(t_n)$	$\varepsilon_2^{(n)}$	$\varrho_2^{(n)}$	T'_n
0	1.02	6.20	2.02	5.24
1	2.02	9.20	3.02	8.24
2	3.02	12.20	4.02	11.24
3	4.02	15.21	5.02	14.24
4	5.02	16.88	6.02	16.71

- Extrapolation of $(\varepsilon_2(t_n))$

n	$\varepsilon_2(t_n)$	$\varepsilon_2^{(n)}$	$\varrho_2^{(n)}$	T'_n
0	1.32	6.20	2.33	5.24
1	2.33	9.20	3.33	8.24
2	3.33	12.20	4.33	11.24
3	4.33	15.21	5.33	14.24
4	5.33	16.88	6.33	16.71

- Extrapolation of $(\varrho_2(t_n))$

n	$\varrho_2(t_n)$	$\varepsilon_2^{(n)}$	$\varrho_2^{(n)}$	T'_n
0	4.28	12.78	7.28	5.24
1	7.28	16.81	10.28	8.24
2	10.28	16.86	13.28	11.24
3	13.28	16.86	16.18	14.24
4	16.41	16.86	16.86	16.71

- Extrapolation of (T_n)

n	T_n	$\varepsilon_2^{(n)}$	$\varrho_2^{(n)}$	T'_n
0	0.55	5.20	1.55	4.24
1	1.55	8.20	2.55	7.24
2	2.55	11.20	3.55	10.24
3	3.55	14.20	4.55	13.24
4	4.55	17.13	5.55	16.15

All these methods seem to work quite well but they still need to be studied and justified from the theoretical point of view.

6.7.4 Infinite integrals

In this section we shall be concerned with the computation of infinite integrals of the form

$$I = \int_a^\infty f(x)\,dx.$$

Let us set

$$F(t) = \int_a^t f(x)\,dx\,, \qquad t \geq a.$$

Thus for t sufficiently large $F(t)$ is usually a good approximation of I. But, on the other hand, if t is large it is quite difficult to obtain a good numerical approximation of $F(t)$ by a quadrature rule. To avoid this drawback it is possible to use values of t quite close to a and then an extrapolation method. In the literature two such approaches can be found

- computation of $(S_n = F(t_n))$ where $t_0 < t_1 < t_2 < \cdots$ and $\lim_{n\to\infty} t_n = \infty$ and extrapolation by a scalar algorithm such as the ε-algorithm, the ϱ-algorithm, ... or, more generally the E-algorithm.
- computation of $F(t)$ for a fixed value of t and extrapolation by the confluent form of some algorithm such as the ε-algorithm.

In this section we shall review these two approaches and propose a new one combining them. It will consist in applying a confluent algorithm first for different values of t (for example the confluent form of the ε-algorithm will produce the sequence $\varepsilon_{2k}(t_0), \varepsilon_{2k}(t_1), \ldots$ for a fixed value of k) and then to accelerate the convergence of this sequence by a scalar algorithm. This new approach remains to be studied from the theoretical point of view.

Let us give a very simple example to illustrate the effectiveness of these approaches. We consider

$$I = \int_a^\infty e^{-x}\,dx = e^{-a}.$$

We have

$$F(t) = e^{-a} - e^{-t} = I - e^{-t} = I - F'(t).$$

Thus $F(n) = I - e^{-n}$ which shows that applying Aitken's Δ^2 process to $(F(n))$ will give I exactly. On the other hand if the confluent form of the ε-algorithm is applied to $F(t)$ then, by theorem 5.4, $\varepsilon_2(t) = I$. Thus both approaches give the exact result (if the integral $F(t)$ is computed exactly).

Let us begin our review by scalar extrapolation algorithms. In order to be able to choose the best possible extrapolation algorithm for the sequence $(S_n = F(t_n))$ we must first derive the asymptotic expansion of

$$I - F(t) = \int_t^\infty f(x)\,dx$$

and then choose (t_n). Such an analysis was given by Sidi [398] when

$$f(x) = u(\theta(x))\, e^{\phi(x)} h(x)$$

where u denotes either the sine or the cosine function, $\theta \in A^{(m)}$, $\phi \in A^{(k)}$ and $\lim_{x\to-\infty} \phi(x) = -\infty$ if $k \geq 1$, $h \in A^{(\gamma)}$, $A^{(j)}$ denoting the set of

infinitely differentiable functions for all $x > a$ and having an asymptotic expansion of the form

$$\alpha(x) \sim x^j \sum_{i=0}^{\infty} a_i x^{-i} \qquad (x \to \infty)$$

where j is a non-negative integer.

With these assumptions θ has the form $\theta(x) = \bar{\theta}(x) + \Delta(x)$ where $\bar{\theta}$ is a polynomial of degree m and $\Delta \in A^{(0)}$. Similarly $\phi(x) = \bar{\phi}(x) + \Lambda(x)$ where $\bar{\phi}$ is a polynomial of degree k and $\Lambda \in A^{(0)}$.

Therefore

$$\begin{aligned} e^{i\theta(x)} &= e^{i\bar{\theta}(x)}\delta(x) \quad \text{with} \quad \delta(x) = e^{i\Delta(x)} \\ e^{\phi(x)} &= e^{\bar{\phi}(x)}\lambda(x) \quad \text{with} \quad \lambda(x) = e^{\Lambda(x)}. \end{aligned}$$

If we set

$$\beta^*(x) = x^{-\gamma} h(x)\delta(x)\lambda(x)\beta(x)$$

then $\beta^* \in A^{(0)}$ and we have

$$I - F(t) = t^{\sigma+\gamma} e^{\bar{\phi}(t)} \{\cos\bar{\theta}(t) b_1(t) + \sin\bar{\theta}(t) b_2(t)\}$$

with $\sigma = \min(-m+1, -k+1)$ and

$$\begin{aligned} b_1(t) &= \begin{cases} \operatorname{Re}\beta^*(t) & \text{if} \quad f(x) = \operatorname{Re} g(x) \\ \operatorname{Im}\beta^*(t) & \text{if} \quad f(x) = \operatorname{Im} g(x) \end{cases} \\ b_2(t) &= \begin{cases} -\operatorname{Im}\beta^*(t) & \text{if} \quad f(x) = \operatorname{Re} g(x) \\ \operatorname{Re}\beta^*(t) & \text{if} \quad f(x) = \operatorname{Im} g(x) \end{cases} \\ g(x) &= e^{i\theta(x)} e^{\phi(x)} h(x). \end{aligned}$$

It follows from this analysis that if $t_0 < t_1 < t_2 < \ldots$ are taken as the consecutive zeros of $\sin\bar{\theta}(t)$ (or of $\cos\bar{\theta}(t)$) then $I - F(t_n)$ has an asymptotic expansion of the form

$$I - F(t_n) = g(t_n) \sum_{i=0}^{\infty} a_i t_n^{-i}$$

where

$$g(t) = \begin{cases} \cos\bar{\theta}(t) t^{\sigma+\gamma} e^{\bar{\phi}(t)} & \text{if the } t_n'\text{s are the zeros of } \sin\bar{\theta}(t) \\ \sin\bar{\theta}(t) t^{\sigma+\gamma} e^{\bar{\phi}(t)} & \text{if the } t_n'\text{s are the zeros of } \cos\bar{\theta}(t). \end{cases}$$

Thus the sequence $(S_n = F(t_n))$ can be efficiently extrapolated by the W-algorithm (see section 2.1).

For example if

$$I = \int_0^\infty \sin\left(\pi x^2/2\right) dx = 1/2$$

then $u(x) = \sin x$, $\theta(x) = \bar{\theta}(x) = \pi x^2/2$, $\phi(x) =$ constant and $\gamma = 0$. The t_n's are the zeros of $\pi x^2/2 = (n+1)\pi$ and thus $t_n = \sqrt{2(n+1)}$. Since $m = 2$ and $k = 0$ we have $\sigma = -1$. It follows that $g(t) = \cos(\pi t^2/2)/t$ and $g(t_n) = (-1)^{n+1}/t_n$ and we obtain

k	$W_k^{(0)}$
1	0.4997
3	0.500002
5	0.499999991
7	0.50000000004
9	0.4999999999998
11	0.5000000000000009

Examples are also given by Hasegawa and Torii [213].

Fourier and Hankel transforms can be treated by this procedure. Examples are given by Sidi [395] but, at this time, the W-algorithm was not yet found and extrapolation was performed by modifying a procedure due to Levin and Sidi [286] which is based on the D and d rational transformations. For other rational transformations, see Sidi [398]. These transformations were still extended later by Gray and Wang [200]. The W-algorithm was also proved to be efficient for divergent oscillatory infinite integrals that are defined in the sense of Abel summability that is if we set

$$\begin{aligned} f_\varepsilon(x) &= e^{-\varepsilon x} f(x), \qquad \varepsilon > 0 \\ F_\varepsilon(t) &= \int_a^t f_\varepsilon(x)\, dx \\ I_\varepsilon &= \int_a^\infty f_\varepsilon(x)\, dx \end{aligned}$$

we say that I is defined in the sense of Abel summability if

$$\lim_{\varepsilon \to 0+} I_\varepsilon = I.$$

For example if

$$I = \int_0^\infty x^2 J_0(x)\,dx$$

we have $m = 1$, $\theta(x) = x$, $\gamma = 3/2$. The t_n's are the zeros of $\sin t$, thus $t_n = (n+1)\pi$ and $g(t_n) = (-1)^n t_n^{3/2}$. We obtain

k	$W_k^{(0)}$
0	−1.65
3	−0.9999657
6	−0.99999999438
9	−0.9999999999991695
12	−1.0000000000001010

For the theory and other numerical examples, see Sidi [400] and Sidi [402, 403] for a review of these techniques. The convergence is studied by Sidi [405]. The same idea was used by Toda and Ono [435] for convergent integrals. They computed

$$I_n = \int_0^\infty e^{-a_n x} f(x)\,dx$$

with $a_n = 2^{-n}$ for $n = 0, 1, \ldots$. Obviously $\lim_{n\to\infty} I_n = I$. The sequence (I_n) was then extrapolated by Richardson's process. The theoretical study of this method was done by Sugihara [421] who also proposed to compute

$$J_n = \int_0^\infty e^{-a_n x^2} f(x)\,dx$$

followed by Richardson extrapolation. If both methods are applied to

$$\int_0^\infty \frac{\sin(x+1)}{(x+1)^2}\,dx = 0.50406706190692837198\ldots$$

we obtain

n	I_n	$T_n^{(0)}$ for I_n	J_n	$T_n^{(0)}$ for J_n
0	0.35478		0.40605	
1	0.42905	0.50333	0.46696	0.52786
2	0.46893	0.51063	0.50675	0.55278
3	0.48756	0.50456	0.52005	0.52556
4	0.49614	0.504057	0.51636	0.50093
5	0.50019	0.50406647	0.51055	0.50153
6	0.50215	0.5040670622	0.50729	0.50434
7	0.50311	0.504067061949	0.50567	0.50409
8	0.50359	0.504067061906947	0.50486	0.504638
9	0.50383	0.50406706190692817	0.50446	0.504067158

A modification of the W-algorithm was introduced by Sidi [403]. It can be applied to the computation of convergent or divergent oscillatory integrals. This modification of the W-algorithm consists in taking $g(t_n) = F(t_{n+1}) - F(t_n)$ in the algorithm instead of the previous g obtained by a tedious asymptotic analysis of $I - F(t)$. With this modification the assumptions to be checked only involve $\bar{\theta}$. Let us also mention that an acceleration technique closely related to Euler's transformation was studied by Lyness [303], see also Gabutti [166]. Since it is impossible here to enter into the details and to present all the results and algorithms, the interested reader is referred to the literature.

Let us now review the second approach. The first transformation which can be considered as a confluent algorithm was introduced by Gray and Atchison [195]. With our previous notations it consists in setting

$$G_1(F(t), h) = \frac{F(t+h) - R(t,h)\,F(t)}{1 - R(t,h)}$$

where $R(t,h) = f(t+h)/f(t)$ and $h > 0$.

Obviously

$$\lim_{t\to\infty} \frac{G_1(F(t),h) - I}{F(t+h) - I} = 0$$

if and only if

$$\lim_{t\to\infty} \frac{F(t+h) - F(t)}{I - F(t+h)} \cdot \frac{R(t,h)}{1 - R(t,h)} = 1.$$

Of course this transformation has little value if $\lim_{t\to\infty} R(t,h) = 0$ or 1. To avoid this drawback Gray and Atchison [196] introduced a parameter

α in the process

$$\overline{G}_1(F(t),h) = F(t+h) + \alpha \cdot \frac{F(t+h)-F(t)}{I-R(t,h)} \cdot R(t,h).$$

If $\alpha = \lim_{t\to\infty} \frac{1-R(t,h)}{R(t,h)} \cdot \frac{I-F(t+h)}{F(t+h)-F(t)}$ then $\overline{G}_1(F(t),h)$ converges to I faster than $F(t+h)$ when t tends to infinity. In practice the computation of α needs the knowledge of I and thus the procedure cannot be used directly. However if

$$\lim_{t\to\infty} \left(\frac{1}{f(t+h)} - \frac{1}{f(t)} \right) = \infty$$

then l'Hospital's rule can be applied to the preceding ratio and we obtain

$$\alpha = \frac{\lim_{t\to\infty} -f(t+h) \Big/ \frac{d}{dt}[1/f(t+h) - 1/f(t)]^{-1}}{\lim_{t\to\infty} [f(t+h)-f(t)]/f'(t)}.$$

If $\lim_{t\to\infty} R(t,h) = a \neq 0$ and 1 then $\alpha = 1$ and $\overline{G}_1(F(t),h)$ is identical to $G_1(F(t),h)$.

The same drawback can also be avoided by considering the new transformation

$$G(F(t),h) = \frac{F(ht) - R(t,h)\,F(t)}{1-R(t,h)}$$

with $R(t,h) = h\,f(ht)/f(t)$. This transformation was studied by Atchison and Gray [10] and other generalizations by Gray and Atchison [195]. However the most interesting generalization of G_1 was given by Gray, Atchison and McWilliams [197] who considered

$$G_k(F(t),h) = \frac{\begin{vmatrix} F(t) & F(t+h) & \cdots & F(t+kh) \\ f(t) & f(t+h) & \cdots & f(t+kh) \\ f(t+h) & f(t+2h) & \cdots & f(t+(k+1)h) \\ \vdots & \vdots & & \vdots \\ f(t+(k-1)h) & f(t+kh) & \cdots & f(t+(2k-1)h) \end{vmatrix}}{\begin{vmatrix} 1 & 1 & \cdots & 1 \\ f(t) & f(t+h) & \cdots & f(t+kh) \\ f(t+h) & f(t+2h) & \cdots & f(t+(k+1)h) \\ \vdots & \vdots & & \vdots \\ f(t+(k-1)h) & f(t+kh) & \cdots & f(t+(2k-1)h) \end{vmatrix}}.$$

These ratios of determinants can be recursively computed by the E-algorithm (section 2.1) or by the G-transformation (section 2.4) which was derived for that purpose.

Let us consider the computation of

$$I = \int_0^\infty \frac{\sin x}{x}\, dx = \frac{\pi}{2} = 1.570796326\ldots$$

We obtain

			absolute error
$F(10\pi)$	=	1.5390290796	0.032
$G_1(F(9\pi),\pi)$	=	1.5707884696	0.0000077
$G_2(F(7\pi),\pi)$	=	1.5707952217	0.0000011
$G_3(F(5\pi),\pi)$	=	1.5707961387	0.000000188
$G_4(F(3\pi),\pi)$	=	1.5707962057	0.000000121

When h tends to zero then the confluent form of the ε-algorithm is recovered, more precisely

$$\lim_{h\to\infty} G_k(F(t),h) = \varepsilon_{2k}(t).$$

Moreover these two transformations have the property that if f satisfies a linear differential equation of order k with constant coefficients then

$$G_k(F(t),h) = \varepsilon_{2k}(t) = I.$$

The main difference between higher order G-transformations and the confluent form of the ε-algorithm is that the first method requires the numerical computation of several integrals $F(t), F(t+h), \ldots, F(t+kh)$ while the second process needs the computation of only one integral (and of no integral at all if $t = a$) and the knowledge of the functions $f', f'', \ldots, f^{(2k)}$ (which can often be obtained by a computer algebra program). Thus if the confluent form of the ε-algorithm (or of the other algorithms described in chapter 5) is applied to

$$F(t) = \int_a^t f(x)\, dx$$

then $F'(t) = f(t)$, $F''(t) = f'(t)$ and so on.

Let us for example apply the confluent form of the ε-algorithm to

$$\int_1^{\infty} \frac{e^{-x}}{x}\, dx = 0.219383934\ldots$$

We obtain

t	$F(t)$	$\varepsilon_2(t)$	$\varepsilon_4(t)$	$\varepsilon_6(t)$
1	0	0.18393972	0.21021682	0.21917633
2	0.17048342	0.21559518	0.21881745	0.21938119
3	0.20633555	0.21878232	0.21932348	0.21938384
4	0.21560458	0.21926771	0.21937545	0.21938393
5	0.21823564	0.21935863	0.21938252	0.21938393
6	0.21902385	0.21937796	0.21938367	0.21938393

Reciprocally if I is known, the procedure can be used to obtain approximations of $F(t)$ for a fixed value of t, see Brezinski [56] for the details and numerical examples.

Let us now combine a confluent algorithm with a scalar extrapolation method.

For example we can try to extrapolate each column (or each row) in the preceding table by using Aitken's Δ^2 process. We obtain

1^{st} row	0.21459634	0.22381166
2^{nd} row	0.21906532	0.21950073
3^{rd} row	0.21934808	0.21939142
4^{th} row	0.21937871	0.21938465
5^{th} row	0.21938304	0.21938402
6^{th} row	0.21938376	0.21938394

1^{st} column	0.21588292	0.21883652	0.21927850	0.21936098
2^{nd} column	0.21913913	0.21935491	0.21937959	0.21938318
3^{rd} column	0.21935511	0.21938140	0.21938363	0.21938389
4^{th} column	0.21938387	0.21938393	0.21938393	

Thus the extrapolation of the columns gives good results while that of the rows does not increase the precision. Such a procedure needs to be theoretically studied.

6.7.5 Multiple integrals

Let us now assume that we want to obtain an approximation of

$$I = \int_C f(x)\,dx$$

where $x \in \mathbb{R}^p$ and C is a domain in $\mathbb{R}^p$. Such integrals are usually approximated by using a one dimensional quadrature rule successively for each variable thus leading to a so-called product quadrature rule. For example let us assume that we have to compute

$$I = \int_a^b \int_{g_1(x)}^{g_2(x)} f(x,y)\,dy\,dx.$$

This can be written as

$$I = \int_a^b F(x)\,dx$$

where $F(x) = \int_{g_1(x)}^{g_2(x)} f(x,y)\,dy$.

I is approximated by a quadrature rule of the form

$$I \simeq I_n = \sum_{i=0}^{n} A_i^{(n)} F(x_i).$$

Then, in this formula, $F(x_i)$ is again approximated by a quadrature formula

$$F(x_i) \simeq \sum_{j=0}^{m} B_j^{(m)} f(x_i, y_j).$$

The value of m and the quadrature formula (that is the $B_j^{(m)}$'s and the y_j's) can depend on the index i and we finally obtain an approximation of I of the form

$$\sum_{i=0}^{n} \sum_{j=0}^{m_i} C_{ij}^{(n,m_i)} f\left(x_i, y_j^{(i)}\right).$$

When C is a simplex or an hypercube and when the trapezoidal rule is used then the Euler-MacLaurin expansion can be generalized even if

f has vertex and edge singularities, see Lyness [304] or De Doncker-Kapenga [132] for reviews and references. Then if an asymptotic expansion of the error is known it is possible to use a well-adapted extrapolation method to accelerate the convergence of a sequence of product quadrature rules.

For example Genz [180] used the ε-algorithm in a local adaptative procedure (see also Genz [178, 179]). But since, as mentioned by Lyness [304], almost all the product quadrature formulæ have an error of the form

$$I_n - I = a_1 g_1(n) + a_2 g_2(n) + \cdots$$

where the functions g_i are like $n^{-\alpha_i}$ or $n^{-\alpha_i} \cdot \ln n$ and satisfy

$$\lim_{n \to \infty} g_{i+1}(n) / g_i(n) = 0$$

it seems appropriate to extrapolate by the E-algorithm (or simply to solve the underlying system of linear equations). This was the method followed by Hollosi and Keast [230]. Numerical results can be found in De Doncker-Kapenga [132]. They show that, starting from 8 results obtained by the product trapezoidal rule the best of which having a relative error of $0.453 \cdot 10^{-1}$, one can obtain an error of $0.102 \cdot 10^{-7}$. As stated by this author, the use of extrapolation methods in automatic integration works *for a large class of problems without reference to the specific integrand behaviour.* But, on the other hand, as pointed out by Lyness [304] *extrapolation in numerical quadrature is very useful and even optimal for a small class of problems, is convenient but not optimal for a much wider class; but there is a huge class of problems for which it is not relevant or appropriate and should not be used.* Thus these techniques still need some more studies before being efficiently implemented and used in an adaptative quadrature package.

The extrapolation of infinite double integrals of the form

$$I = \int_0^\infty \int_0^\infty f(x, y)\, dx\, dy$$

was studied by Levin [285] by generalizations of the confluent form of the ε-algorithm and the G-transformation.

6.7.6 Numerical differentiation

The formulæ for approximating the derivatives of a function are well known. For example

$$\begin{array}{ll} & [f(x+h)-f(x)]/h \\ \text{or} & [f(x+h/2)-f(x-h/2)]/h \end{array}$$

are approximations of $f'(x)$ while $f''(x)$ can be approximated by

$$\begin{array}{ll} & [f(x+2h)-2f(x+h)+f(x)]/h^2 \\ \text{or by} & [f(x+h)-2f(x)+f(x-h)]/h^2. \end{array}$$

By Taylor's formula we have, if f is sufficiently differentiable

$$\begin{aligned} [f(x+h)-f(x)]/h &= f'(x)+a_1h+a_2h^2+\cdots \\ [f(x+h/2)-f(x-h/2)]/h &= f'(x)+a_1'h^2+a_2'h^4+\cdots \\ [f(x+2h)-2f(x+h)+f(x)]/h^2 &= f''(x)+b_1h+b_2h^2+\cdots \\ [f(x+h)-2f(x)+f(x-h)]/h^2 &= f''(x)+b_1'h^2+b_2'h^4+\cdots \end{aligned}$$

where the coefficients depend on the higher derivatives of f. Thus, due to the expansion of the error, Richardson extrapolation is well adapted to these formulæ with h as the variable in the first and the third cases and h^2 in the others.

Let us give a numerical example taken from Laurent [275]. It consists in applying the second formula to $f(x)=1/(x-1)$ with $x=0$. We have

n	h_n	$T_0^{(n)}$	$T_n^{(0)}$
0	1	−1.3333333	−1.3333333
1	1/2	−1.0666667	−0.9777777
2	1/4	−1.0158729	−1.0003527
3	1/8	−1.0039216	−0.99999862
4	1/16	−1.0009775	−0.99999998

6.8 Prediction

Up to now, extrapolation methods were used to predict the value of the limit S of a sequence (S_n) knowing some of its terms, say $S_0, S_1, \ldots, S_n$. We shall now use extrapolation methods to predict the values of the unknown terms of the sequence, that is $S_{n+1}, S_{n+2}, \ldots$ and so on.

Such an idea was first formulated by Gilewicz [186] who used the expansion into a formal power series of Padé approximants to estimate the unknown coefficients of a series. The same idea was also exploited by Sidi and Levin [409] who used the so-called t-transformation which is a rational approximation of the Padé-type for power series.

Of course the same type of idea can be used for sequences instead of series. Since Aitken's Δ^2 process is one of the most popular and successful extrapolation method we shall first use it for predicting the unknown terms of a sequence. Then, because of its generality, we shall make use of the E-algorithm. But before going into the details let us explain the basic idea of prediction as given by Brezinski [71].

In section 1.2 we saw that an extrapolation method consisted in writing the system of equations

$$R(S_i, \ldots, S_{i+q}, S) = 0\,, \qquad i = n, \ldots, n+p$$

where R depends on p unknowns $a_1, \ldots, a_p$ and then to solve it for obtaining the value of the unknown S (which usually depends on the first index n and on $k = p + q$ and was denoted, for that reason, by $T_k^{(n)}$).

Having determined, by this procedure, the values of $a_1, \ldots, a_p$ and $S = T_k^{(n)}$ it is now possible to obtain an approximate value $S_{n+k+1}^{(k,n)}$ of S_{n+k+1} by solving the equation

$$R\left(S_{n+p+1}, \ldots, S_{n+k}, S_{n+k+1}^{(k,n)}, T_k^{(n)}\right) = 0.$$

Then an approximate value $S_{n+k+2}^{(k,n)}$ of S_{n+k+2} can be obtained by solving the equation

$$R\left(S_{n+p+2}, \ldots, S_{n+k}, S_{n+k+1}^{(k,n)}, S_{n+k+2}^{(k,n)}, T_k^{(n)}\right) = 0$$

and so on. Since these approximate values $S_{n+k+i}^{(k,n)}$ for $i = 1, 2, \ldots$ depend on the indexes n and k, they are indicated in the upper position. If n is fixed it will be dropped and similarly for k.

Aitken's Δ^2 process corresponds to $p = q = 1$ and

$$R(u_0, u_1, u) = u_1 - u + a_1(u_0 - u) = 0.$$

We have

$$T_2^{(n)} = S_{n+2} - \frac{(\Delta S_{n+1})^2}{\Delta^2 S_n}$$

and thus

$$\begin{aligned} S_i^{(n)} &= S_i, & i = 0, \ldots, n+2 \\ S_{n+i+2}^{(n)} &= S_{n+2} + \frac{\Delta S_{n+1}}{\Delta S_n} \cdot \left(S_{n+i+1}^{(n)} - S_{n+1} \right), & i = 1, 2, \ldots \end{aligned}$$

Let us study the difference $S_{n+i+2} - S_{n+i+2}^{(n)}$ when n goes to infinity. We have the

Theorem 6.32

Let (S_n) be a convergent sequence such that $\exists M, N, \forall n \geq N, |\Delta S_{n+1} / \Delta S_n| \leq M$. Then $\forall i \geq 1$, $\lim_{n \to \infty} \left(S_{n+i+2} - S_{n+i+2}^{(n)} \right) = 0$.

Thus the greater n, the better the predicted values. We also have the more precise result

Theorem 6.33

Let (S_n) be a convergent sequence such that there exists a with $\lim_{n \to \infty} \Delta S_{n+1} / \Delta S_n = a$. Then $\forall i \geq 1$, $\lim_{n \to \infty} \left(S_{n+i+2} - S_{n+i+2}^{(n)} \right) \Big/ \Delta S_{n+1} = 0$. Moreover if $a \neq 0$, then $\forall i \geq 1$, $\lim_{n \to \infty} \left(S_{n+i+2} - S_{n+i+2}^{(n)} \right) \Big/ \Delta S_{n+i+1} = 0$.

Let us take $S_n = 0.9^n/(n+1)$. We have the following results

n	i	$n+i+2$	$S_{n+i+2} - S_{n+i+2}^{(n)}$	$\left(S_{n+i+2} - S_{n+i+2}^{(n)} \right) \Big/ \Delta S_{n+1}$
0	5	7	−0.12297	0.68319
1	4	7	−0.04371	0.49809
2	3	7	−0.01446	0.28343
3	2	7	−0.00398	0.12138
4	1	7	−0.00071	0.03147
0	1	3	−0.02884	0.16023
1	1	4	−0.00825	0.09404
2	1	5	−0.00313	0.06132
3	1	6	−0.00141	0.04286
4	1	7	−0.00071	0.03147
0	1	3	−0.02884	0.16023
0	2	4	−0.06059	0.33662
0	3	5	−0.08709	0.48382
0	4	6	−0.10752	0.59732
0	5	7	−0.12297	0.68319

The last results show that the precision decreases when n is fixed and when i increases which is not surprising. This fact can be rigorously proved in the following case

Theorem 6.34

Let (S_n) be a sequence converging to S and such that $\forall n$, $\Delta S_n \leq 0$ and

$$\frac{\Delta S_1}{\Delta S_0} \leq \frac{\Delta S_2}{\Delta S_1} \leq \cdots < 1.$$

Then, $\forall n$

$$S - S^{(n)} \leq \cdots \leq S_{n+4} - S_{n+4}^{(n)} \leq S_{n+3} - S_{n+3}^{(n)} \leq 0$$

where $S^{(n)} = \lim_{i \to \infty} S_{n+i+2}^{(n)}$.

The assumptions of this theorem are satisfied, in particular, by totally monotonic sequences that is such that $\forall k, n, (-1)^k \Delta^k S_n \geq 0$ (see section 2.3). The assumption $\Delta S_{n+1}/\Delta S_n < 1$ for all n is not restrictive since if $\Delta S_{n+1}/\Delta S_n = 1$ for some index n then $\forall i \geq n$, $\Delta S_i = \Delta S_n$ and the sequence cannot converge. Totally monotonic sequences also satisfy the assumptions of theorems 6.32 and 6.33.

Let us now assume that R has the form

$$S_n - S - a_1 g_1(n) - \cdots - a_k g_k(n) = 0$$

where, as in the E-algorithm (see section 2.1), the auxiliary sequences $(g_i(n))$ can depend on (S_n). Then, performing the same procedure as above, we obtain, for $i = 1, 2, \ldots$

$$S_{n+k+i}^{(k,n)} = -\frac{\begin{vmatrix} 0 & 1 & g_1(n+k+i) & \cdots & g_k(n+k+i) \\ S_n & 1 & g_1(n) & \cdots & g_k(n) \\ \vdots & \vdots & \vdots & & \vdots \\ S_{n+k} & 1 & g_1(n+k) & \cdots & g_k(n+k) \end{vmatrix}}{\begin{vmatrix} 1 & g_1(n) & \cdots & g_k(n) \\ \vdots & \vdots & & \vdots \\ 1 & g_1(n+k) & \cdots & g_k(n+k) \end{vmatrix}}.$$

If $g_1(n+k+i), \ldots, g_k(n+k+i)$ depend on $S_0, \ldots, S_{n+k+i-1}$ we shall replace in their expressions S_{n+k+1} by $S_{n+k+1}^{(k,n)}$, $\ldots$, $S_{n+k+i-1}$ by $S_{n+k+i-1}^{(k,n)}$.

The predicted values $S_{n+k+i}^{(k,n)}$ can be recursively computed by a generalization of the Neville-Aitken scheme for polynomial interpolation which is due to Mühlbach [329]. It is the so-called Mühlbach-Neville-Aitken algorithm (MNA for short), a particular case of which is the E-algorithm, see Brezinski [62]. Assuming that $S_0, \ldots, S_m$ are known, then for $j > m$ and $n + k \leq m$, this algorithm is as follows

$$S_j^{(0,n)} = S_n\,, \quad g_{0,i}^{(n)} = g_i(n) - g_i(j)\,,$$
$$n = 0, 1, \ldots;\ i = 1, 2, \ldots$$

$$S_j^{(k,n)} = S_j^{(k-1,n)} - \frac{S_j^{(k-1,n+1)} - S_j^{(k-1,n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}\,,$$
$$n = 0, 1, \ldots;\ k = 1, 2, \ldots$$

$$g_{k,i}^{(n)} = g_{k-1,i}^{(n)} - \frac{g_{k-1,i}^{(n+1)} - g_{k-1,i}^{(n)}}{g_{k-1,k}^{(n+1)} - g_{k-1,k}^{(n)}} \cdot g_{k-1,k}^{(n)}\,,$$
$$n = 0, 1, \ldots;\ k = 1, 2, \ldots;\ i > k.$$

Since the rules of this algorithm are the same as those of the E-algorithm, except for the initializations, it can be implemented by using the subroutine EALGO (see section 7.3) for the E-algorithm.

We have the following convergence result

Theorem 6.35

Let (S_n) be a convergent sequence. If $\forall i$, there exists $b_i \neq 1$ such that $\lim_{n\to\infty} g_i(n+1)/g_i(n) = b_i$ and if $\forall j \neq i$, $b_j \neq b_i$ then $\forall i, k$, $\lim_{n\to\infty} \left(S_{n+k+i}^{(k,n)} - S_{n+k+i}\right) = 0$.

If $g_i(n) = x_n^i$ then the MNA-algorithm reduces to the Neville-Aitken scheme for polynomial interpolation and we have, for $j > m$ and $n + k \leq m$

$$S_j^{(0,n)} = S_n\,, \qquad n = 0, 1, \ldots$$

$$S_j^{(k,n)} = S_j^{(k-1,n)} - \frac{S_j^{(k-1,n+1)} - S_j^{(k-1,n)}}{x_{n+k} - x_n} \cdot (x_n - x_j)\,,$$
$$n = 0, 1, \ldots;\ k = 1, 2, \ldots$$

In both cases, if $n + k > m$, the unknown exact values $S_{m+1}, \ldots, S_{n+k}$ have to be replaced by the predicted ones.

If $\exists b \neq 0, \pm 1$ such that $\lim_{n\to\infty} x_{n+1}/x_n = b$ then the conditions of theorem 6.35 are satisfied.

Let us give an application of prediction techniques to the solution of a parabolic evolution problem as considered by Morandi Cecchi, Redivo Zaglia and Scenna [326]. We are looking for a numerical solution of

$$\begin{cases} u_t - \Delta u = f & \text{in } Q_T \\ u|_{\partial\Omega} = 0, & t \geq 0 \\ u(x,0) = u_0(x) & \text{in } \Omega \end{cases}$$

where $u = u(x,t), f = f(x,t), Q_T = \Omega \times [0,T]$ and $\Omega \subset \mathbb{R}^d$, $(d = 1,\ldots,n)$.

Under suitable conditions, described by Morandi Cecchi and Nocifforo [324], the following variational formulation of this problem has a unique solution

$$\begin{cases} \text{find } u \in L^2(0,T;H_0^1(\Omega)) \text{ such that} \\ \int_0^T [(\nabla u, \nabla\phi) - (u,\phi_t)]\,\mathrm{d}t = \int_0^T (f,\phi)\mathrm{d}t + (u_0(x),\phi(0)) \\ \forall \phi \in L^2(0,T;H_0^1(\Omega)) \text{ with } \phi(T) = 0 \text{ and} \\ (\cdot,\cdot) \text{ being the inner product in } L^2(\Omega). \end{cases}$$

We consider a discretization with respect to t at the values $t_0 = 0 < t_1 < \cdots < t_N = T$. In every slab $\Omega \times [t_n, t_{n+1}]$ we take a finite element discretization of the variational problem with the approximate solution and the test functions in $S_h^p(\Omega) \otimes P^q([t_n, t_{n+1}])$ where $S_h^p(\Omega)$ is the finite element space of piecewise polynomials of degree p in Ω, h the mesh parameter, and $P^q([t_n, t_{n+1}])$ is the set of polynomials of degree q in $[t_n, t_{n+1}]$.

Going from one slab to the next one, the continuity in time of the approximate solution is imposed. Thus we obtain an iterative process giving the solution at time t_{n+1} from the solution at time t_n. This method is equivalent to applying first Galerkin finite elements in space and then finite differences in the time t.

If f is independent of t we obtain an iterative scheme of the form

$$U^{n+1} = MU^n + b$$

where U^n is the solution at $t_n = n \cdot \Delta t\,(n = 0, 1, \ldots, N-1)$ in the nodes of the subdivision of Ω.

We are interested in obtaining U^N from $U^0, U^1, \ldots, U^m$ where $m \ll N$. This can be done by the prediction techniques described above.

Denoting by $\overline{U}^n \in \mathbb{R}^{N_h}$ the vector of the coefficients of U^n, we consider, for example, the following initial boundary value problem with homogeneous Neumann initial conditions

domain	$\Omega = (0, \pi)$,	$N_h = 21$, $T = 1$ sec., $N = 200$
initial condition	$u_0(x) = \cos(x)$	
exact solution	$u(x,t) = e^{-t}\cos(x)$.	

Applying the prediction algorithm obtained from Aitken's Δ^2 process, only the first three time level values $(\overline{U}^0, \overline{U}^1, \overline{U}^2)$ are sufficient to predict $\overline{U}(T)$ with, at least, the same precision as that given by the method itself. In the following table the exact solutions $\overline{U}^0$, $\overline{U}^N$ and the number of the exact significant digits for the approximated and predicted solutions at $t_N = T$ are shown. Because of the symmetry in the behaviour of the solution, in the table only $N_h/2 + 1$ nodes in space are reported.

node	Exact solution		Approx. sol.	Pred. sol.
	$\overline{U}^0$	$\overline{U}^N$	n. of sign. digits	
1	1.000000	0.367879	2.539	2.539
2	0.987688	0.363350	2.539	2.539
3	0.951057	0.349874	2.539	2.539
4	0.891007	0.327783	2.539	2.539
5	0.809017	0.297621	2.539	2.539
6	0.707107	0.260130	2.539	2.539
7	0.587785	0.216234	2.539	2.539
8	0.453990	0.167014	2.539	2.539
9	0.309017	0.113681	2.539	2.539
10	0.156434	0.057549	2.539	2.539
11	0.000000	0.000000	all	all

Other examples with homogeneous Dirichlet initial conditions can be found in Morandi Cecchi, Redivo Zaglia and Scenna [326].

This page intentionally left blank

Chapter 7

SOFTWARE

In the first section of this chapter we shall describe the general *directions for use* of the subroutines which can be found on the floppy disk provided with this book. The second section deals with some questions and problems related to computer arithmetic. The list of the programs contained in the floppy disk is given in section 7.3 together with some explanations about them.

7.1 Programming the algorithms

We shall now give the general principles which govern the use of the subroutines performing an extrapolation algorithm for scalar and vector sequences.

Let us take the example of the ε-algorithm. The situation is absolutely the same for the other algorithms. The simplest method for programming the ε-algorithm is to store in the computer's memory all the $\varepsilon_k^{(n)}$'s. Starting from a given number of terms of the initial sequence, the columns in the ε-array are then computed one by one from the two preceding ones. Notice that, due to the rule of the ε-algorithm, each column contains one term less than the preceding column. Thus, starting from $S_0, \ldots, S_k$ one can only compute the triangular array of figure 7.1. The same is true for the progressive form of the algorithm; then starting from $S_0, \ldots, S_k$ and $\varepsilon_0^{(0)}, \ldots, \varepsilon_k^{(0)}$ the array is filled up descending diagonal by descending diagonal, each diagonal having one term less than the diagonal above it.

If a new term, S_{k+1}, is added then the new ascending diagonal $\varepsilon_0^{(k+1)}$, $\varepsilon_1^{(k)}, \ldots, \varepsilon_k^{(1)}, \varepsilon_{k+1}^{(0)}$ is easily computed. The same is true for the progressive form of the algorithm.

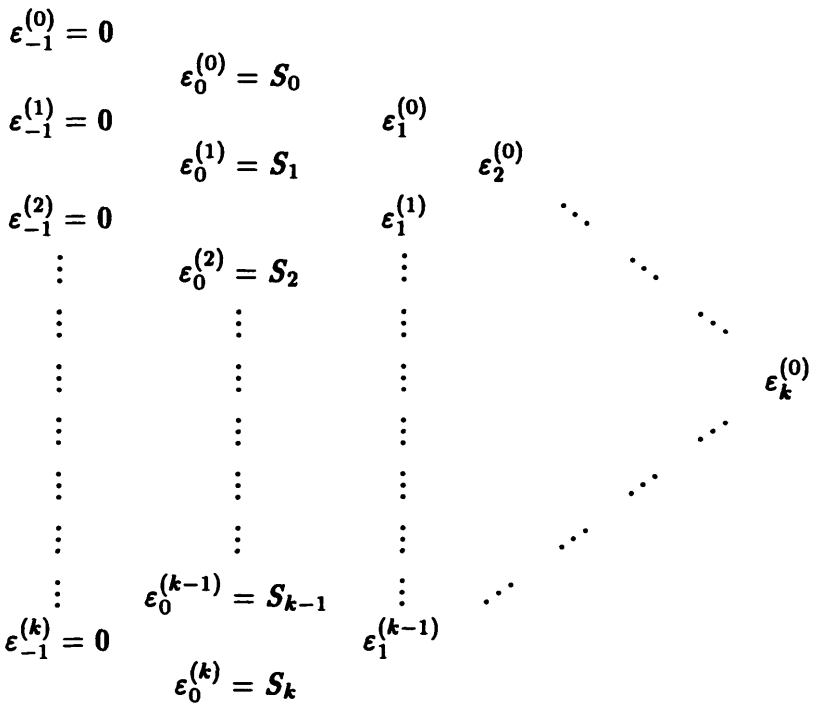


Figure 7.1: The ε-array.

Let us assume now that we want to reduce the computer storage. The simplest method for implementing the ε-algorithm is to store only the last two columns, then to compute the new column and to make a shift. However this procedure suffers from a capital drawback: if it is wanted to add new terms, $S_{k+1}, S_{k+2}, \ldots$ all the computations have to be done again. This drawback can be avoided by keeping, in the computer's memory, the last ascending diagonal $\varepsilon_0^{(k)}, \varepsilon_1^{(k-1)}, \varepsilon_2^{(k-2)}, \ldots, \varepsilon_k^{(0)}$. Thus, if a new term $\varepsilon_0^{(k+1)} = S_{k+1}$ is added, $\varepsilon_1^{(k)}$ is computed from $\varepsilon_{-1}^{(k+1)}, \varepsilon_0^{(k)}$ and $\varepsilon_0^{(k+1)}$. Then follows the computation of $\varepsilon_2^{(k-1)}$ from $\varepsilon_0^{(k)}, \varepsilon_1^{(k-1)}$ and $\varepsilon_1^{(k)}$. After that, $\varepsilon_3^{(k-2)}$ is obtained from $\varepsilon_1^{(k-1)}, \varepsilon_2^{(k-2)}$ and $\varepsilon_2^{(k-1)}$ and so on. Thus, gradually, a new ascending diagonal replaces the old one. In order to program this technique more easily, some auxiliary storage is needed as explained by Wynn [474, 476].

Some algorithms are more difficult to program and, in particular,

some of them need the storage of several diagonals and of some other auxiliary quantities or arrays. However, from the user's point of view, the situation is the same and this technique allows a very easy use of the algorithms and of the corresponding subroutines.

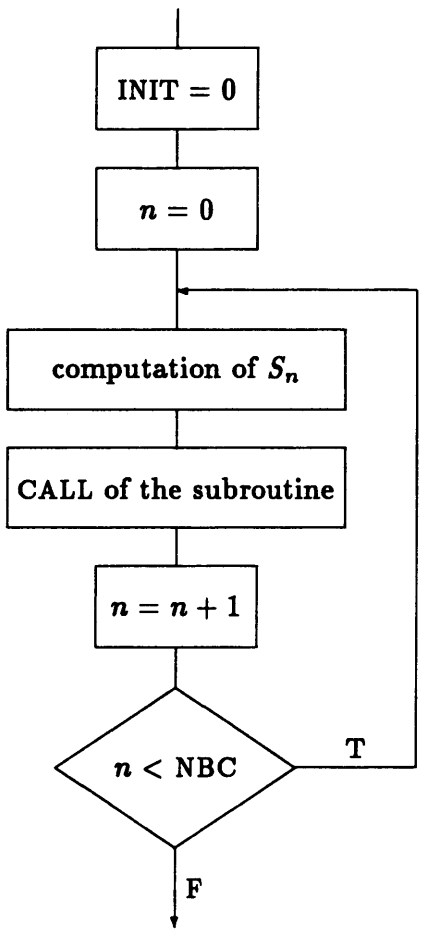


Figure 7.2: The flowchart of the programs.

The subroutines have to be used *in parallel* with the computation of the successive terms of the sequence (S_n). These successive terms are computed in a loop. After the computation of each new term of the sequence, a CALL of the subroutine is made and the subroutine computes

the new ascending diagonal as far as possible (or up to a certain column according to the choice of the user). Then the subroutine returns to the main program, in the loop where the terms of the sequence are computed. Of course, before beginning the loop, the subroutine must be told (by setting the flag INIT to 0, which will be put to 1 by the subroutine) that it will be given a new sequence to transform. The normal forms of all the algorithms were programmed in that way, see Brezinski [56].

Thus the use of the subroutines is described by the flowchart of figure 7.2, where NBC is the maximum number of calls (see also the pseudocode given in section 7.3).

The subroutines using a particular rule for avoiding numerical instability are more difficult to program. Indeed the particular rules are only valid when two adjacent quantities in a column are nearly equal and they can no more be used in the case of three nearly equal quantities. Thus such points of instability must be located by pointers as explained by Wynn [476] (see also Brezinski [56]). However, using such a subroutine is exactly the same as using a subroutine without particular rules.

7.2 Computer arithmetic

Extrapolation algorithms are often subject to an important propagation of rounding errors due, most of the time, to cancellation. Thus these algorithms must be programmed with great care and, as explained in the previous chapters, particular rules for avoiding (when possible) numerical instability have to be used. However this is not always possible and we shall give a numerical example for illustrating this point. This numerical example was already mentioned in section 1.9.

If Aitken's Δ^2 process is applied to the sequence $(S_n = \lambda^n)$ where $\lambda \neq 1$ then, $\forall n$, we must have $\varepsilon_2^{(n)} = 0$. We made use of the five following formulæ which are equivalent

$$T_1: \quad \varepsilon_2^{(n)} = \left(S_n S_{n+2} - S_{n+1}^2\right) \Big/ \left(S_{n+2} - 2S_{n+1} + S_n\right)$$

$$T_2: \quad \varepsilon_2^{(n)} = \left(S_n S_{n+2} - S_{n+1}^2\right) \Big/ \left(S_{n+2} + S_n - 2S_{n+1}\right)$$

$$T_3: \quad \varepsilon_2^{(n)} = S_{n+1} - \left(S_{n+2} - S_{n+1}\right)\left(S_{n+1} - S_n\right) / \left(S_{n+2} - 2S_{n+1} + S_n\right)$$

$$T_4: \quad \varepsilon_2^{(n)} = S_{n+1} - \left(S_{n+2} - S_{n+1}\right)\left(S_{n+1} - S_n\right) / \left(S_{n+2} + S_n - 2S_{n+1}\right)$$

$$T_5: \quad \varepsilon_2^{(n)} = S_{n+1} + 1/(1/(S_{n+2} - S_{n+1}) - 1/(S_{n+1} - S_n)) .$$

The sequence (S_n) has been computed in double precision but the computation of $\varepsilon_2^{(n)}$ has been done in single precision. The computations have been conducted on a VAX computer (1st line) and on a ZENITH Supersport PC/Computer (with a mathematical coprocessor) (2nd line).

We obtained the following results with $\lambda = 0.9997$

n	T_1	T_2	T_3	T_4	T_5
1	−1.0000000	−0.5000000	−0.5098506	0.2449247	−0.5095115
	−0.5098506	−0.5098506	−0.5098506	−0.5098506	−0.5098506
2	0.5000000	0.5000000	0.2450747	0.2450747	0.2450722
	0.2450747	0.2450747	0.2450747	0.2450747	0.2450747
3	−1.0000000	−0.5000000	−0.5086511	0.2452246	−0.5089999
	−0.5086511	−0.5086511	−0.5086511	−0.5086511	−0.5086511
4	0.5000000	0.5000000	0.2453744	0.2453744	0.2454439
	0.2453745	0.2453745	0.2453745	0.2453745	0.2453745
5	0.0000000	0.0000000	0.2456741	0.2456741	0.2456981
	0.2456741	0.2456741	0.2456741	0.2456741	0.2456741
6	0.0000000	0.0000000	−0.5065536	0.2458239	−0.5065745
	−0.5065536	−0.5065536	−0.5065536	−0.5065536	−0.5065536
7	−1.0000000	−0.5000000	−0.5062542	0.2458239	−0.5063213
	−0.5062542	−0.5062542	−0.5062542	−0.5062542	−0.5062542
8	0.5000000	0.5000000	0.2459735	0.2459735	0.2459050
	0.2459736	0.2459736	0.2459736	0.2459736	0.2459736

Then the results oscillate continuously between false values except for the VAX which gives the right answer from time to time with formulæ T_1 and T_2.

Let us now consider the sequence

$$\begin{aligned} S_0 &= 1 \\ S_{n+1} &= \exp(-S_n) , \qquad n = 0, 1, \ldots \end{aligned}$$

which converges to $S = 0.5671432904\ldots$. It can be proved that its convergence is accelerated by Aitken's Δ^2 process and we obtained the following results

n	T_1	T_2	T_3	T_4	T_5
1	0.5822261	0.5822261	0.5822261	0.5822261	0.5822261
	0.5822261	0.5822261	0.5822261	0.5822261	0.5822261
10	0.5671424	0.5671484	0.5671438	0.5671438	0.5671438
	0.5671438	0.5671438	0.5671438	0.5671438	0.5671438
15	0.5671901	0.5670865	*	*	*
	*	*	*	*	*
25	0.5526316	0.5250000	*	*	*
	*	*	*	*	*
26	0.5454546	0.6000000	*	*	*
	*	*	*	*	*
27	0.5833333	0.5833333	*	*	*
	*	*	*	*	*
28	0.5000000	0.7500000	*	*	*
	*	*	*	*	*
29	0.5000000	0.5000000	*	*	*
	*	*	*	*	*

(with $* = 0.567143$).

Formulæ T_1 and T_2 (which are known to be numerically unstable) always give 0.5 for $n \geq 29$ on the VAX computer and thus the sequence $\left(\varepsilon_2^{(n)}\right)$ converges to a wrong answer. Notice that the same formulæ give the right answer on the ZENITH computer.

As mentioned in section 6.6, sums are often to be computed specially in vector extrapolation algorithms. Due to possible cancellation errors such a computation is often unstable. However the arithmetic of the computer can be corrected by using a procedure due to Pichat [358]

we want to compute $S = \sum_{i=1}^{n} y_i$

- set $S_1 = y_1$
- for $i = 1, \ldots, n-1$, do
  $S_{i+1} = S_i + y_{i+1}$
  $e_i = -S_{i+1} + S_i + y_{i+1}$ if $|S_i| \geq |y_{i+1}|$
  $e_i = -S_{i+1} + y_{i+1} + S_i$ if $|S_i| < |y_{i+1}|$
- compute $T = S_n + \sum_{i=1}^{n-1} e_i$.

It can be proved that all the digits of T are exact up to the precision of the computer if it works with a check digit.

Some compilers (usually on micro-computers) perform the intermediate computations with a precision higher than the precision used for representing the results in the computer's memory (which can be true, in particular, when a coprocessor is present). In such a case the above correction does not bring any improvement and other techniques must be used depending on the number of arithmetical registers. They are described in details by Pichat and Vignes [359].

Let us give a numerical example for illustrating the procedure.

$$S = 1 + 1000 \cdot 10^{-m} = 1 + \sum_{i=1}^{1000} 10^{-m}$$

has been computed without and with Pichat's correction on an EPSON HX 20 computer working with 6 decimal digits in single precision. We obtained from our BASIC program

m	S	without correction	with correction
1	101	100.999	101
2	11	11.0002	11
3	2	2.00005	2
4	1.1	1.10002	1.1
5	1.01	1.01001	1.01
6	1.001	1.00095	1.001
7	1.0001	1.00012	1.0001
8	1.00001	1.00000	1.00001

On a ZENITH Supersport computer with an arithmetical coprocessor and working with 7 decimal digits in single precision, our FORTRAN program gave the exact results without correction.

7.3 Programs

The floppy disk provided with this book contains subroutines for most of the extrapolation algorithms described in the preceding chapters. They have been written in FORTRAN 77 and we tried to make them as portable as possible. Subroutines for some extrapolation algorithms were already given by Brezinski [56, 59, 64]. These subroutines were all completely

rewritten and their programming have been improved and often simplified. A complete description of the programming techniques used can be found in Redivo Zaglia [373]. The transcription of all the subroutines into another language (such as PASCAL) was made very easy using structured programming.

For each algorithm we give a main program showing how to use it (with the same name as the principal subroutine, preceded by the letter M) and two files containing the corresponding numerical results (with the same name as the main program). In the file with the extension .PC, the user can find the results obtained on a PC with the Microsoft FORTRAN Optimizing Compiler. The second file, with the extension .VAX, contains the results obtained with the VAX FORTRAN Compiler.

The principal subroutines are (in alphabetical order)

ACCES	ACCES algorithm
BLBORD	Block bordering method
BORDER	Bordering method
COMPOS	Composite sequence transformations
CRPA	CRPA algorithm
EALGO	E-algorithm
EPSRHA	Simultaneous ϱ and ε-algorithms and generalizations
EPSRHO	ϱ-algorithm, ε-algorithm and its generalizations
EPSTOP	Tolopogical ε-algorithm
EPSVEC	Vector ε-algorithm
GTRAN	G-transformation
HALGO	H-algorithm
IDELTA	Iterated Aitken's Δ^2 process
INVLAP	Laplace transform inversion
LEVINT	Levin's transforms
OMEGA	ω-algorithm
OVERHO	Overholt process
QDALGO	qd-algorithm
RICHAR	Richardson process
RSALGO	rs-algorithm
SEALGO	Simultaneous E-algorithm
SELECT	Automatic selection
THETA	Θ-algorithm
VEALGO	Vector E-algorithm
VGRAN	Vector G-transformation

Let us give general comments on the subroutines implementing ex-

trapolation procedures. More detailed explanations can be founded in each of them.

- Before the first CALL of a subroutine for treating a new sequence, the input/output integer argument INIT must be set to 0. Its value is changed to 1 by the subroutine during the first CALL. As long as INIT has the value 1, the subroutine considers that it is treating the same sequence as in the preceding calls. Thus for treating a new sequence, this argument has to be reset to 0.

- The input (or input/output) integer argument MAXCOL controls the last column to be computed in the array. For example, for the ε-algorithm, MAXCOL must be even (if not it is replaced by MAXCOL-1) and the subroutine returns after the k-th CALL

$$\varepsilon_{k-1}^{(0)} \qquad \text{if } k \leq \text{MAXCOL}+1 \text{ and } k \text{ is odd}$$

$$\varepsilon_{k-2}^{(1)} \qquad \text{if } k < \text{MAXCOL}+1 \text{ and } k \text{ is even}$$

$$\varepsilon_{\text{MAXCOL}}^{(k-1-\text{MAXCOL})} \qquad \text{if } k > \text{MAXCOL}.$$

That is we shall obtain successively

call 1	$\varepsilon_0^{(0)}$
call 2	$\varepsilon_0^{(1)}$
call 3	$\varepsilon_2^{(0)}$
call 4	$\varepsilon_2^{(1)}$
$\vdots$	$\vdots$
call MAXCOL	$\varepsilon_{\text{MAXCOL}-2}^{(1)}$
call MAXCOL + 1	$\varepsilon_{\text{MAXCOL}}^{(0)}$
$\vdots$	$\vdots$
call MAXCOL + m	$\varepsilon_{\text{MAXCOL}}^{(m-1)}$

Usually the last diagonal or the last row which has been computed in the array is also given.

- There are no error and/or warning messages printed in the subroutines but the occurrence of an error and/or a warning is indicated by a nonzero value of the output (or input/output) integer argument IER. Its output value is a code indicating the nature of the error and/or the warning. Each subroutine has its own codes and, for some of them, IER must be reset to 0 after an error and/or a warning.

- The subroutines do not print the results. They are all returned by arguments.

- At the beginning of each subroutine, the user could find its purpose, usage, a list and a description of the arguments, remarks, explanations about the dimensions of the arrays, and a list of the routines eventually required.

- There are several types of arguments.

  input Must be initialized in the calling procedure and is not modified by the subroutine.

  output Cannot be a constant, a parameter constant or an expression. However such an argument must not be modified by the user because it is used in the next call of the subroutine.

  input/output Must be initialized in the calling procedure but not as a constant, a parameter constant or an expression. The routine returns an output value through it.

  Some subroutines need the use of **working** arrays. They appear in the list of the arguments of the subroutines because their dimensions must be given in the calling procedure. Such working arrays must not be modified by the user between two consecutive calls of the subroutine. They can only be used after the end of the loop containing the CALL of the subroutine.

- For arrays with two dimensions, only the last one can have an assumed-size declarator. The first dimension must be given as specified through the argument IR. Only overpassing this first dimension is checked. In each subroutine, the dimensions of the various arrays are given in the remarks.

- In order to avoid division by zero, the input argument EPS is used to check if the absolute value of a denominator is less than EPS. Thus EPS must be positive. If EPS is negative or equal to zero, then a division by zero can occur because the subroutines do not control whether or nor EPS is a negative or zero real number.

- All the arguments which are not integers must be declared in a DOUBLE PRECISION statement.

- The parallel use of the subroutines described in section 7.1 does not allow to treat simultaneously several sequences with the same subroutine. In order to show how to avoid this drawback, the subroutine SEALGO for the E-algorithm has been specially written. The other subroutines can be modified in a similar way.

- Since some extrapolation algorithms are quite similar, they have been gathered in a single subroutine. Two subroutines of this type have been written. They are

LEVINT	for Levin's transforms
EPSRHO	for the ϱ-algorithm, the ε-algorithm and its generalizations.

  With each of this subroutines, it is impossible, in the same calling loop, to transform a sequence by several of the transformations contained in the same subroutine. This drawback can be avoided for the ϱ and ε-algorithms by using the subroutine EPSRHA.

  A similar subroutine can be written for Levin's transforms.

- The subroutines which are not directly connected with an extrapolation algorithm have a different use which is explained in each of them. They are ACCES, BLBORD, BORDER and INVLAP.

The main programs for using a subroutine implementing an extrapolation algorithm could be written according to the following pseudocode

1. **Specifications for variables and constants**
   list of integer variables and parameter constants
   list of real variables and parameter constants

2. **Specifications for parameter constants**
   MAXCOL, NBC, EPS, ...
3. **Specifications for vectors and matrices**
   list and dimensions of integer arrays
   list and dimensions of real arrays
4. **Initialization**
   INIT = 0
   other initializations required by the algorithm
5. **For $n = 0$ to NBC−1 do**
   computation of S_n
   computation of other quantities required by the algorithm
   CALL of the subroutine
   control on IER warning/error index
   print the intermediate results

end for

Let us now list the examples used for testing the various subroutines.

- ACCES
  For $n = 0, 1, \ldots$

$$\begin{aligned} S_n &= \frac{1}{n+1} \cdot \sin a(n+1) \\ D_n &= b\left(S_n^2 - S_n\right) \end{aligned}$$

  with $a = 1.0$ and $b = 2.0$.

- BLBORD

$$\begin{pmatrix} 1 & 1 & 1 & 1 & -1 & 0 & -1 \\ 1 & 1 & 2 & 0 & 1 & 1 & -1 \\ 1 & 1 & -1 & 0 & 2 & -2 & 0 \\ -1 & 1 & 2 & 0 & -1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{pmatrix} = \begin{pmatrix} -2 \\ 13 \\ -2 \\ 22 \\ 25 \\ 18 \\ 9 \end{pmatrix}$$

- BORDER
  Hilbert matrix of dimension n ($a_{i,j} = 1/(i+j-1)$) with the right hand side computed such that the solution is $x_i = 1$ for $i = 1, \ldots, n$.

- COMPOS
  For $n = 1, 2, \ldots$

$$S_n = \sin(an)/n$$

  with $a = 3.0$.

- CRPA
  For $n = 0, 1, \ldots$

$$(x_n)_j = \cos j \Big/ (n+1)^j$$

  with $(z_k)_j = \sin(kj)$ for $k = 1, 2, \ldots$ where $(a)_j$ is the j-th component of the vector a.

- EALGO
  For $n = 0, 1, \ldots$

$$\begin{aligned} S_n &= 1/(n+1) \\ g_i(n) &= \Delta S_{n+i-1}, \qquad i = 1, 2, \ldots \end{aligned}$$

- EPSRHA
  For $n = 1, 2, \ldots$

1^{st} sequence	$n^2 \cdot 0.8^n$	
2^{nd} sequence	$1/n$	$x_n = n$
3^{rd} sequence	$\sin\left(0.8 \cdot n^2\right) \cdot 0.9^{n^2}$	$x_n = n^2$
4^{th} sequence	$1/n^3$	$x_n = n$

- EPSRHO
  For $n = 1, 2, \ldots$

$$\begin{aligned} S_n &= 1/\ln^2(1+n) \\ x_n &= \ln(1+n) \end{aligned}$$

- EPSTOP
  For $n = 1, 2, \ldots$

$$\begin{aligned} y &= (1, \ldots, 1)^{\mathrm{T}} \\ (S_n)_j &= j \cdot 0.8^n + n \cdot 0.9^n \end{aligned}$$

  where $(S_n)_j$ designates the j-th component of the vector S_n.

- EPSVEC
  identical to the example for EPSTOP

- GTRAN
  For $n = 1, 2, \ldots$

$$\begin{aligned} S_n &= 1/n \\ x_n &= \Delta S_n \end{aligned}$$

- HALGO
  For $n = 1, 2, \ldots$

$$\begin{aligned} (S_n)_j &= 1/n^j \\ g_i(n) &= (\Delta S_n)_1 , \qquad i = 1, 2 \ldots \end{aligned}$$

  where $(\Delta S_n)_1$ is the first component of the vector ΔS_n.

- IDELTA
  For $n = 1, 2, \ldots$

$$S_n = 1/n$$

- INVLAP

$$\begin{aligned} F(p) &= \ln\left(1 + a^2 \,/ p^2\right) \\ p_i &= 0.1 + i \cdot h, \qquad\qquad i = 0, \ldots, 2k+1 \\ f(t) &= 2(1 - \cos at)/t \\ t &= -1.0 + (j-1) \cdot 0.4, \qquad j = 1, \ldots, 5 \end{aligned}$$

  with $a = 1.0, k = 3$ and $h = 2/(2k+1)$.

- LEVINT
  For $n = 0, 1, \ldots$

$$S_n = x - \frac{x^2}{2} + \cdots + (-1)^n \frac{x^{n+1}}{n+1}$$

  which converges to $\ln(1 + x)$.

- OMEGA
  For $n = 1, 2, \ldots$

$$S_n = 1/n$$

- OVERHO
  For $n = 1, 2, \ldots$

$$S_n = 1/n$$

- QDALGO
  For $n = 1, 2, \ldots$

$$x_n = (-1)^{n+1} 20.5\, n - 17.5\, (n + 1)$$

- RICHAR
  For $n = 1, 2, \ldots$

$$\begin{aligned} x_n &= 1/n \\ S_n &= x_n^5 \end{aligned}$$

- RSALGO
  Same example as for QDALGO

- SEALGO
  Same example as for HALGO

- SELECT
  Same example as for COMPOS

- THETA
  For $n = 1, 2, \ldots$

$$S_n = \ln(1 + S_{n-1}) \quad \text{with } S_0 = 1.$$

  This sequence tends to zero.

- VEALGO
  Same example as for HALGO

- VGTRAN
  For $n = 1, 2, \ldots$

$$S_{n+1} = S_{n-1}/2$$

with

$$S_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad S_1 = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

and

$$x_n = \sum_{j=1}^{3} (\Delta S_n)_j$$

BIBLIOGRAPHY

[1] J. Abadie. The GRG method for nonlinear programming. In H. Greenberg, ed., *Design and Implementation of Optimization Software*, Dordrecht, 1978. Noordhoff.

[2] J. Abaffy and E. Spedicato. *ABS Projection Algorithms.* Ellis Horwood, Chichester, 1989.

[3] N. Aboun. *Etude et Réalisation d'un Logiciel Vectorisé du GRG avec Estimation de la Précision de la Solution. Implementation de l'ε-Algorithme Vectoriel pour l'Accélération de la Convergence.* Thèse 3ème cycle, Université de Paris VI, 1985.

[4] M. Abramowitz and I. A. Stegun, eds. *Handbook of Mathematical Functions.* Dover, New-York, 1965.

[5] S. Achakir. *Connexion entre les Méthodes de Point Fixe et d'Accélération de la Convergence.* Thèse 3ème cycle, Université de Lille I, 1982.

[6] A. C. Aitken. On Bernoulli's numerical solution of algebraic equations. *Proc. R. Soc. Edinb.*, 46:289–305, 1926.

[7] N. C. Albertsen, G. Jacobsen, and S. B. Sørensen. Nonlinear transformations for accelerating the convergence of M-dimensional series. *Math. Comput.*, 41:623–634, 1983.

[8] R. Alt. A-stable one-step methods with step-size control for stiff systems of ordinary differential equations. *J. Comput. Appl. Math.*, 4:29–35, 1978.

[9] A. L. Andrew. Iterative computation of derivatives of eigenvalues and eigenvectors. *J. Inst. Maths. Appl.*, 24:209–218, 1979.

[10] T. A. Atchison and H. L. Gray. Nonlinear transformations related to the evaluation of improper integrals. II. *SIAM J. Numer. Anal.*, 5:451–459, 1968.

[11] G. A. Baker Jr. and P. R. Graves-Morris. *Padé Approximants.* Addison-Wesley, Reading, 1981. 2 vols.

[12] A. B. Bakushinskii. A numerical method for solving Fredholm integral equations of the first kind. *Zh. Vychisl. Mat. Mat. Fiz.*, 5(4):744–749, 1965.

[13] M. N. Barber and C. J. Hamer. Extrapolation of sequences using a generalized epsilon-algorithm. *J. Aust. Math. Soc., Ser. B*, 23:229–240, 1982.

[14] B. Baron and S. Wajc. Convergence acceleration of non-scalar sequences with non-linear transformations. *Math. Comput. Simulation*, 23:133–141, 1981.

[15] F. L. Bauer. La méthode d'intégration numérique de Romberg. In *Colloque sur l'Analyse Numérique*, pp. 119–129, Louvain, 1961. Librairie Universitaire.

[16] B. Beckermann. A connection between the E-algorithm and the epsilon algorithm. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 443–446, Basel, 1989. J. C. Baltzer.

[17] A. Beckman, B. Fornberg, and A. Tengvald. A method for acceleration of the convergence of infinite series. *BIT*, 9:78–80, 1969.

[18] J. M. Beguin, C. Gouriéroux, and A. Monfort. Identification of a mixed autoregressive moving average process: the corner method. In O. D. Andersen, ed., *Time Series*, pp. 423–436, Amsterdam, 1980. North-Holland.

[19] D. Belkić. New hybrid non-linear transformations of divergent perturbation series for quadratic Zeeman effects. *J. Phys. A: Math. Gen.*, 22:3003–3010, 1989.

[20] G. E. Bell and G. M. Phillips. Aitken acceleration of some alternating series. *BIT*, 24:70–77, 1984.

[21] M. Bellalij. *Procédés de Contrôle de l'Erreur et Accélération de la Convergence.* Thèse 3ème cycle, Université de Lille I, 1985.

[22] M. Bellalij. A simultaneous process for convergence acceleration and error control. *J. Comput. Appl. Math.*, 33:217–231, 1990.

[23] M. D. Benchiboun. *Etude de Certaines Généralisations du Δ^2 d'Aitken et Comparaison de Procédés d'Accélération de la Convergence.* Thèse 3ème cycle, Université de Lille I, 1987.

[24] M. D. Benchiboun. Some results on non–linear and non–logarithmic sequences and their acceleration. *Zastosow. Mat.* To appear.

[25] C. M. Bender and S. A. Orszag. *Advanced Mathematical Methods for Scientists and Engineers.* McGraw-Hill, New-York, 1978.

[26] A. Berlinet. Some useful algorithms in time series analysis. In J.P. Flourens et al., eds., *Alternative Approaches to Time Series Analysis*, pp. 95–120, Bruxelles, 1984. Publ. des Facultés Univ. St. Louis.

[27] A. Berlinet. Sequence transformations as statistical tools. *Appl. Numer. Math.*, 1:531–544, 1985.

[28] J. Beuneu. Some projection methods for nonlinear problems. Publication ANO 134, Université de Lille I, 1984.

[29] S. Bhowmick, R. Bhattacharya, and D. Roy. Iterations of convergence accelerating nonlinear transforms. *Comput. Phys. Comm.*, 54:31–46, 1989.

[30] P. Bjørstad, G. Dahlquist, and E. Grosse. Extrapolation of asymptotic expansions by a modified δ^2 formula. *BIT*, 21:56–65, 1981.

[31] K. O. Bowman and L. R. Shenton. *Properties of Estimators for the Gamma Distribution.* Marcel Dekker, New-York, 1988.

[32] K. O. Bowman and L. R. Shenton. *Continued Fractions in Statistical Analysis.* Marcel Dekker, New-York, 1989.

[33] C. Brezinski. Application de l'ε-algorithme à la résolution des systèmes non linéaires. *C. R. Acad. Sci. Paris*, 271 A:1174–1177, 1970.

[34] C. Brezinski. Application du ϱ-algorithme à la quadrature numérique. *C. R. Acad. Sci. Paris*, 270 A:1252–1253, 1970.

[35] C. Brezinski. Résultats sur les procédés de sommation et sur l'ε-algorithme. *RIRO*, R3:147–153, 1970.

[36] C. Brezinski. Accélération de suites à convergence logarithmique. *C. R. Acad. Sci. Paris*, 273 A:727–730, 1971.

[37] C. Brezinski. Comparaison de suites convergentes. *RIRO*, R2:95–99, 1971.

[38] C. Brezinski. Convergence d'une forme confluente de l'ε-algorithme. *C. R. Acad. Sci. Paris*, 273 A:582–585, 1971.

[39] C. Brezinski. Etudes sur les ε et ϱ-algorithmes. *Numer. Math.*, 17:153–162, 1971.

[40] C. Brezinski. *Méthodes d'Accélération de la Convergence en Analyse Numérique.* Thèse d'Etat, Université de Grenoble, 1971.

[41] C. Brezinski. Sur un algorithme de résolution des systèmes non linéaires. *C. R. Acad. Sci. Paris*, 272 A:145–148, 1971.

[42] C. Brezinski. Transformation rationnelle d'une fonction. *C. R. Acad. Sci. Paris*, 273 A:772–774, 1971.

[43] C. Brezinski. Conditions d'application et de convergence de procédés d'extrapolation. *Numer. Math.*, 20:64–79, 1972.

[44] C. Brezinski. Généralisation des extrapolations polynomiales et rationnelles. *RAIRO*, R1:61–66, 1972.

[45] C. Brezinski. L'ε-algorithme et les suites totalement monotones et oscillantes. *C. R. Acad. Sci. Paris*, 276 A:305–308, 1973.

[46] C. Brezinski. Accélération de la convergence de suites dans un espace de Banach. *C. R. Acad. Sci. Paris*, 278 A:351–354, 1974.

[47] C. Brezinski. Intégration des systèmes différentiels à l'aide du ϱ-algorithme. *C. R. Acad. Sci. Paris*, 278 A:875–878, 1974.

[48] C. Brezinski. Some results in the theory of the vector ε-algorithm. *Linear Algebra Appl.*, 8:77–86, 1974.

[49] C. Brezinski. Computation of the eigenelements of a matrix by the ε-algorithm. *Linear Algebra Appl.*, 11:7–20, 1975.

[50] C. Brezinski. Forme confluente de l'ε-algorithme topologique. *Numer. Math.*, 23:363–370, 1975.

[51] C. Brezinski. Généralisation de la transformation de Shanks, de la table de Padé et de l'ε-algorithme. *Calcolo*, 12:317–360, 1975.

[52] C. Brezinski. Génération de suites totalement monotones et oscillantes. *C. R. Acad. Sci. Paris*, 280 A:729–731, 1975.

[53] C. Brezinski. Numerical stability of a quadratic method for solving systems of nonlinear equations. *Computing*, 14:205–211, 1975.

[54] C. Brezinski. Computation of Padé approximants and continued fractions. *J. Comput. Appl. Math.*, 2:113–123, 1976.

[55] C. Brezinski. *Accélération de la Convergence en Analyse Numérique*, volume 584 of *LNM*. Springer-Verlag, Berlin, 1977.

[56] C. Brezinski. *Algorithmes d'Accélération de la Convergence. Etude Numérique.* Editions Technip, Paris, 1978.

[57] C. Brezinski. Convergence acceleration of some sequences by the ε-algorithm. *Numer. Math.*, 29:173–177, 1978.

[58] C. Brezinski. Limiting relationships and comparison theorems for sequences. *Rend. Circ. Mat. Palermo, II. Ser.*, 28:273–280, 1979.

[59] C. Brezinski. Programmes FORTRAN pour transformations de suites et de séries. Publication ANO 3, Université de Lille I, 1979.

[60] C. Brezinski. Sur le calcul de certains rapports de déterminants. In L. Wuytack, ed., *Padé Approximation and its Applications*, volume 765 of *LNM*, pp. 184–210, Berlin, 1979. Springer-Verlag.

[61] C. Brezinski. A general extrapolation algorithm. *Numer. Math.*, 35:175–187, 1980.

[62] C. Brezinski. The Mühlbach-Neville-Aitken algorithm and some extensions. *BIT*, 20:444–451, 1980.

[63] C. Brezinski. *Padé-Type Approximation and General Orthogonal Polynomials*, volume 50 of *ISNM*. Birkhäuser-Verlag, Basel, 1980.

[64] C. Brezinski. Algorithm 585: A subroutine for the general interpolation and extrapolation problems. *ACM Transactions on Mathematical Software*, 8:290–301, 1982.

[65] C. Brezinski. Some new convergence acceleration methods. *Math. Comput.*, 39:133–145, 1982.

[66] C. Brezinski. About Henrici's method for nonlinear equations. Symposium on Numerical Analysis and Computational Complex Analysis, Zürich, unpublished, August 1983.

[67] C. Brezinski. Error control in convergence acceleration processes. *IMA J. Numer. Anal.*, 3:65–80, 1983.

[68] C. Brezinski. Recursive interpolation, extrapolation and projection. *J. Comput. Appl. Math.*, 9:369–376, 1983.

[69] C. Brezinski. Some determinantal identities in a vector space, with applications. In H. Werner and H. J. Bünger, eds., *Padé Approximation and its Applications. Bad-Honnef 1983*, volume 1071 of *LNM*, pp. 1–11, Berlin, 1984. Springer-Verlag.

[70] C. Brezinski. Composite sequence transformations. *Numer. Math.*, 46:311–321, 1985.

[71] C. Brezinski. Prediction properties of some extrapolation methods. *Appl. Numer. Math.*, 1:457–462, 1985.

[72] C. Brezinski. Vitesse de convergence d'une suite. *Rev. Roum. Math. Pures Appl.*, 30:403–417, 1985.

[73] C. Brezinski. How to accelerate continued fractions. In P. Chenin et al., eds., *Computer and Computing*, pp. 35–39, Paris, 1986. Masson.

[74] C. Brezinski. Orthogonal and Stieltjes polynomials with respect to an even functional. *Rend. Semin. Mat. Torino*, 45:75–82, 1987.

[75] C. Brezinski. Successive modifications of limit periodic continued fractions. *J. Comput. Appl. Math.*, 19:67–74, 1987.

[76] C. Brezinski. Error estimate in Padé approximation. In M. Alfaro et al., eds., *Orthogonal Polynomials and their Applications*, volume 1329 of *LNM*, pp. 1–19, Berlin, 1988. Springer-Verlag.

[77] C. Brezinski. A new approach to convergence acceleration methods. In A. Cuyt, ed., *Nonlinear Numerical Methods and Rational Approximation*, pp. 373–405, Dordrecht, 1988. Reidel.

[78] C. Brezinski. On the asymptotic behaviour of continued fractions. *Appl. Numer. Math.*, 4:231–239, 1988.

[79] C. Brezinski. Other manifestations of the Schur complement. *Linear Algebra Appl.*, 111:231–247, 1988.

[80] C. Brezinski. Partial Padé approximation. *J. Approximation Theory*, 54:210–233, 1988.

[81] C. Brezinski. Quasi-linear extrapolation processes. In R.P. Agarwal et al., eds., *Numerical Mathematics. Singapore 1988*, volume 86 of *ISNM*, pp. 61–78, Basel, 1988. Birkhäuser-Verlag.

[82] C. Brezinski. Contraction properties of sequence transformations. *Numer. Math.*, 54:565–574, 1989.

[83] C. Brezinski. Procedures for estimating the error in Padé approximation. *Math. Comput.*, 53:639–648, 1989.

[84] C. Brezinski. A survey of iterative extrapolation by the E-algorithm. *Det Kong. Norske Vid. Selsk. Skr.*, 2:1–26, 1989.

[85] C. Brezinski. Algebraic properties of the E-transformation. In *Numerical Analysis and Mathematical Modelling*, volume 24 of *PWN*, pp. 85–90, Warsaw, 1990. Banach Center Publications.

[86] C. Brezinski. Bordering methods and progressive forms for sequence transformations. *Zastosow. Mat.*, 20:435–443, 1990.

[87] C. Brezinski. *History of Continued Fractions and Padé Approximants.* Springer-Verlag, Berlin, 1990.

[88] C. Brezinski. *A Bibliography on Continued Fractions, Padé Approximation, Extrapolation and Related Subjects.* Prensas Universitarias de Zaragoza, Zaragoza, 1991.

[89] C. Brezinski. *Biorthogonality and its Applications to Numerical Analysis.* Marcel Dekker, New-York, 1991.

[90] C. Brezinski. Implementing the jackknife. *Appl. Comput. Math.*, 42:111–119, 1991.

[91] C. Brezinski. The asymptotic behaviour of sequences and new series transformations based on the Cauchy product. *Rocky Mt. J. Math.*, 21, 1991. To appear.

[92] C. Brezinski and M. Crouzeix. Remarques sur le procédé Δ^2 d'Aitken. *C. R. Acad. Sci. Paris*, 270 A:896–898, 1970.

[93] C. Brezinski, J. P. Delahaye, and B. Germain-Bonne. Convergence acceleration by extraction of linear subsequences. *SIAM J. Numer. Anal.*, 20:1099–1105, 1983.

[94] C. Brezinski and A. Lembarki. Acceleration of extended Fibonacci sequences. *Appl. Numer. Math.*, 2:1–8, 1986.

[95] C. Brezinski and A. Lembarki. The linear convergence of limit periodic continued fractions. *J. Comput. Appl. Math.*, 19:75–77, 1987.

[96] C. Brezinski and S. Paszkowski. Optimal linear contractive sequence transformations. *J. Comput. Appl. Math.* To appear.

[97] C. Brezinski and M. Redivo Zaglia. Construction of extrapolation processes. *Appl. Numer. Math.*, 8, 1991. To appear.

[98] C. Brezinski, M. Redivo Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos type algorithms. *Numer. Algorithms*, 1, 1991. To appear.

[99] C. Brezinski, M. Redivo Zaglia, and H. Sadok. A breakdown-free Lanczos type algorithm for solving linear systems. *Numer. Math.* To appear.

[100] C. Brezinski and A. C. Rieu. The solution of systems of equations using the vector ε-algorithm, and an application to boundary value problems. *Math. Comput.*, 28:731–741, 1974.

[101] C. Brezinski and H. Sadok. Vector sequence transformations and fixed point methods. In C. Taylor et al., eds., *Numerical Methods in Laminar and Turbulent Flows*, pp. 3–11, Swansea, 1987. Pineridge Press.

[102] C. Brezinski and J. van Iseghem. Padé-type approximants and linear functional transformations. In P. R. Graves-Morris et al., eds., *Rational Approximation and Interpolation*, volume 1105 of *LNM*, pp. 100–108, Berlin, 1984. Springer-Verlag.

[103] C. Brezinski and J. van Iseghem. Padé approximations. In P. G. Ciarlet and J. L. Lions, eds., *Handbook of Numerical Analysis*, volume 3, Amsterdam, 1992. North-Holland.

[104] C. Brezinski and G. Walz. Sequences of transformations and triangular recursion schemes with applications in numerical analysis. *J. Comput. Appl. Math.*, 34:361–383, 1991.

[105] R. A. Brualdi and H. Schneider. Determinantal identities: Gauss, Schur, Cauchy, Sylvester, Kronecker, Jacobi, Binet, Laplace, Muir and Cayley. *Linear Algebra Appl.*, 52-53:769–791, 1983.

[106] R. Bulirsch and J. Stoer. Fehlerabschätzungen und Extrapolation mit rationalen Funktionen bei Verfahren vom Richardson-Typus. *Numer. Math.*, 6:413–427, 1964.

[107] R. Bulirsch and J. Stoer. Numerical treatment of ordinary differential equations by extrapolation. *Numer. Math.*, 8:1–13, 1966.

[108] R. Bulirsch and J. Stoer. Numerical quadrature by extrapolation. *Numer. Math.*, 9:271–278, 1967.

[109] S. Cabay and L. W. Jackson. A polynomial extrapolation method for finding limits and antilimits of vector sequences. *SIAM J. Numer. Anal.*, 13:734–752, 1976.

[110] J. Caldwell. An application of extrapolation to the limit. *BIT*, 20:251–253, 1980.

[111] J. R. Cannon and M. Morandi Cecchi. Numerical experiments on the solution of some biharmonic problems by mathematical programming techniques. *SIAM J. Numer. Anal.*, 4:147–154, 1967.

[112] C. Carstensen. On a general epsilon algorithm. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 437–441, Basel, 1989. J. C. Baltzer.

[113] C. Carstensen. On a general rho-algorithm. *J. Comput. Appl. Math.*, 33:61–71, 1990.

[114] E. Christiansen and H. G. Petersen. Estimation of convergence orders in repeated Richardson extrapolation. *BIT*, 29:48–59, 1989.

[115] P. G. Ciarlet. *Introduction à l'Analyse Numérique Matricielle et à l'Optimisation.* Masson, Paris, 1982.

[116] G. Claessens, G. Loizou, and L. Wuytack. Comments on a root finding method using Padé approximation. *BIT*, 17:360–361, 1977.

[117] F. Cordellier. Caractérisation des suites que la première étape du θ-algorithme transforme en suites constantes. *C. R. Acad. Sci. Paris*, 284 A:389–392, 1977.

[118] F. Cordellier. Particular rules for the vector ε-algorithm. *Numer. Math.*, 27:203–207, 1977.

[119] F. Cordellier. Démonstration algébrique de l'extension de l'identité de Wynn aux tables de Padé non normales. In L. Wuytack, ed., *Padé Approximation and its Applications*, volume 765 of *LNM*, pp. 36–60, Berlin, 1979. Springer-Verlag.

[120] F. Cordellier. Utilisation de l'invariance homographique dans les algorithmes de losange. In H. Werner and H. J. Bünger, eds., *Padé Approximation and its Applications. Bad-Honnef 1983*, volume 1071 of *LNM*, pp. 62–94, Berlin, 1984. Springer-Verlag.

[121] F. Cordellier. *Interpolation Rationnelle et Autres Questions: Aspects Algorithmiques et Numériques.* Thèse d'Etat, Université de Lille I, 1989.

[122] A. Croft. Convergence acceleration of linear second-order two-point boundary value problems on a semi-infinite domain. *Appl. Numer. Math.* To appear.

[123] M. Crouzeix and A. L. Mignot. *Analyse Numérique des Equations Différentielles.* Masson, Paris, 1989. 2nd ed.

[124] C. W. Cryer. Numerical methods for free and moving boundary problems. In A. Iserles and M. J. D. Powell, eds., *The State of the Art in Numerical Analysis*, pp. 601–622, Oxford, 1987. Clarendon Press.

[125] A. Cuyt. Accelerating the convergence of a table with multiple entry. *Numer. Math.*, 41:281–286, 1983.

[126] A. Cuyt. *General Order Multivariate Rational Hermite Interpolants.* Habilitation, University of Antwerp, 1986.

[127] A. Cuyt. A recursive computation scheme for multivariate rational interpolants. *SIAM J. Numer. Anal.*, 24:228–239, 1987.

[128] A. Cuyt. Old and new multidimensional convergence accelerators. *Appl. Numer. Math.*, 6:169–185, 1989-1990.

[129] A. Cuyt and B. Verdonk. Different techniques for the construction of multivariate rational interpolants. In A. Cuyt, ed., *Nonlinear Numerical Methods and Rational Approximation*, pp. 167–190, Dordrecht, 1988. Reidel.

[130] A. Cuyt and B. Verdonk. Rational interpolation on general data sets in $\mathbf{C}^n$. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 415–429, Basel, 1989. J. C. Baltzer.

[131] A. Cuyt and L. Wuytack. *Nonlinear Methods in Numerical Analysis.* North-Holland, Amsterdam, 1987.

[132] E. De Donker-Kapenga. Asymptotic expansions and their applications in numerical integration. In P. Keast and G. Fairweather, eds., *Numerical Integration*, pp. 141–151, Dordrecht, 1987. Reidel.

[133] J. P. Delahaye. Détermination de la période d'une suite pseudo-périodique. *Bull. Dir. Etud. Rech., Sér. C*, 1:65–79, 1980.

[134] J. P. Delahaye. Liens entre la suite du rapport des erreurs et celle du rapport des différences. *C. R. Acad. Sci. Paris*, 290 A:343–346, 1980.

[135] J. P. Delahaye. Accélération de la convergence des suites dont le rapport des erreurs est borné. *Calcolo*, 18:104–116, 1981.

[136] J. P. Delahaye. Automatic selection of sequence transformations. *Math. Comput.*, 37:197–204, 1981.

[137] J. P. Delahaye. Optimalité du procédé Δ^2 d'Aitken pour l'accélération de la convergence linéaire. *RAIRO, Anal. Numér.*, 15:321–330, 1981.

[138] J. P. Delahaye. *Sequence Transformations.* Springer-Verlag, Berlin, 1988.

[139] J. P. Delahaye and B. Germain-Bonne. Résultats négatifs en accélération de la convergence. *Numer. Math.*, 35:443–457, 1980.

[140] J. P. Delahaye and B. Germain-Bonne. The set of logarithmically convergent sequences cannot be accelerated. *SIAM J. Numer. Anal.*, 19:840–844, 1982.

[141] J. Della Dora. *Contribution à l'Approximation de Fonctions de la Variable Complexe au Sens de Hermite-Padé et de Hardy.* Thèse d'Etat, Université de Grenoble, 1980.

[142] P. Deuflhard. Recent progress in extrapolation methods for ordinary differential equations. *SIAM Rev.*, 27:505–535, 1985.

[143] J. Dieudonné. *Fondements de l'Analyse Moderne.* Gauthier-Villars, Paris, 1968. Tome 1.

[144] A. Draux. Convergence acceleration of sequences of Padé approximants. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 459–462, Basel, 1989. J. C. Baltzer.

[145] A. Draux. On composite sequence transformations. *Appl. Numer. Math.*, 7:217–226, 1991.

[146] A. Draux and P. van Ingelandt. *Polynômes Orthogonaux et Approximants de Padé. Logiciels.* Editions Technip, Paris, 1987.

[147] J. E. Drummond. A formula for accelerating the convergence of a general series. *Bull. Aust. Math. Soc.*, 6:69–74, 1972.

[148] J. E. Drummond. Summing a common type of slowly convergent series of positive terms. *J. Aust. Math. Soc., Ser. B*, 19:416–421, 1976.

[149] R. P. Eddy. Extrapolation to the limit of a vector sequence. In P. C. C. Wang, ed., *Information Linkage Between Applied Mathematics and Industry*, pp. 387–396, New-York, 1979. Academic Press.

[150] B. Efron. *The Jackknife, the Boostrap and Other Resampling Plans.* SIAM, Philadelphia, 1982.

[151] H. W. Engl. Regularization of linear and nonlinear ill-posed problems: convergence rates. *J. Comput. Appl. Math.* To appear.

[152] C. Espinoza. *Contribution à la Résolution Numérique de Certains Systèmes d'Equations.* Thèse 3ème cycle, Université de Grenoble, 1977.

[153] E. Fabry. *Théorie des Séries à Termes Constants.* Hermann, Paris, 1910.

[154] V. N. Faddeeva. *Computational Methods of Linear Algebra.* Dover, New-York, 1959.

[155] Fang Chen. To appear.

[156] A. Fdil. Some new results of convergence acceleration for the E-algorithm. *Zastosow. Mat.* To appear.

[157] T. Fessler, W. F. Ford, and D. A. Smith. Algorithm 602, HURRY: An acceleration algorithm for scalar sequences and series. *ACM Trans. Math. Software*, 9:355–357, 1983.

[158] T. Fessler, W. F. Ford, and D. A. Smith. HURRY: An acceleration algorithm for scalar sequences and series. *ACM Trans. Math. Software*, 9:346–354, 1983.

[159] R. Fletcher. Conjugate gradient methods for indefinite systems. In *Numerical Analysis. Dundee 1975*, volume 506 of *LNM*, pp. 73–89, Berlin, 1976. Springer-Verlag.

[160] R. Fletcher. *Practical Methods of Optimization.* Wiley, Chicester, 1987. 2nd edition.

[161] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Comput. J.*, 7:149–154, 1964.

[162] W. F. Ford and A. Sidi. An algorithm for a generalization of the Richardson extrapolation process. *SIAM J. Numer. Anal.*, 24:1212–1232, 1987.

[163] W. F. Ford and A. Sidi. Recursive algorithms for vector extrapolation methods. *Appl. Numer. Math.*, 4:477–489, 1988.

[164] Y. Fouquart. Utilisation des approximants de Padé pour l'étude des largeurs équivalentes des raies formées en atmosphère diffusante. *J. Quant. Spectrosc. Radiat. Transfert*, 14:497–508, 1974.

[165] L. Fox. Romberg integration for a class of singular integrands. *Comput. J.*, 10:87–93, 1967.

[166] B. Gabutti. *On a New Series Acceleration Method Arising in the Integration of Strongly Oscillating Functions.* Libreria Editrice Universitaria Levrotto e Bella, Torino, 1981.

[167] B. Gabutti. On two methods for accelerating convergence of series. *Numer. Math.*, 43:439–461, 1984.

[168] B. Gabutti. An algorithm for computing generalized Euler's transformations of series. *Computing*, 34:107–116, 1985.

[169] B. Gabutti and J. N. Lyness. An acceleration method for the power series of entire functions of order 1. *Math. Comput.*, 39:587–597, 1982.

[170] B. Gabutti and J. N. Lyness. Some generalizations of the Euler-Knopp transformation. *Numer. Math.*, 48:199–220, 1986.

[171] W. Gander and G. H. Golub. Discussion of Buja, Hastie and Tibshirani: Linear smoothers and additive models. *Ann. Stat.*, 17:529–532, 1989.

[172] W. Gander, G. H. Golub, and D. Gruntz. Solving linear equations by extrapolation. Manuscript NA–89–11, Computer Science Department, Stanford University, 1989.

[173] C. R. Garibotti and F. F. Grinstein. Recent results relevant to the evaluation of infinite series. *J. Comput. Appl. Math.*, 9:193–200, 1983.

[174] M. Gasca and E. Lebron. On Aitken-Neville formulæ for multivariate interpolation. In E. L. Ortiz, ed., *Numerical Approximation of Partial Differential Equations*, Amsterdam, 1987. Elsevier.

[175] M. Gasca and G. Mühlbach. Multivariate interpolation: a survey with regard to extrapolation. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 431–436, Basel, 1989. J. C. Baltzer.

[176] N. Gastinel. Conditionnement des problèmes d'approximation (aux moindres carrés) et des problèmes de lissage. In *Programmation en Mathématiques Numériques*, pp. 111–118, Paris, 1968. CNRS.

[177] E. Gekeler. On the solution of systems of equations by the epsilon algorithm of Wynn. *Math. Comput.*, 26:427–436, 1972.

[178] A. C. Genz. Applications of the ε-algorithm to quadrature problems. In P. R. Graves-Morris, ed., *Padé Approximants and their Applications*, pp. 105–116, New-York, 1973. Academic Press.

[179] A. C. Genz. Some extrapolation methods for the numerical calculation of multidimensional integrals. In D. J. Evans, ed., *Software for Numerical Mathematics*, pp. 159–172, New-York, 1974. Academic Press.

[180] A. C. Genz. *The Approximate Calculation of Multidimensional Integrals Using Extrapolation Methods.* PhD thesis, University of Kent, Canterbury, 1975.

[181] B. Germain-Bonne. Transformations de suites. *RAIRO*, R1:84–90, 1973.

[182] B. Germain-Bonne. *Estimation de la Limite de Suites et Formalisation de Procédés d'Accélération de la Convergence.* Thèse d'Etat, Université de Lille I, 1978.

[183] B. Germain-Bonne. Convergence acceleration of number–machine sequences. *J. Comput. Appl. Math.*, 32:83–88, 1990.

[184] B. Germain-Bonne and C. Kowalewski. Accélération de la convergence par utilisation d'une transformation auxiliaire. Publication ANO 77, Université de Lille I, 1982.

[185] B. Germain-Bonne and A. M. Litovsky. Sufficient information for accelerating the convergence. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 453–458, Basel, 1989. J. C. Baltzer.

[186] J. Gilewicz. Numerical detection of the best Padé approximant and determination of the Fourier coefficients of insufficiently sampled functions. In P. R. Graves-Morris, ed., *Padé Approximants and their Applications*, pp. 99–103, New-York, 1973. Academic Press.

[187] G. H. Golub and C. F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, 1989. 2nd edition.

[188] M. Graça. On curvature-type transformations. To appear.

[189] W. B. Gragg. *Repeated Extrapolation to the Limit in the Numerical Solution of Ordinary Differential Equations.* PhD thesis, UCLA, 1963.

[190] W. B. Gragg. On extrapolation algorithms for ordinary initial value problems. *SIAM J. Numer. Anal.*, 2:384–403, 1965. ser. B.

[191] N. Grant Cooper, ed. *From Cardinals to Chaos. Reflections on the Life and Legacy of Stanislaw Ulam.* Cambridge University Press, Cambridge, 1989.

[192] P. R. Graves-Morris. Vector valued rational interpolants I. *Numer. Math.*, 42:331–348, 1983.

[193] P. R. Graves-Morris. Extrapolation methods for vector sequences. *Numer. Math.* To appear.

[194] H. L. Gray. On a unification of bias reduction and numerical approximation. In J. N. Srivastava, ed., *Probability and Statistics*, pp. 105–116, Amsterdam, 1988. North-Holland.

[195] H. L. Gray and T. A. Atchison. Nonlinear transformations related to the evaluation of certain improper integrals. *SIAM J. Numer. Anal.*, 4:363–371, 1967.

[196] H. L. Gray and T. A. Atchison. The generalized G-transform. *Math. Comput.*, 22:595–606, 1968.

[197] H. L. Gray, T. A. Atchison, and G. V. McWilliams. Higher order G-transformations. *SIAM J. Numer. Anal.*, 8:365–381, 1971.

[198] H. L. Gray and W. D. Clark. On a class of nonlinear transformations and their applications to the evaluation of infinite series. *J. Res. Natl. Bur. Stand., B-Math. Sci.*, 73 B:251–274, 1969.

[199] H. L. Gray and W. R. Schucany. *The Generalized Jackknife Statistics.* Marcel Dekker, New-York, 1972.

[200] H. L. Gray and Suojin Wang. An extension of the Levin-Sidi class of nonlinear transformations for accelerating convergence of infinite integrals and series. *Appl. Math. Comput.*, 33:75–87, 1989.

[201] J. Grotendorst. Maple programs for converting series expansions to rational functions using the Levin transformation. *Comput. Phys. Commun.*, 55:325–335, 1989.

[202] J. Grotendorst. Approximating functions by means of symbolic computation and a general extrapolation method. *Comput. Phys. Commun.*, 59:289–301, 1990.

[203] J. Grotendorst and E. O. Steinborn. Use of nonlinear convergence accelerators for the efficient evaluation of GTO molecular integrals. *J. Chem. Phys.*, 84:5617–5623, 1986.

[204] J. Grotendorst, E. J. Weniger, and E. O. Steinborn. Efficient evaluation of infinite–series representation for overlap, two–center

nuclear attraction, and Coulomb integrals using nonlinear convergence accelerators. *Phys. Rev. A*, 33:3706–3726, 1986.

[205] S. Å. Gustafson. Convergence acceleration by means of numerical quadrature. *BIT*, 6:117–128, 1966.

[206] S. Å. Gustafson. A method of computing limit values. *SIAM J. Numer. Anal.*, 10:1080–1090, 1973.

[207] A. J. Guttmann. Asymptotic analysis of power-series expansions. *Phase Transitions*, 13:1–234, 1989.

[208] I. Haccart. *Accélération de la Convergence de Suites, Séries et Intégrales Doubles.* Thèse 3ème cycle, Université de Lille I, 1983.

[209] W. Hackbusch. *Multi-Grid Methods and Applications.* Springer-Verlag, Berlin, 1985.

[210] M. M. Hafez and H. K. Cheng. Convergence acceleration of relaxation solutions for transonic flow computations. *AIAA J.*, 15:329–336, 1977.

[211] M. M. Hafez, S. Palaniswamy, G. Kuruvila, and M. D. Salas. Applications of Wynn's ε-algorithm to transonic flow calculation. *AIAA J.*, pp. 458–469, 1987.

[212] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods.* Wiley, New-York, 1964.

[213] T. Hasegawa and T. Torii. Indefinite integration of oscillatory functions by the Chebyshev series expansion. *J. Comput. Appl. Math.*, 17:21–29, 1987.

[214] J. Haslinger and P. Neittaanmäki. *Finite Element Approximation for Optimal Shape Design.* Wiley, Chichester, 1988.

[215] A. Hautot. Convergence acceleration of continued fractions of Poincaré's type. *Appl. Numer. Math.*, 4:309–322, 1988.

[216] T. Håvie. On a modification of Romberg's algorithm. *BIT*, 6:24–30, 1966.

[217] T. Håvie. On the practical application of the modified Romberg algorithm. *BIT*, 7:103–113, 1967.

[218] T. Håvie. Derivation of explicit expressions for the error terms in the ordinary and the modified Romberg algorithms. *BIT*, 9:18–29, 1969.

[219] T. Håvie. Generalized Neville type extrapolation schemes. *BIT*, 19:204–213, 1979.

[220] T. Håvie. Aitken/Neville type algorithms for iterative interpolation/extrapolation using generalized rational functions. Math. and Comput., University of Trondheim, Norway, 1982.

[221] T. Håvie. An algorithm for iterative interpolation/extrapolation using generalized rational functions. Math. and Comput., University of Trondheim, Norway, 1982.

[222] T. Håvie. Two algorithms for iterative interpolation and extrapolation using generalized rational functions. Math. and Comput., University of Trondheim, Norway, 1983.

[223] T. Håvie and M. J. D. Powell. An application of Gaussian elimination to interpolation by generalized rational functions. In P. R. Graves-Morris et al., eds., *Rational Approximation and Interpolation*, volume 1105 of *LNM*, pp. 442–452, Berlin, 1984. Springer-Verlag.

[224] T. Håvie and H. H. Simonsen. An extrapolation algorithm for the numerical approximation of Cauchy principal value integrals. Math. and Comput., University of Trondheim, Norway, 1986.

[225] P. Henrici. *Elements of Numerical Analysis.* Wiley, New-York, 1964.

[226] P. Henrici. *Applied and Computational Complex Analysis.* Wiley, New-York, 1974.

[227] D. Herceg, R. Vulanović, and N. Petrović. Higher order schemes and Richardson extrapolation for singular perturbation problems. *Bull. Aust. Math. Soc.*, 39:129–139, 1989.

[228] R. L. Higgins. *Topics in the Application of Summation Methods.* PhD thesis, Drexel University, Philadelphia, 1976.

[229] P. Hillion. Intervalle d'approximation pour les suites à quotients de différences successives monotones. *C. R. Acad. Sci. Paris*, 277 A:853–856, 1973.

[230] M. Hollosi and P. Keast. Numerical integration of singular functions over a triangle. Mathematics, statistics and computing science, Dalhousie University, 1985.

[231] D. B. Hunter. Romberg's method for certain integrals involving a singularity. *BIT*, 7:200–205, 1967.

[232] D. B. Hunter. Some Gauss-type formulæ for the evaluation of Cauchy principal values of integrals. *Numer. Math.*, 19:419–424, 1972.

[233] D. B. Hunter. The numerical evaluation of Cauchy principal values of integrals by Romberg integration. *Numer. Math.*, 21:185–192, 1973.

[234] K. Iguchi. On the Aitken's δ^2-process. *Inform. Process. Japan*, 15:36–40, 1975.

[235] K. Iguchi. An algorithm of an acceleration process covering the Aitken's δ^2-process. *Inform. Process. Japan*, 16:89–93, 1976.

[236] K. Iguchi. Convergence property of Aitken's δ^2-process and the applicable acceleration process. *J. Inform. Process.*, 7:22–30, 1984.

[237] B. M. Irons and R. C. Tuck. A version of the Aitken accelerator for computer iteration. *Int. J. Numer. Methods Eng.*, 1:275–277, 1969.

[238] A. Iserles. Complex dynamics of convergence acceleration. *IMA J. Numer. Anal.*, 11:205–240, 1991.

[239] L. Jacobsen. A method for convergence acceleration of continued fractions $K(a_n/1)$. In W. B. Jones et al., eds., *Analytic Theory of Continued Fractions*, volume 932 of *LNM*, pp. 74–86, Berlin, 1982. Springer-Verlag.

[240] L. Jacobsen. Remarks to a definition of convergence acceleration illustrated by means of continued fractions $K(a_n/1)$ where $a_n \to 0$. *J. Comput. Appl. Math.*, 21:101–110, 1988.

[241] L. Jacobsen and H. Waadeland. Convergence acceleration of limit periodic continued fractions under asymptotic side conditions. *Numer. Math.*, 53:285–298, 1988.

[242] L. Jacobsen and H. Waadeland. An asymptotic property for tails of limit periodic continued fractions. *Rocky Mt. J. Math.*, 21, 1991. To appear.

[243] M. J. Jamieson and T. H. O'Beirne. A note on the generalization of Aitken's δ^2-transformation. *J. Phys. B: Atom. Molec. Phys.*, 11:L 31–L 35, 1978.

[244] K. Jbilou. *Méthodes d'Extrapolation et de Projection. Applications aux Suites de Vecteurs.* Thèse 3ème cycle, Université de Lille I, 1988.

[245] K. Jbilou and H. Sadok. Some results about vector extrapolation methods and related fixed point iterations. *J. Comput. Appl. Math.* To appear.

[246] A. Jennings. Accelerating the convergence of matrix iterative processes. *J. Inst. Maths Applics*, 8:99–110, 1971.

[247] B. Jones. Extrapolation for higher orders of convergence. *J. Inst. Maths Applics*, 17:27–36, 1976.

[248] B. Jones. A note on Aitken's δ^2 technique. *SIGNUM Newsletter*, 17:23, 1982.

[249] B. Jones. A note on the T_{+m} transformation. *Nonlin. Anal., Theory Methods Appl.*, 6:303–305, 1982.

[250] W. B. Jones and W. J. Thron. *Continued Fractions. Analytic Theory and Applications.* Addison-Wesley, Reading, 1980.

[251] W. B. Jones and W. J. Thron. Continued fractions in numerical analysis. *Appl. Numer. Math.*, 4:143–230, 1988.

[252] W. Joubert. Lanczos methods for the solution of nonsymmetric systems of linear equations. *SIAM J.* To appear.

[253] D. C. Joyce. Survey of extrapolation processes in numerical analysis. *SIAM Rev.*, 13:435–490, 1971.

[254] D. K. Kahaner. Numerical quadrature by the ϵ-algorithm. *Math. Comput.*, 26:689–693, 1972.

[255] S. Kaniel and J. Stein. Least-square acceleration of iterative methods for linear equations. *J. Optimization Theory Appl.*, 14:431–437, 1974.

[256] G. Karrholm. A method of iteration applied to beams resting on springs. Trans. Chalmers Univ. of Technology, 199, Göteborg, Sweden, 1958.

[257] M. Kateb. *Itérations d'Algorithmes d'Accélération de la Convergence.* Thèse 3ème cycle, Université de Lille I, 1983.

[258] T. A. Keagy and W. F. Ford. Acceleration by subsequence transformations. *Pac. J. Math.*, 132:357–362, 1988.

[259] H. B. Keller. Global homotopies and Newton methods. In C. de Boor and G. H. Golub, eds., *Recent Advances in Numerical Analysis*, pp. 73–94, New-York, 1978. Academic Press.

[260] J. E. Kiefer and G. H. Weiss. A comparison of two methods for accelerating the convergence of Fourier series. *Comput. Math. Appl.*, 7:527–535, 1981.

[261] J. P. Killingbeck. Microcomputer calculations in physics. *Rep. Prog. Phys.*, R48:54–99, 1985.

[262] R. F. King. An efficient one-point extrapolation method for linear convergence. *Math. Comput.*, 35:1285–1290, 1980.

[263] R. J. Kooman. *Convergence Properties of Recurrence Sequences.* Dissertation, Rijksuniversiteit Leiden, 1989.

[264] R. J. Kooman. Private communication, letter dated January 19, 1990.

[265] R. J. Kooman and R. Tijdeman. Convergence properties of linear recurrence sequences. *Nieuw. Arch. Wiskd.*, 8:13–25, 1991.

[266] C. Kowalewski. Accélération de la convergence pour certaines suites à convergence logarithmique. In M. G. de Bruin and H. van Rossum, eds., *Padé Approximation and its Applications*, volume 888 of *LNM*, pp. 263–272, Berlin, 1981. Springer-Verlag.

[267] C. Kowalewski. *Possibilités d'Accélération de la Convergence Logarithmique.* Thèse 3ème cycle, Université de Lille I, 1981.

[268] M. Křížek and P. Neittaanmäki. *Finite Element Approximation of Variational Problems and Applications.* Longman Scientific & Technical, Burnt Mill, 1990.

[269] M. La Porte and J. Vignes. *Algorithmes Numériques. Analyse et Mise en Œuvre. 1 – Arithmétique des Ordinateurs. Systèmes Linéaires.* Editions Technip, Paris, 1974.

[270] V. Lakshmikantham and D. Trigiante. *Theory of Difference Equations: Numerical Methods and Applications.* Academic Press, New-York, 1988.

[271] J. P. Lambert. Quasi-random sequences for optimization and numerical integration. In P. Keast and G. Fairweather, eds., *Numerical Integration*, pp. 193–203, Dordrecht, 1987. Reidel.

[272] F. M. Larkin. Some techniques for rational interpolation. *Comput. J.*, 10:178–187, 1967.

[273] F. M. Larkin. On the acceleration of certain sequences by rational interpolation. Technical report, Queen's University, Dept. of Computing, 1977.

[274] P. J. Laurent. Un théorème de convergence pour le procédé d'extrapolation de Richardson. *C. R. Acad. Sci. Paris*, 256:1435–1437, 1963.

[275] P. J. Laurent. *Etude de Procédés d'Extrapolation en Analyse Numérique.* Thèse d'Etat, Université de Grenoble, 1964.

[276] R. Lawther. Modification of iterative processes for improved convergence characteristics. *Int. J. Numer. Methods Eng.*, 15:1149–1159, 1980.

[277] H. Le Ferrand. Convergence of the topological ε-algorithm for solving systems of nonlinear equations. To appear.

[278] A. Lembarki. *Méthodes de Projection et Extensions: Etude Théorique et Pratique.* Thèse 3ème cycle, Université de Lille I, 1984.

[279] A. Lembarki. *Accélération des Fractions Continues.* Thèse d'Etat, Université de Lille I, 1987.

[280] A. Lembarki. Acceleration of limit periodic continued fractions by the T_{+m} transformation. *J. Comput. Appl. Math.*, 19:109–116, 1987.

[281] A. Lembarki. Convergence acceleration of limit k-periodic continued fractions. *Appl. Numer. Math.*, 4:337–349, 1988.

[282] P. Lepora and B. Gabutti. An algorithm for the summation of series. *Appl. Numer. Math.*, 3:523–528, 1987.

[283] D. Levin. Development of non-linear transformations for improving convergence of sequences. *Int. J. Comput. Math.*, B3:371–388, 1973.

[284] D. Levin. Numerical inversion of the Laplace transform by accelerating the convergence of Bromwich's integral. *J. Comput. Appl. Math.*, 1:247–250, 1975.

[285] D. Levin. On accelerating the convergence of infinite double series and integrals. *Math. Comput.*, 35:1331–1345, 1980.

[286] D. Levin and A. Sidi. Two new classes of nonlinear transformations for accelerating the convergence of infinite integrals and series. *Appl. Math. Comput.*, 9:175–215, 1981.

[287] P. Levrie and A. Bultheel. A note on two convergence acceleration methods for ordinary continued fractions. *J. Comput. Appl. Math.*, 24:403–409, 1988.

[288] L. Lillich. *Der ε-Algorithmus.* Diplomarbeit, University of Mannheim, 1986.

[289] Q. Lin, I. H. Sloan, and R. Xie. Extrapolation of the iterated-collocation method for integral equations of the second kind. *SIAM J. Numer. Anal.*, 27:1535–1541, 1990.

[290] A. M. Litovsky. *Accélération de la Convergence des Ensembles Synchrones.* Thèse, Université de Lille I, 1989.

[291] S. L. Loi. A general algorithm for rational interpolation. Technical report, University of Canterbury, New-Zealand, 1982.

[292] S. L. Loi. *Quadratic Approximation and its Applications to Acceleration of Convergence.* PhD thesis, University of Canterbury, New-Zealand, 1982.

[293] S. L. Loi. Convergence acceleration by means of quadratic approximation. Technical report, National University of Singapore, dept. of Math., 1984.

[294] S. L. Loi and A. W. McInnes. An algorithm for generalized rational interpolation. *BIT*, 23:105–117, 1983.

[295] S. L. Loi and A. W. McInnes. An algorithm for quadratic approximation. *J. Comput. Appl. Math.*, 11:161–174, 1984.

[296] I. M. Longman. Numerical Laplace transform inversion of a function arising in viscoelasticity. *J. Comput. Phys.*, 10:224–231, 1972.

[297] I. M. Longman. The summation of series. *Appl. Numer. Math.*, 2:135–141, 1986.

[298] I. M. Longman. A note on the summation of power series. *Appl. Numer. Math.*, 4:431–437, 1988.

[299] I. M. Longman and M. Sharir. Laplace transform inversion of rational functions. *Geophys. J. R. Astron. Soc.*, 25:299–305, 1971.

[300] L. Lorentzen and H. Waadeland. *Continued Fractions and Some of its Applications.* North-Holland, Amsterdam. To appear.

[301] S. Lubkin. A method of summing infinite series. *J. Res. Natl. Bur. Stand.*, 48:228–254, 1952.

[302] J. N. Lyness. The Euler-Maclaurin expansion for the Cauchy principal value integral. *Numer. Math.*, 46:611–622, 1985.

[303] J. N. Lyness. Integrating some infinite oscillating tails. *J. Comput. Appl. Math.*, 12–13:109–117, 1985.

[304] J. N. Lyness. Numerical integration. Part A: extrapolation methods for multi-dimensional quadrature. In J. L. Mohamed and J. E. Walsh, eds., *Numerical Algorithms*, pp. 105–124, Oxford, 1986. Clarendon Press.

[305] L. A. Lyusternik and A. R. Yanpol'skii, eds. *Mathematical Analysis.* Pergamon Press, London, 1965.

[306] G. Marchouk and V. Shaydurov. *Raffinement des Solutions des Schémas aux Différences.* Mir, Moscou, 1983.

[307] G. I. Marchuk and V. V. Shaidurov. *Difference Methods and their Extrapolation.* Springer-Verlag, Berlin, 1983.

[308] I. Marx. Remark concerning a non-linear sequence-to-sequence transform. *J. Math. and Phys.*, 42:334–335, 1963.

[309] A. C. Matos. Acceleration methods for sequences (S_n) such that $\Delta S_n = \sum_{i=1}^{\infty} a_i g_i(n)$. In C. Brezinski, ed., *Numerical and Applied Mathematics*, volume 1.2, pp. 447–451, Basel, 1989. Baltzer.

[310] A. C. Matos. *Construction de Procédés d'Extrapolation à Partir de Développements Asymptotiques.* Thèse, Université de Lille I, 1989.

[311] A. C. Matos. Acceleration methods based on convergence tests. *Numer. Math.*, 58:329–340, 1990.

[312] A. C. Matos. A convergence acceleration method based on a good estimation of the absolute value of the error. *IMA J. Numer. Anal.*, 10:243–251, 1990.

[313] A. C. Matos. Extrapolation algorithms based on the asymptotic expansion of the inverse of the error. Application to continued fractions. *J. Comput. Appl. Math.*, 32:179–190, 1990.

[314] A. C. Matos. Acceleration results for the vector E-algorithm. *Numer. Algorithms*, 1, 1991. To appear.

[315] A. C. Matos and M. Prévost. Acceleration property of the E-algorithm. To appear.

[316] J. C. Maxwell. *A Treatise on Electricity and Magnetism.* Oxford University Press, Oxford, 1892.

[317] J. B. McLeod. A note on the ε-algorithm. *Computing*, 7:17–24, 1971.

[318] G. Meinardus. Über das asymptotische Verhalten von Iterationfolgen. *Z. Angew. Math. Mech.*, 63:70–72, 1983.

[319] M. Mešina. Convergence acceleration for the iterative solution of $x = Ax + f$. *Comput. Methods Appl. Mech. Eng.*, 10:165–173, 1977.

[320] P. Midy. Scaling transformations and extrapolation algorithms for vector sequences. To appear.

[321] J. C. Miellou. Extrapolation aggregation algorithm of monotone kind. Application to "one obstacle's" stationnary problems. In *Free Boundary Problems, Pavia, sept.–oct. 1979*, volume II, pp. 411–438. Istituto Nazionale di Alta Matematica Francesco Severi, Roma, 1980.

[322] M. Minoux. *Mathematical Programming.* Wiley, Chichester, 1986.

[323] M. Monchamp and A. Chamaraux. *Méthodes ... ou Recettes (?).* Ellipses, Paris, 1986.

[324] M. Morandi Cecchi and R. Nociforo. Discrete finite elements method in space-time domain for parabolic linear problem. To appear.

[325] M. Morandi Cecchi and M. Redivo Zaglia. Mathematical programming techniques to solve biharmonic problems by a recursive projection algorithm. *J. Comput. Appl. Math.*, 32:191–201, 1990.

[326] M. Morandi Cecchi, M. Redivo Zaglia, and G. Scenna. Prediction of the numerical solution of parabolic problems. To appear.

[327] V. A. Morozov. *Methods for Solving Incorrectly Posed Problems.* Springer-Verlag, Berlin, 1984.

[328] S. L. Moshier. *Methods and Programs for Mathematical Functions.* Ellis Horwood, Chichester, 1989.

[329] G. Mühlbach. Neville-Aitken algorithms for interpolating by functions of Čebyšev-systems in the sense of Newton and in a generalized sense of Hermite. In A. G. Law and B. N. Sahney, eds., *Theory of Approximation with Applications*, pp. 200–212, New-York, 1976. Academic Press.

[330] G. Mühlbach. On interpolating by generalized polynomials of one and of several variables. *Det Kong. Norske Vid. Selsk. Skr.*, 2:87–101, 1989.

[331] G. Mühlbach and L. Reimers. Linear extrapolation by rational functions, exponentials and logarithmic functions. *J. Comput. Appl. Math,*, 17:329–344, 1987.

[332] M. A. Piñar and V. Ramirez. Recursive inversion of Hankel matrices. *Monogr. Acad. Ciencias Zaragoza*, 1:119–128, 1988.

[333] F. Neelamkavil. *Computer Simulation and Modelling.* Wiley, Chichester, 1987.

[334] W. Niethammer. Numerical application of Euler's series transformation and its generalizations. *Numer. Math.*, 34:271–283, 1980.

[335] W. Niethammer and H. Wietschorke. On the acceleration of limit periodic continued fractions. *Numer. Math.*, 44:129–137, 1984.

[336] A. F. Nikiforov and V. B. Uvarov. *Special Functions of Mathematical Physics.* Birkäuser Verlag, Basel, 1988.

[337] N. E. Nörlund. Fractions continues et différences réciproques. *Acta Math.*, 34:1–108, 1911.

[338] A. W. M. Nourein. Root determination by use of Padé approximants. *BIT*, 16:291–297, 1976.

[339] E. Novak. *Deterministic and Stochastic Error Bounds in Numerical Analysis*, volume 1349 of *LNM*. Springer-Verlag, Berlin, 1988.

[340] C. Nowakowski. *Essai d'Etablissement d'un Procédé Rationnel et Pratique des Calculs de Climatisation d'Eté en Régime Variable.* Thèse d'ingénieur CNAM, Paris, 1974.

[341] T. H. O'Beirne. On linear iterative processes and on methods of improving the convergence of certain types of iterated sequences. Technical report, Torpedo Experimental Establishment, Greenock, May 1947.

[342] G. Opfer. On monotonicity preserving linear extrapolation sequences. *Computing*, 5:259–26, 1970.

[343] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New-York, 1970.

[344] H. Ortloff. Asymptotisches Verhalten und Konvergenzbeschleunigung von Iterationsfolgen. *Numer. Math.*, 49:545–559, 1986.

[345] H. Ortloff and D. Böhm. Algorithmen zur Konvergenzbeschleunigung von Iterationsfolgen und Reihen und ihre Implementierung. *Wiss. Z. Techn. Hochsch. Magdeburg*, 29:120–126, 1985.

[346] N. Osada. Accelerable subsets of logarithmic sequences. *J. Comput. Appl. Math.*, 32:217–228, 1990.

[347] N. Osada. A convergence acceleration method for some logarithmically convergent sequences. *SIAM J. Numer. Anal.*, 27:178–189, 1990.

[348] N. Osada. Acceleration methods for vector sequences. *J. Comput. Appl. Math.* To appear.

[349] N. Osada. A method for obtaining sequence transformations. *IMA J. Numer. Anal.* To appear.

[350] A. M. Ostrowski. *Solution of Equations and Systems of Equations.* Academic Press, New-York, 1966.

[351] K. J. Overholt. Extended Aitken acceleration. *BIT*, 5:122–132, 1965.

[352] K. J. Overholt. The P-algorithms for extrapolation. *Det Kong. Norske Vid. Selsk. Skr.*, 2:121–129, 1989.

[353] S. Paszkowski. Sur un problème d'approximation posé par C. Brezinski. Publication ANO 210, Université de Lille I, 1989.

[354] J. Patry. *Les Fractions Continues: Théorie et Applications.* Editions Technip, Paris, 1991.

[355] R. Pennacchi. Le trasformazioni razionali di una successione. *Calcolo*, 5:37–50, 1968.

[356] D. Petit. *Etude de Certains Procédés d'Accélération de la Convergence en Analyse Numérique.* Thèse 3ème cycle, Université de Lille I, 1977.

[357] G. M. Phillips. Aitken sequences and Fibonacci numbers. *Am. Math. Mon.*, 91:354–357, 1984.

[358] M. Pichat. Correction d'une somme en arithmétique à virgule flottante. *Numer. Math.*, 19:400–406, 1972.

[359] M. Pichat and J. Vignes. *Ingéniérie du Contrôle de la Précision des Calculs sur Ordinateur.* Editions Technip, Paris. To appear.

[360] R. Piessens, E. de Donker-Kapenga, C. W. Überhuber, and D. K. Kahaner. *Quadpack. A Subroutine Package for Automatic Integration.* Springer-Verlag, Berlin, 1983.

[361] R. Powell and S. M. Shah. *Summability Theory and its Applications.* Van Nostrand Reinhold, London, 1972.

[362] M. Prévost. *Sommation de Certaines Séries Formelles par Approximation de la Fonction Génératrice.* Thèse 3ème cycle, Université de Lille I, 1983.

[363] M. Prévost. Calculation of poles of meromorphic functions with q-d, r-s and ε-algorithms. Acceleration of these processes. *J. Comput. Appl. Math.*, 19:89–98, 1987.

[364] M. Prévost. Determinantal expressions for partial Padé approximants. *Appl. Numer. Math.*, 6:221–224, 1989/90.

[365] M. Prévost. Acceleration of some logarithmic sequences of moments. To appear.

[366] M. Prévost. Aitken's Δ^2 and Lubkin's W processes as Padé-type approximants. Generalization. To appear.

[367] M. Prévost. Dirac masses determination with orthogonal polynomials and ε-algorithm. Application to totally monotonic sequences. To appear.

[368] B. P. Pugachev. Acceleration of the convergence of iterative processes and a method of solving systems of nonlinear equations. *U.S.S.R. Comput. Math. Math. Phys.*, 17:199–207, 1978.

[369] W. C. Pye and T. A. Atchison. An algorithm for the computation of the higher order G-transformation. *SIAM J. Numer. Anal.*, 10:1–7, 1973.

[370] L. D. Pyle. A generalized inverse ε-algorithm for constructing intersection projection matrices, with applications. *Numer. Math.*, 10:86–102, 1967.

[371] L. Qun and L. Tao. The combination of approximate solutions for accelerating the convergence. *RAIRO, Anal. Numér.*, 18:153–160, 1984.

[372] N. Rahmani-Gasmi. *Stabilité Numérique de l'Algorithme de GRG. Accélération de la Convergence et Optimisation du Temps de Calcul.* Thèse 3ème cycle, Université de Paris VI, 1985.

[373] M. Redivo Zaglia. *Extrapolation, Méthodes de Lanczos et Polynômes Orthogonaux: Théorie et Conception de Logiciels.* Thèse, Université de Lille I. To appear.

[374] G. Ribière. Régularisation d'opérateurs. *RIRO*, 5:57–79, 1967.

[375] W. Romberg. Vereinfachte numerische Integration. *Kgl. Norske Vid. Selsk. Forsk.*, 28:30–36, 1955.

[376] J. B. Rosen. The gradient projection method for nonlinear programming. Part I. Linear constraints. *J. SIAM*, 8:181–217, 1960.

[377] C. S. Rudisill and Y. Y. Chu. Numerical methods for evaluating the derivatives of eigenvalues and eigenvectors. *AAIA J.*, 13:834–837, 1975.

[378] H. Rutishauser. *Der Quotienten-Differenzen Algorithmus.* Birkhäuser-Verlag, Basel, 1957.

[379] P. Sablonnière. Convergence acceleration of logarithmic fixed point sequences. *J. Comput. Appl. Math.*, 19:55–60, 1987.

[380] P. Sablonnière. Comparison of four algorithms accelerating the convergence of some logarithmic fixed point sequences. *Numer. Algorithms*, 1, 1991. To appear.

[381] H. Sadok. *Accélération de la Convergence de Suites par Utilisation des Transformations Composites.* Thèse 3ème cycle, Université de Lille I, 1986.

[382] H. Sadok. *Accélération de la Convergence de Suites Vectorielles et Méthodes de Point Fixe.* Thèse, Université de Lille I, 1988.

[383] H. Sadok. About Henrici's transformation for accelerating vector sequences. *J. Comput. Appl. Math.*, 29:101–110, 1990.

[384] H. Sadok. Quasi-linear vector extrapolation methods. To appear.

[385] E. B. Saff. Polynomials orthogonal on semi–circles: Gautschi meets Padé. Third International Congress on Orthogonal Polynomials and their Applications, Erice, June 1–6, 1990, unpublished manuscript, 1990.

[386] K. Sato. On acceleration procedures of some slowly convergent infinite summations. *J. Inf. Process.*, 4:152–154, 1981.

[387] J. R. Schmidt. On the numerical solution of linear simultaneous equations by an iterative method. *Phil. Mag.*, 7:369–383, 1941.

[388] C. Schneider. Vereinfachte Rekursionen zur Richardson-Extrapolation in Spezialfällen. *Numer. Math.*, 24:177–184, 1975.

[389] G. A. Sedogbo. Convergence acceleration of some logarithmic sequences. *J. Comput. Appl. Math.*, 32:253–260, 1990.

[390] R. E. Shafer. On quadratic approximation. *SIAM J. Numer. Anal.*, 11:447–460, 1974.

[391] D. Shanks. An analogy between transient and mathematical sequences and some nonlinear sequence-to-sequence transforms suggested by it. Part I. Memorandum 9994, Naval Ordnance Laboratory, White Oak, July 1949.

[392] D. Shanks. Non linear transformations of divergent and slowly convergent sequences. *J. Math. Phys.*, 34:1–42, 1955.

[393] J. A. Shanks. Romberg tables for singular integrands. *Comput. J.*, 15:360–361, 1972.

[394] J. A. Shanks. On forming the Romberg table. *J. Comput. Appl. Math.*, 11:343–351, 1984.

[395] A. Sidi. Extrapolation methods for oscillary infinite integrals. *J. Inst. Maths Applics*, 26:1–20, 1980.

[396] A. Sidi. Numerical quadrature and nonlinear sequence transformations: unified rules for efficient computation of integrals with algebraic and logarithmic endpoint singularities. *Math. Comput.*, 35:851–874, 1980.

[397] A. Sidi. An algorithm for a special case of a generalization of the Richardson extrapolation process. *Numer. Math.*, 38:299–307, 1982.

[398] A. Sidi. The numerical evaluation of very oscillatory infinite integrals by extrapolation. *Math. Comput.*, 38:517–529, 1982.

[399] A. Sidi. Convergence and stability properties of minimal polynomial and reduced rank extrapolation algorithms. *SIAM J. Numer. Anal.*, 23:197–209, 1986.

[400] A. Sidi. Extrapolation methods for divergent oscillatory infinite integrals that are defined in the sense of summability. *J. Comput. Appl. Math.*, 17:105–114, 1987.

[401] A. Sidi. Extrapolation vs. projection methods for linear systems of equations. *J. Comput. Appl. Math.*, 22:71–88, 1988.

[402] A. Sidi. Generalisations of Richardson extrapolation with application to numerical integration. In H. Brass and G. Hämmerlin, eds., *Numerical Integration III*, volume 85 of *ISNM*, pp. 237–250, Basel, 1988. Birkhäuser-Verlag.

[403] A. Sidi. A user-friendly extrapolation method for oscillatory infinite integrals. *Math. Comput.*, 51:249–266, 1988.

[404] A. Sidi. On a generalization of the Richardson extrapolation process. *Numer. Math.*, 57:365–377, 1990.

[405] A. Sidi. On rates of acceleration of extrapolation methods for oscillatory infinite integrals. *BIT*, 30:347–357, 1990.

[406] A. Sidi. Efficient implementation of minimal polynomial and reduced rank extrapolation methods. *J. Comput. Appl. Math.*. To appear.

[407] A. Sidi and J. Bridger. Convergence and stability analysis for some vector extrapolation methods in the presence of defective iteration matrices. *J. Comput. Appl. Math.*, 22:35–61, 1988.

[408] A. Sidi, W. F. Ford, and D. A. Smith. Acceleration of convergence of vector sequences. *SIAM J. Numer. Anal.*, 23:178–196, 1986.

[409] A. Sidi and D. Levin. Prediction properties of the t-transformation. *SIAM J. Numer. Anal.*, 20:589–598, 1983.

[410] S. Skelboe. Computation of the periodic steady-state response to non linear networks by extrapolation methods. *IEEE Trans. Circuits Syst.*, 27:161–175, 1980.

[411] F. Sloboda. Nonlinear iterative methods and parallel computation. *Apl. Mat.*, 21:252–262, 1976.

[412] A. C. Smith. A one-parameter method for accelerating the convergence of sequences and series. *Comput. Math. Appl.*, 4:157–171, 1978.

[413] D. A. Smith and W. F. Ford. Acceleration of linear and logarithmic convergence. *SIAM J. Numer. Anal.*, 16:223–240, 1979.

[414] D. A. Smith and W. F. Ford. Numerical comparison of nonlinear convergence accelerators. *Math. Comput.*, 38:481–499, 1982.

[415] D. A. Smith, W. F. Ford, and A. Sidi. Extrapolation methods for vector sequences. *SIAM Rev.*, 29:199–233, 1987.

[416] D. A. Smith, W. F. Ford, and A. Sidi. Correction to "Extrapolation methods for vector sequences". *SIAM Rev.*, 30:623–624, 1988.

[417] J. Spanier and K. B. Oldham. *An Atlas of Functions.* Hemisphere Publ. Corp., Washington, 1987.

[418] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, Berlin, 1980.

[419] R. F. Streit. The evaluation of double series. *BIT*, 12:400–408, 1972.

[420] R. F. Streit. The T_{+m} transformation. *Math. Comput.*, 30:505–511, 1976.

[421] M. Sugihara. Methods of numerical integration of oscillatory functions by the DE-formula with Richardson extrapolation. *J. Comput. Appl. Math.*, 17:47–68, 1987.

[422] B. A. Szabo. Estimation and control of error based on p convergence. In I. Babuška et al., eds., *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, pp. 61–78, Chichester, 1986. Wiley.

[423] R. C. E. Tan. Accelerating the convergence of an iterative method for derivatives of eigensystems. *J. Comput. Phys.*, 67:230–235, 1986.

[424] R. C. E. Tan. Computing derivatives of eigensystems by the topological ε-algorithm. *Appl. Numer. Math.*, 3:539–550, 1987.

[425] R. C. E. Tan. Computing derivatives of eigensystems by the vector ε-algorithm. *IMA J. Numer. Anal.*, 7:485–494, 1987.

[426] R. C. E. Tan. An extrapolation method for computing derivatives of eigensystems. *Int. J. Comput. Math.*, 22:63–73, 1987.

[427] R. C. E. Tan. *Extrapolation Methods and Iterative Computation of Derivatives of Eigensystems.* PhD thesis, La Trobe University, 1988.

[428] R. C. E. Tan. Implementation of the topological ε-algorithm. *SIAM J. Sci. Stat. Comput.*, 9:839–848, 1988.

[429] R. C. E. Tan. Some acceleration methods for iterative computation of derivatives of eigenvalues and eigenvectors. *Int. J. Numer. Methods Eng.*, 28:1505–1519, 1989.

[430] R. C. E. Tan and A. L. Andrew. Computing derivatives of eigenvalues and eigenvectors by simultaneous iteration. *IMA J. Numer. Anal.*, 9:111–122, 1989.

[431] U. Tempelmeier. A new proof of the cross-rule for the ε-algorithm based on Schur complements. *J. Comput. Appl. Math.*, 21:55–61, 1988.

[432] T. N. Thiele. *Interpolationsrechnung.* Teubner, Leipzig, 1909.

[433] W. J. Thron and H. Waadeland. On certain transformation of continued fractions. In W. B. Jones et al., eds., *Analytic Theory of Continued Fractions*, volume 932 of *LNM*, pp. 225–240, Berlin, 1982. Springer-Verlag.

[434] A. N. Tikhonov. On the solution of ill-posed problems and the method of regularization. *Sov. Math. Dokl.*, 4:1035–1038, 1963.

[435] H. Toda and H. Ono. Some remarks for efficient usage of the double exponential formulas. *RIMS Kokyuroku*, 339:74–109, 1978. (in japanese).

[436] J. F. Traub and H. Woźniakowski. *A General Theory of Optimal Algorithms.* Academic Press, New-York, 1980.

[437] W. F. Trench. An algorithm for the inversion of finite Hankel matrices. *J. SIAM*, 13:1102–1107, 1965.

[438] G. M. Trojan. Lower bounds and fast algorithms for sequence acceleration. *J. Assoc. Comput. Mach.*, 31:329–335, 1984.

[439] G. M. Trojan. Optimal transformations for scalar sequences. *Numer. Math.*, 48:21–32, 1986.

[440] R. R. Tucker. The δ^2-process and related topics. *Pac. J. Math.*, 22:349–359, 1967.

[441] M. D. van Dyke and A. J. Guttmann. Subsonic potential flow past a circle and the transonic controversy. *J. Aust. Math. Soc., Ser. B*, 24:243–261, 1983.

[442] J. van Iseghem. Vector Padé approximants. In R. Vichnevetsky and J. Vignes, eds., *Numerical Mathematics and Applications*, pp. 73–77, Amsterdam, 1985. North-Holland.

[443] J. van Iseghem. *Approximants de Padé Vectoriels.* Thèse d'Etat, Université de Lille I, 1987.

[444] J. van Iseghem. Laplace transform inversion and Padé-type approximants. *Appl. Numer. Math.*, 3:529–538, 1987.

[445] J. van Iseghem. Vector orthogonal relations. Vector QD algorithm. *J. Comput. Appl. Math.*, 19:141–150, 1987.

[446] J. van Iseghem. Convergence of the vector QD-algorithm. Zeros of vector orthogonal polynomials. *J. Comput. Appl. Math.*, 25:33–46, 1989.

[447] J. M. Vanden Broeck and L. W. Schwartz. A one-parameter family of sequence transformations. *SIAM J. Math. Anal.*, 10:658–666, 1979.

[448] R. S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, 1962.

[449] B. Verdonk. *Different Techniques for Multivariate Rational Interpolation and Padé Approximation.* Thesis, University of Antwerp, Antwerp, 1988.

[450] Yu. V. Vorobyev. *Method of Moments in Applied Mathematics.* Gordon and Breach, New-York, 1965.

[451] G. Walz. A contour integral representation of linear extrapolation methods. *Numer. Math.*, 55:477–480, 1989.

[452] G. Walz. Error bounds and stopping rules for extrapolation methods. *IMA J. Numer. Anal.*, 9:185–198, 1989.

[453] G. Walz. Increasing the convergence modulus of an asymptotic expansion. *Appl. Numer. Math.*, 6:415–423, 1989/1990.

[454] A. Wambecq. Nonlinear methods in solving ordinary differential equations. *J. Comput. Appl. Math.*, 1:27–33, 1976.

[455] A. Wambecq. Rational Runge-Kutta methods for solving systems of ordinary differential equations. *Computing*, 20:333–342, 1978.

[456] S. Wang. *F-accélération et Transformations Diagonales.* Thèse, Université de Lille I, 1991.

[457] E. J. Weniger. Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series. *Comput. Physics Reports*, 10:189–371, 1989.

[458] E. J. Weniger. On the derivation of iterated sequence transformations for the acceleration of convergence and the summation of divergent series. *Comput. Phys. Commun.*, 64:19–45, 1991.

[459] E. J. Weniger, J. Grotendorst, and E. O. Steinborn. Some applications of nonlinear convergence accelerators. *Int. J. Quantum Chem.: Quantum Chem. Symp.*, 19:181–191, 1986.

[460] E. J. Weniger and E. O. Steinborn. Nonlinear sequence transformations for the efficient evaluation of auxiliary functions for GTO molecular integrals. In M. Defranceschi and J. Delhalle, eds., *Numerical Determination of the Electronic Structure of Atoms, Diatomic and Polyatomic Molecules*, pp. 341–346, Dordrecht, 1989. Kluwer Academic.

[461] H. Werner and L. Wuytack. Non-linear quadrature rules in the presence of a singularity. *Comput. Math. Appl.*, 4:237–245, 1978.

[462] D. V. Widder. *The Laplace Transform.* Princeton University Press, Princeton, 1946.

[463] J. Wimp. Some transformations of monotone sequences. *Math. Comput.*, 26:251–254, 1972.

[464] J. Wimp. Toeplitz arrays, linear sequence transformations and orthogonal polynomials. *Numer. Math.*, 23:1–17, 1974.

[465] J. Wimp. *Acceleration methods*, volume 1, pp. 181–210. In *Encyclopedia of Computer Science and Technology*, New-York, 1975, Dekker.

[466] J. Wimp. Accelerating convergence by root finding. *Nonlinear Anal., Theory, Meth. and Appl.*, 5:1341–1347, 1981.

[467] J. Wimp. *Sequence Transformations and their Applications.* Academic Press, New-York, 1981.

[468] L. Wuytack. A new technique for rational extrapolation to the limit. *Numer. Math.*, 17:215–221, 1971.

[469] L. Wuytack. Numerical integration by using nonlinear techniques. *J. Comput. Appl. Math.*, 1:267–272, 1975.

[470] P. Wynn. On a device for computing the $e_m(S_n)$ transformation. *MTAC*, 10:91–96, 1956.

[471] P. Wynn. On a procrustean technique for the numerical transformation of slowly convergent sequences and series. *Proc. Cambridge Phil. Soc.*, 52:663–671, 1956.

[472] P. Wynn. Confluent forms of certain nonlinear algorithms. *Arch. Math.*, 11:223–234, 1960.

[473] P. Wynn. Acceleration techniques for iterated vector and matrix problems. *Math. Comput.*, 16:301–322, 1962.

[474] P. Wynn. Acceleration techniques in numerical analysis, with particular reference to problems in one independent variable. In *Proc. IFIP Congress*, pp. 149–156, Amsterdam, 1962. North-Holland.

[475] P. Wynn. Singular rules for certain nonlinear algorithms. *BIT*, 3:175–195, 1963.

[476] P. Wynn. An arsenal of Algol procedures for the evaluation of continued fractions and for effecting the epsilon algorithm. *Chiffres*, 4:327–362, 1966.

[477] P. Wynn. On the convergence and stability of the epsilon algorithm. *SIAM J. Numer. Anal.*, 3:91–122, 1966.

[478] P. Wynn. Upon a conjecture concerning a method for solving linear equations, and certain other matters. MRC Technical Summary Report 626, MRC, Madison, 1966.

[479] P. Wynn. Upon systems of recursions which obtain among the quotients of the Padé table. *Numer. Math.*, 8:264–269, 1966.

[480] P. Wynn. Transformations to accelerate the convergence of Fourier series. In *Gertrude Blanch Anniversary Volume*, pp. 339–379. Wright Patterson Air Force Base, 1967.

[481] P. Wynn. A note on the generalized Euler transformation. *Computer J.*, 14:437–441, 1971.

[482] P. Wynn. Sur les suites totalement monotones. *C. R. Acad. Sci. Paris*, 275 A:1065–1068, 1972.

[483] P. Wynn. Transformation de séries à l'aide de l'ε-algorithme. *C. R. Acad. Sci. Paris*, 275 A:1351–1353, 1972.

[484] P. Wynn. Upon some continuous prediction algorithms. I. *Calcolo*, 9:197–234, 1972.

[485] P. Wynn. A numerical method for estimating parameters in mathematical models. Rep. CRM 443, Université de Montréal, 1974.

[486] N. Yamamoto. Regularization of solutions of nonlinear equations with singular Jacobian matrices. *J. Inform. Process.*, 7:16–21, 1984.

[487] Yue-Kuen Kwok and D. Barthez. An algorithm for the numerical inversion of Laplace transforms. *Inverse Problems*, 5:1089–1095, 1989.

[488] O. C. Zienkiewicz and R. Löhner. Accelerated "relaxation" or direct solution? Future prospects for fem. *Int. J. Numer. Methods Eng.*, 21:1–11, 1985.

This page intentionally left blank

INDEX