SIAM REVIEW
Vol. 47, No. 2, pp. 317–346

Adaptive Smoothed Aggregation (αSA) Multigrid*

M. Brezina†
R. Falgout‡
S. MacLachlan†
T. Manteuffel†
S. McCormick†
J. Ruge†

Abstract. Substantial effort has been focused over the last two decades on developing multilevel iterative methods capable of solving the large linear systems encountered in engineering practice. These systems often arise from discretizing partial differential equations over unstructured meshes, and the particular parameters or geometry of the physical problem being discretized may be unavailable to the solver. Algebraic multigrid (AMG) and multilevel domain decomposition methods of algebraic type have been of particular interest in this context because of their promises of optimal performance without the need for explicit knowledge of the problem geometry. These methods construct a hierarchy of coarse problems based on the linear system itself and on certain assumptions about the smooth components of the error. For smoothed aggregation (SA) multigrid methods applied to discretizations of elliptic problems, these assumptions typically consist of knowledge of the near-kernel or near-nullspace of the weak form. This paper introduces an extension of the SA method in which good convergence properties are achieved in situations where explicit knowledge of the near-kernel components is unavailable. This extension is accomplished in an adaptive process that uses the method itself to determine near-kernel components and adjusts the coarsening processes accordingly.

Key words. algebraic multigrid (AMG), generalized smoothed aggregation (SA), adaptive method, black-box method

AMS subject classifications. 65F10, 65N55, 65F30

DOI. 10.1137/050626272

1. Introduction.

Increasing demands for accuracy in computational simulation have led to significant innovation in both computer hardware and algorithms for scientific computation. Multigrid methods have demonstrated their efficiency in solving the linear systems generated by discretizing elliptic partial differential equations (PDEs), including Laplace's and those of linear elasticity. Algebraic multigrid (AMG) methods

*Published electronically April 29, 2005. This paper originally appeared in *SIAM Journal on Scientific Computing*, Volume 25, Number 6, 2004, pages 1896–1920. This work was sponsored by the Department of Energy under grants DE-FC02-01ER25479 and DE-FG03-99ER25217, Lawrence Livermore National Laboratory under contract B533502, Sandia National Laboratories under contract 15268, and the National Science Foundation under grant DMS-0410318.

http://www.siam.org/journals/sirev/47-2/62627.html

†Department of Applied Mathematics, Campus Box 526, University of Colorado at Boulder, Boulder, CO 80309-0526 (mbrezina@math.cudenver.edu, scott.maclachlan@colorado.edu, tmanteuf@boulder.colorado.edu, stevem@boulder.colorado.edu, jruge@boulder.colorado.edu).

‡Center for Applied Scientific Computation, Lawrence Livermore National Lab, P.O. Box 808, Livermore, CA 94551 (rfalgout@llnl.gov).

offer this efficiency without reliance on knowledge of the grid geometry or of variation in the coefficients of the differential operator. For these reasons, AMG algorithms are often the method of choice for solving linear systems that arise from discretizations over unstructured grids or with significant variation in the operator.

Over the last decade, smoothed aggregation (SA; cf. [21, 23, 22, 20, 9]) has emerged as an efficient AMG solver for the algebraic systems obtained by discretizing certain classes of differential equations on unstructured meshes. In particular, SA is often very efficient at solving the systems that arise from problems of three-dimensional (3D) thin-body elasticity, a task that can tax traditional AMG techniques.

As with classical AMG [4, 18, 19], the standard SA method bases its transfer operators on certain assumptions about the nature of *algebraically smooth error*, that is, error that simple relaxation schemes, such as Jacobi or Gauss–Seidel, cannot effectively eliminate. For SA applied to discretizations of elliptic PDEs, this assumption usually takes the form of explicit knowledge of the near-kernel of the associated weak form. This knowledge is easy to obtain for large classes of problems. For example, it is simple to determine representative near-kernel components for finite element discretizations of second- or fourth-order PDEs, including many nonscalar problems. In more general situations, however, this knowledge may not be readily available.

Consider the case where a matrix is provided without knowledge of how the original problem was discretized or scaled. Seemingly innocuous discretization practices, such as the use of scaled bases, can hamper algebraic multilevel solvers if these practices are not taken into account. Even the simplest problems, discretized on regular grids using standard finite elements, can pose serious difficulties if the resulting matrix has been scaled, without providing this information to the solver. Other discretization practices leading to problematic linear equations include the use of exotic bases and systems problems in which different local coordinates are used for different parts of the model.

To successfully solve such problems when only the matrix is provided, we need a process by which the algebraic multilevel solver can determine how to effectively coarsen a linear system using only information from the system itself. The method we propose here, which we call *adaptive* smoothed aggregation (αSA), is an attempt to do just that. This modification of standard SA is based on the simple principle that applying a linear iterative method to the homogeneous problem ($Ax = 0$) quickly reveals error components that the method does not effectively reduce. While this principle is easily stated in loose terms, the resulting algorithm and its implementation can be very subtle. We hope to expose these subtleties in the presentation that follows.

This paper develops αSA in such a way that good convergence properties are recovered even if explicit knowledge of the near-kernel is incomplete or lacking altogether. This should facilitate solution in cases where the problem geometry, discretization method, or coefficients of the differential operator are not explicitly known to the solver. At the same time, we strive to keep iteration costs and storage requirements low.

The setup phase for αSA can be succinctly described as an adaptive iterative process performed on the method itself, starting from a given primitive method (possibly even a simple relaxation scheme), with error propagation operator M_0. We attempt to recover error components that are not effectively reduced by M_0. We proceed by first putting M_0 to the test: given a small number, n, of iterations and a random initial guess, $\mathbf{e}_0$, compute

$$\mathbf{e}_n \leftarrow M_0^n \mathbf{e}_0. \tag{1.1}$$

We evaluate convergence of (1.1) and, if the method performs well in the sense that $\mathbf{e}_n$ is much smaller than $\mathbf{e}_0$ in an appropriate norm, then the method is accepted and the adaptive scheme stops. Otherwise, the resulting approximation, $\mathbf{e}_n$, must, by definition, be an error that cannot be effectively reduced by M_0. The idea now is to define a coarsening process that does effectively reduce this "algebraically smooth" error (and other errors that are locally similar), while not corrupting whatever error elimination properties the method may already have. Smoothed aggregation provides a convenient and efficient framework for developing such an improved coarsening process based on $\mathbf{e}_n$, yielding an improved method with error propagation operator M_1. The whole process can then be repeated with M_1 in place of M_0, continuing in this way to generate a sequence of improving methods, M_k.

The concept of using a multigrid algorithm to improve itself is not new. Using representative smooth vectors in the coarsening process was first developed in the early stages of the AMG project of Brandt, McCormick, and Ruge (documented in [15]), where *interpolation* (or *prolongation*) was defined to fit vectors obtained by relaxation on the homogeneous problem. In [4], a variation of this idea was used for recovering typical AMG convergence rates for a badly scaled scalar elliptic problem. While the method there was very basic and used only one candidate, it contained many of the ingredients of the approach developed below. These concepts were developed further in [14, 16, 17, 19]. The idea of fitting eigenvectors corresponding to the smallest eigenvalues was advocated in [14] and [19], where an AMG algorithm determining these eigenvectors through Rayleigh quotient minimization was outlined. These vectors were, in turn, used to update the AMG interpolation and coarse-level operators. A more sophisticated adaptive framework appropriate for the standard AMG is currently under investigation [7].

Another method of the type developed here is the bootstrap AMG scheme proposed recently by Brandt [3] and Brandt and Ron [5]. It differs somewhat from the methods described here in that it starts on the fine level by iterating on a number of different random initial guesses, with interpolation then constructed to approximately fit the set of resulting vectors in a least-squares sense.

Various other attempts have been made to allow for a multilevel solver itself to determine from the discrete problem the information required to successfully solve it, without a priori assumptions on the form of the smooth error. These include the methods of [13, 8, 6, 11, 12]. All of these approaches, however, require access to the local finite element matrices of the problem in order to construct the multigrid transfer operators based on the eigenvectors associated with the smallest eigenvalues of the aggregated stiffness matrices. Although these methods exhibit attractive convergence properties, their need to construct, store, and manipulate coarse-level element information typically leads to increased storage requirements compared to those of classical AMG or standard SA. The method we develop here aims to achieve similar good convergence properties to the element-based methods without the need for element information or the attendant extra storage it demands.

This paper is organized as follows. In section 2, we briefly recall standard multilevel methods, including smoothed aggregation, and introduce notation used throughout the remainder of the paper. Readers who are unfamiliar with the fundamental concepts assumed here may wish to further consult basic references on multigrid and AMG (e.g., [10]) and on SA (e.g., [20]). Section 3 describes the basic principles and ideas behind the adaptive methodology, and section 4 develops the particular αSA scheme. Finally, section 5 presents computational examples demonstrating the performance of the SA method based on these adaptive setup concepts.

2. Preliminaries. For those unfamiliar with standard multilevel methods, we describe here the principles of the multilevel methodology and give details of the generalized SA multigrid method [20]. Concepts introduced here are those necessary for the development of the adaptive methodology and its particular applications to the SA multigrid algorithm. This section also introduces most of the notation and conventions used in the following sections.

For many discrete linear problems arising from elliptic PDEs and elsewhere, the matrix, A, is large, sparse, symmetric, and positive definite. (We assume these properties throughout this paper, although much of what we develop holds for more general cases.) Thus, the solution of

$$A\mathbf{x} = \mathbf{b} \tag{2.1}$$

can be treated by simple iterative (so-called *relaxation*) methods that involve corrections based on the residual, $\mathbf{r} = A\mathbf{x} - \mathbf{b}$. These methods take the form

$$\mathbf{x} \leftarrow \mathbf{x} - R\mathbf{r}, \tag{2.2}$$

where R is an appropriately chosen approximate inverse of A, usually based on a simple matrix splitting. Common examples include pointwise Richardson ($R = sI$), Jacobi ($R = D^{-1}$, where D represents the diagonal of A), or Gauss–Seidel ($R = L^{-1}$, where L represents the lower-triangular part of A). These methods converge to the exact solution of (2.1) for any initial approximation, $\mathbf{x}$ (cf. [24]). However, for these simple choices of R, the error propagation operator, $I - RA$, typically has spectral radius near 1, becoming closer to unity as the problem size increases and rendering relaxation methods alone unacceptably slow for solving (2.1).

Multigrid attempts to rescue relaxation by projecting out the error components that induce this slowness. To make such a correction process effective, the multigrid designer must be able to characterize these slow-to-converge components. Notice that, for a given approximation, $\mathbf{x}$, to the solution, $\mathbf{x}^*$, of (2.1), iteration (2.2) is slow to converge when the error, $\mathbf{e} = \mathbf{x} - \mathbf{x}^*$, yields a relatively small residual, $\mathbf{r} = A\mathbf{e} \approx \mathbf{0}$. We call the error components for which this iteration is slow to converge *algebraically smooth* to emphasize that such components need not, in general, possess the *geometric* smoothness often assumed in designing classical geometric multigrid methods. In particular, for problems resulting from PDE discretization over a given mesh, the algebraically smooth components need not vary slowly between neighboring grid points. Under certain assumptions on A and R, the vectors, $\mathbf{v}$, that yield small Rayleigh quotients, $\frac{\langle A\mathbf{v},\mathbf{v}\rangle}{\|A\|\langle \mathbf{v},\mathbf{v}\rangle} \ll 1$, are easily seen to be algebraically smooth (e.g., when $R = sI$, as in Richardson's method). For this reason, algebraically smooth vectors are often also referred to as the *near-kernel* of A. We use these two terms interchangeably; however, we emphasize that the algebraically smooth vectors that we are interested in may not, for all relaxation schemes, also be near-kernel components.

Once identified, algebraically smooth error can be reduced by a correction process that complements the chosen relaxation scheme. In multigrid, this correction to the algebraically smooth error is carried out by finding an approximation to the error using a problem that has fewer degrees of freedom and, thus, is less expensive to solve. Choosing the coarse space is a complex matter, some details of which are discussed later in this section. In this paper, we focus attention on constructing an appropriate mapping between coarse and fine spaces, and so the discussion here focuses on this map. Let P (without subscript or superscript) denote a generic full-rank operator such that the range of P approximately contains the algebraically smooth components

associated with iteration (2.2), and the column dimension of P is significantly smaller than the dimension of A. Such an operator, referred to as a *prolongator*, provides a means of communicating information between coarse and fine levels and can also be used to define the coarse-level version of (2.1). In particular, it can be used to effectively eliminate error in the range of P from the fine-level approximation by performing a correction of the form

$$\mathbf{x}_{new} \leftarrow \mathbf{x} - P\mathbf{y}. \tag{2.3}$$

An effective way to determine this approximation is to define it *variationally*, as the correction that minimizes the energy of error associated with the updated approximation, $\mathbf{x}_{new}$:

$$\|\mathbf{e}_{new}\|_A = \|\mathbf{e} - P\mathbf{y}\|_A \rightarrow \min, \tag{2.4}$$

where $\mathbf{e}_{new} = \mathbf{x}_{new} - \mathbf{x}^*$ and $\|\cdot\|_A = \langle A\cdot, \cdot\rangle^{\frac{1}{2}}$ denotes the energy (or A-) norm. This approximation, $\mathbf{y}$, is computable: it is the solution to the coarse-level matrix equation,

$$A_c\mathbf{y} = P^T(A\mathbf{x} - \mathbf{b}), \tag{2.5}$$

where $A_c = P^TAP$ is then the variationally defined coarse-level matrix. The fine-level update given in (2.3) is usually referred to as a *coarse-grid correction*; we adopt this terminology here even though the coarse-level problems to be considered are not necessarily associated with any geometric grids. A computationally efficient method demands that the coarse-grid correction be computable at a small multiple of the cost of the relaxation on the fine level. This is possible only when the dimension of coarse-level matrix A_c (or the column dimension of P) is significantly smaller than that of fine-level matrix A, and the nonzero structure of P retains sparsity in A_c.

Thus, a variational two-level *cycle* can be defined by first relaxing on the fine level (typically once or twice), then transferring the residual to the coarse level, defining the right side of (2.5), solving for coarse-grid correction $\mathbf{y}$, and then interpolating and correcting the fine-level approximation according to (2.3). This becomes a multigrid scheme by applying this procedure recursively to solve (2.5), using relaxation followed by correction from yet coarser levels. To obtain a method that effectively reduces all components of the error, it is crucial that the algebraically smooth error components, for which relaxation is inefficient, be well approximated by vectors in the range of P. This is commonly referred to as the *complementarity principle* and underlies all multigrid methods. The process of constructing the prolongation operator is referred to as the *coarsening process* and typically involves first determining the nonzero structure of P and then endowing it with appropriate values to guarantee good approximation of the algebraically smooth error.

For many important problems and discretization techniques, the notion of algebraic and geometric smoothness coincide, and constructing P based on linear interpolation may result in optimal approximation of these geometrically smooth error components. However, for even the simplest second-order scalar PDE problems discretized using unstructured meshes over complicated geometries, constructing P based on linear interpolation may be problematic. To accomplish this task, algebraic multigrid (AMG) [4] was developed. Its basic premise is to determine coarse levels and prolongation operators automatically by inspecting the entries of A.

Although classical AMG has been successfully used for many classes of problems, its coarsening implicitly relies on geometric smoothness of the error, which means that

it is not directly suitable for treating problems where algebraically smooth error is not also geometrically smooth. A generalization of the classical AMG method, suitable for such problems, was recently introduced in [7].

Generalized smoothed aggregation (SA [23, 20]), a variant of AMG, allows for any specified components to be easily incorporated into the construction of the prolongator. However, this requires that these components be known a priori and supplied to the method. For many classes of problems, these vectors are available; here, we focus on the cases when this is not so. This paper introduces αSA, an extension of the SA framework that allows the method to identify and incorporate algebraically smooth components into prolongation to construct a scalable multilevel solver. The use of SA as the underlying method provides a convenient way to handle problems featuring more than one distinct algebraically smooth component, such as the systems of linear elasticity.

We now briefly describe the standard SA method and the principles on which it is founded. Assume that A is of order $n = n_1$ and results from the discretization of an elliptic second- or fourth-order PDE in $\Omega \subset \mathbb{R}^d$, where $d \in \{1, 2, 3\}$. Our aim is to solve $A_1\mathbf{x} = \mathbf{b}_1$, obtained by symmetrically scaling the original system, $A\mathbf{y} = \mathbf{b}$, by its diagonal part, D:

$$A_1 = D^{-1/2}AD^{-1/2}, \qquad \mathbf{b}_1 = D^{-1/2}\mathbf{b}. \tag{2.6}$$

Guided by (2.5), a hierarchy of coarse-level operators is generated as

$$A_{l+1} = (I_{l+1}^l)^T A_l I_{l+1}^l, \tag{2.7}$$

where the *prolongator*, I_{l+1}^l, is defined as the product of a given *prolongation smoother*, S_l, and a *tentative prolongator*, P_{l+1}^l:

$$I_{l+1}^l = S_l P_{l+1}^l \tag{2.8}$$

for $l = 1, \ldots, L-1$. A simple and appropriate choice for the prolongation smoother is Richardson's method with a particular step size:

$$S_l = I - \frac{4}{3\,\lambda_l} A_l, \tag{2.9}$$

where λ_l is an upper bound on the spectral radius of the matrix on level l; $\rho(A_l) \le \lambda_l$. Suppose we are given a smoothing procedure for each level $l \in \{1, \ldots, L\}$ system, $A_l\mathbf{x} = \mathbf{b}_l$, of the form

$$\mathbf{x} \leftarrow (I - R_l A_l)\mathbf{x} + R_l\mathbf{b}_l. \tag{2.10}$$

Here, R_l is some simple approximate inverse of A_l for $l = 1, \ldots, L-1$, which we again consider to be Richardson iteration (e.g., $R_l = s_l I$, where $s_l \approx \frac{1}{\rho(A_l)}$). Assume for simplicity that the coarsest level uses a direct solver: $R_L = A_L^{-1}$. To make use of the existing convergence estimates, we assume that

$$\lambda_{\min}(I - R_l A_l) \ge 0 \quad \text{and} \quad \lambda_{\min}(R_l) \ge \frac{1}{C_R^2 \rho(A_l)}$$

for constant $C_R > 0$ independent of the level, l.

The SA iteration can formally be viewed as a standard variational multigrid process with a special choice of transfer operators I_{l+1}^l. One iteration of SA for

solving $A_1\mathbf{x}_1 = \mathbf{b}_1$ is represented by $\mathbf{x} \leftarrow \mathbf{AMG}(\mathbf{x}, \mathbf{b_1})$ and given by setting $\mathbf{AMG} = \mathbf{AMG}_1$, where $\mathbf{AMG}_l(\cdot,\cdot)$, $l = 1, \ldots, L-1$, is defined recursively as follows.

ALGORITHM 1 ($\mathbf{AMG}_l$).

1. *Presmoothing: Apply ν presmoothings to $A_l\mathbf{x}_l = \mathbf{b}_l$ of the form* $\mathbf{x}_l \leftarrow (I - R_lA_l)\mathbf{x}_l + R_l\mathbf{b}_l$.
2. *Coarse-grid correction:*
   (a) *Set* $\mathbf{b}_{l+1} = (I_{l+1}^l)^T(\mathbf{b}_l - A_l\mathbf{x}_l)$.
   (b) *Set* $\mathbf{x}_{l+1} = 0$ *and solve the coarse-level problem*

$$A_{l+1}\mathbf{x}_{l+1} = \mathbf{b}_{l+1}$$

   by γ applications of $\mathbf{x}_{l+1} \leftarrow \mathbf{AMG}_{l+1}(\mathbf{x}_{l+1}, \mathbf{b}_{l+1})$.
   (c) *Correct the solution on level l:* $\mathbf{x}_l \leftarrow \mathbf{x}_l + I_{l+1}^l\mathbf{x}_{l+1}$.
3. *Postsmoothing: Apply ν postsmoothings to $A_l\mathbf{x}_l = \mathbf{b}_l$ of the form* $\mathbf{x}_l \leftarrow (I - R_lA_l)\mathbf{x}_l + R_l\mathbf{b}_l$.

$\mathbf{AMG}_L$ returns, simply, $\mathbf{x}_L = A_L^{-1}\mathbf{b}_L$.

Note 2.1. Our selection of multigrid smoothing procedure (2.10) and prolongation smoothers S_l follows that of [20], where convergence estimates are obtained. We turn to these results for heuristics later in this section. Our task now is to focus on the construction of the tentative prolongation operators.

The construction of P_{l+1}^l consists of deriving its sparsity structure and then specifying the nonzero values. The structure of nonzeros in P_{l+1}^l can be viewed as choosing the support of spanning functions for a subspace of $\mathbb{R}^{N_l}$ to be treated by coarse-grid correction. That is, the range of P_{l+1}^l is simply the span of its columns; choosing the nonzero structure for a column of P_{l+1}^l defines the support of the basis vector (function) corresponding to that column. This structure is specified by way of a decomposition of the set of degrees of freedom associated with operator A_l into an aggregate partition, $\bigcup_{i=1}^{N_{l+1}} \mathcal{A}_i^l = \{1, \ldots, N_l\}$, $\mathcal{A}_i^l \cap \mathcal{A}_j^l = \emptyset$, $1 \le i < j \le N_{l+1}$, for $l = 1, \ldots, L-1$, where N_l denotes the number of nodes on level l. Note that the number of aggregates on level l naturally defines the number of nodes on the next level: $N_{l+1} = card(\{\mathcal{A}_i^l\})$. Let n_l denote the dimension of level l, and assume at least one degree of freedom is associated with each node, so that $n_l \ge N_l$. Aggregates $\mathcal{A}_i^l$ can be formed based only on the connectivity and strength of connection between the entries of A_l; cf. [23]. Figures 2.1 and 2.2 illustrate possible aggregates over 2D and 3D unstructured and refined meshes.

Although we illustrate these concepts in the example of a finite element discretization, where the notion of a node may be most familiar to the reader, for us a node is a strictly algebraic entity consisting of a list of degrees of freedom. In fact, the finite element analogy is only possible on the finest level; the degrees of freedom on all other levels have no explicit geometry associated with them. Thus, throughout this paper, a node on level $l+1 > 1$ is a set of degrees of freedom associated with the coarse basis functions whose discrete supports contain the same aggregate on level l. Hence, each aggregate, $\mathcal{A}$, on level l gives rise to one node on level $l+1$, and each degree of freedom associated with that node is a coefficient of a particular basis function in the coarse-level basis expansion associated with $\mathcal{A}$.

The second step in constructing generalized aggregation tentative prolongators, P_{l+1}^l, consists of endowing the sparsity structure derived from nodal aggregation with appropriate values. Starting with a given matrix, B_1, whose columns represent the near-kernel of the fine-level operator, we construct tentative prolongators and coarse-

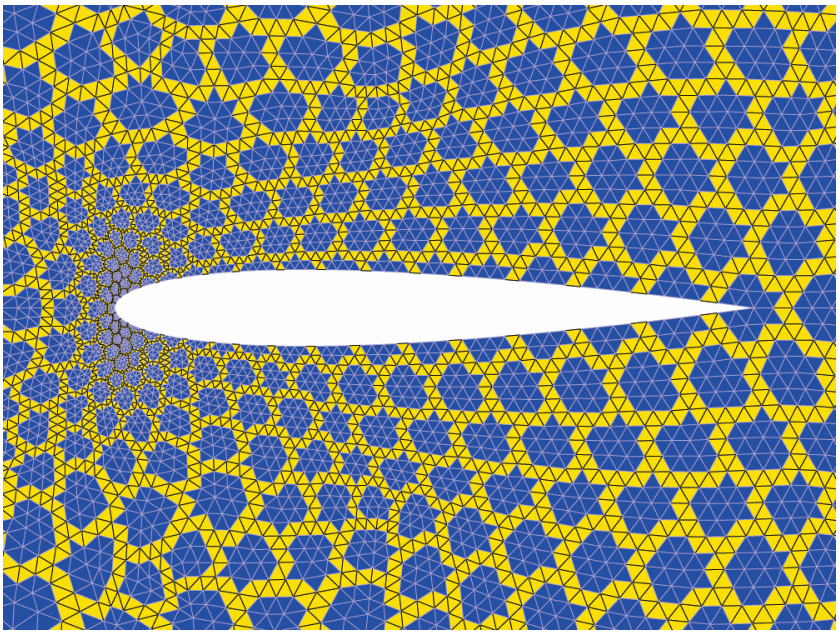

Fig. 2.1 *Possible aggregate configuration over a* 2D *locally refined unstructured mesh. Light-colored mesh edges connect nodes belonging to the same aggregate, while black edges connect nodes belonging to different aggregates.*

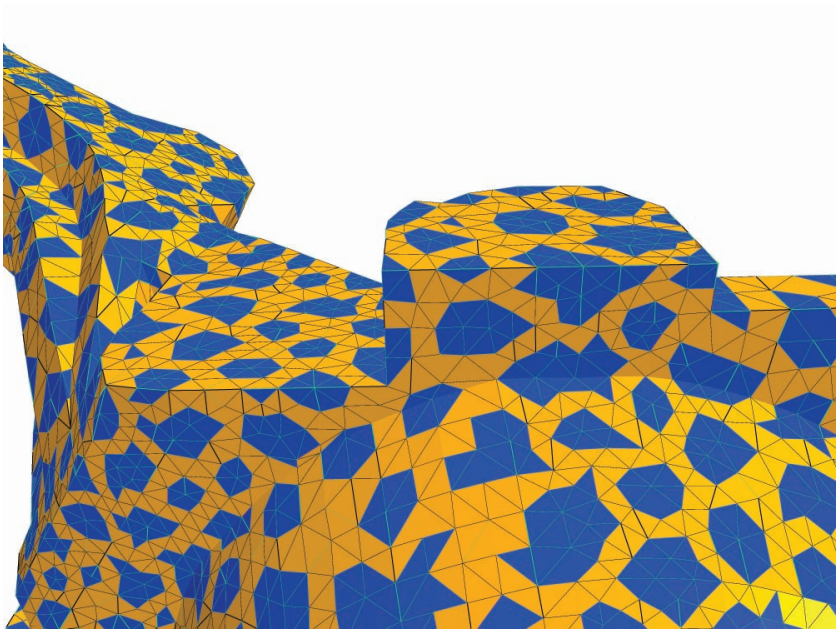

Fig. 2.2 *Possible aggregate configuration over a* 3D *unstructured tetrahedral mesh. Light-colored mesh edges connect nodes belonging to the same aggregate, while black edges connect nodes belonging to different aggregates. Only the aggregate surfaces are depicted.*

level representations of the near-kernel components simultaneously to satisfy

$$P_{l+1}^l B_{l+1} = B_l, \quad (P_{l+1}^l)^T P_{l+1}^l = I. \tag{2.11}$$

This construction of P_{l+1}^l and B_{l+1} is practical and parallelizable because it is achieved by assigning each nodal aggregate a set of columns of P_{l+1}^l with a sparsity structure that is disjoint from all other columns. Thus, obtaining (2.11) amounts to solving a set of local independent orthonormalization problems in which the basis given by the fine-level near-kernel matrix, B_l, restricted to the degrees of freedom of an aggregate, is orthonormalized using the QR algorithm. The resulting orthonormal basis forms the values of a block column of P_{l+1}^l, while the coefficients representing the old basis with respect to the new basis define B_{l+1}; cf. [23, 20]. For ease of discussion, we assume that B_l consists of the same number of columns over each aggregate. In section 4.2, we discuss how this assumption can be relaxed in practice.

Note 2.2. In this way, with B_1, A_1, and $\mathbf{b}_1$ given, the entire multigrid setup can be performed. This construction of the SA multigrid hierarchy, using (2.11), (2.8), and (2.7), and relying on a *given* fine-level near-kernel representation, B_1, is called the *standard SA* setup in this paper. For later reference, we outline the setup in Algorithm 2 below. For details, see [20].

ALGORITHM 2 (standard SA setup). *Given A_1, B_1, and L, do the following for $l = 1, \ldots, L-1$:*

(a) *Construct $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$ based on A_l.*
(b) *Construct B_{l+1} and P_{l+1}^l using* (2.11) *based on $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$.*
(c) *Construct the smoothed prolongator: $I_{l+1}^l = S_l P_{l+1}^l$.*
(d) *Construct the coarse matrix: $A_{l+1} = (I_{l+1}^l)^T A_l I_{l+1}^l$.*

With our choice of smoothing components and a coarsening procedure utilizing (2.11), the standard SA scheme can be proven to converge under certain assumptions on the near-kernel components alone. The following such result motivates the need for standard SA to have access to the near-kernel components and serves to motivate and guide our development of αSA. We note that the result was proved under the assumption that the aggregates used are of ideal size with mesh diameter of 3, but can be generalized for the case of larger aggregates.

Let $\langle \mathbf{u}, \mathbf{v} \rangle_{\mathcal{A}}$ denote the Euclidean inner product over the degrees of freedom corresponding to an aggregate $\mathcal{A}$, let $\|\cdot\|_{\mathcal{A}}$ be the associated norm, and denote the A_1-norm by $|||\mathbf{u}||| = \langle A_1\mathbf{u}, \mathbf{u}\rangle^{1/2}$. Let B_1 denote an $n_1 \times r$ matrix whose columns are thought to form a basis for the near-kernel components corresponding to A_1.

THEOREM 2.3 (Theorem 4.2 of [20]). *With $\tilde{\mathcal{A}}_i^l$ denoting the set of fine-level degrees of freedom corresponding to aggregate $\mathcal{A}_i^l$ on level l, assume that there exists constant $C_a > 0$ such that, for every $\mathbf{u} \in \mathbb{R}^{n_1}$ and every $l = 1, \ldots, L-1$, the following approximation property holds:*

$$\sum_i \min_{\mathbf{w} \in \mathbb{R}^r} \|\mathbf{u} - B_1\mathbf{w}\|_{\tilde{\mathcal{A}}_i^l}^2 \le C_a \frac{9^{l-1}}{\rho(A_1)} \langle A_1\mathbf{u}, \mathbf{u}\rangle. \tag{2.12}$$

Then

$$|||\mathbf{x}^* - \mathbf{AMG}(\mathbf{x}, \mathbf{b}_1)||| \le \left(1 - \frac{1}{c(L)}\right) |||\mathbf{x}^* - \mathbf{x}||| \qquad \forall \mathbf{x} \in \mathbb{R}^{n_1},$$

where $A_1\mathbf{x}^* = \mathbf{b_1}$ and $c(L)$ is a polynomial of degree 3 in L.

Since the use of (2.11) is assumed, condition (2.12) reflects an assumption on all tentative prolongators P_{l+1}^l and can be equivalently restated as

$$\sum_i \min_{\mathbf{w}\in\mathbb{R}^r} \|\mathbf{u} - P_2^1 P_3^2 \dots P_{l+1}^l B_{l+1}\mathbf{w}\|^2_{\tilde{\mathcal{A}}_i^l} \le C_a \frac{9^{l-1}}{\rho(A_1)} \langle A_1\mathbf{u},\mathbf{u}\rangle \tag{2.13}$$

for every $\mathbf{u}\in\mathbb{R}^{n_1}$ and every $l = 1,\dots,L-1$. Thus, in the context of SA, condition (2.12) can be viewed as an alternative formulation of the weak approximation property of Bramble et al. [2]. Note that the required approximation of a fine-level vector is less stringent for coarser levels. Also note that convergence is guaranteed even though no regularity assumptions have been made. Although this convergence bound naturally depends on the number of levels, computational experiments suggest that the presence of elliptic regularity in standard test problems yields optimal performance (i.e., convergence with bounds that are independent of the number of levels).

That polynomial $c(L)$ in the convergence estimate has degree 3 is an artifact of the proof technique used in [20], where no explicit assumptions are made on the smoothness of the coarse-level bases; instead, only the smoothness guaranteed by application of the simple prolongation smoother, S_l, is considered.

Notice that this convergence result hinges on the selection of B_1. In particular, B_1 and the coarse operators, B_{l+1} and P_{l+1}^l, $1\le l\le L-1$, that it induces must guarantee that the left side of (2.13) is small for any vector $\mathbf{u}$ for which $\langle A_1\mathbf{u},\mathbf{u}\rangle$ is small. Since the standard SA method requires that matrix B_1 be given as input, with the columns of B_1 representing a basis of (a superset of) the true kernel of the unconstrained weak form from which A_1 is obtained by discretization, the construction of P_{l+1}^l in (2.11) guarantees that all coarse-level representations, B_l, form a basis for (a superset of) the near-kernel of A_l, $l>1$. In general, B_1 must be chosen such that the induced prolongation operators, P_{l+1}^l, can approximate any low-energy component, $\mathbf{e}$, of the error with an accuracy inversely proportional to $\langle A_1\mathbf{e},\mathbf{e}\rangle$. The purpose of this paper is to enrich a given incomplete (possibly even empty) set of near-kernel components, with approximations computed at runtime in such a way that good convergence can be recovered. The adaptive method that we develop for this purpose can then be viewed as an iterative attempt to satisfy (2.13) heuristically (see Note 4.2 below). Our B_1 is computed only approximately, which means that coarse-level B_l obtained by (2.11) alone may not be the optimal representation of the near-kernel. To remedy this, we carry out the setup computation also on the coarse levels to improve on the initial guess for the coarse-level candidates given by (2.11).

3. The Adaptive Multigrid Methodology. The aim of our adaptive algorithm is to automatically construct an efficient multilevel method. We achieve this by computing a small number of vectors that represent all error components that our method is slow to resolve. Such components are usually referred to by the terms *algebraically smooth*, *near-nullspace*, *near-kernel*, or, in the case of linear elasticity, *rigid body modes*. Here, we simply call them *candidates* because they are generally not fully representative of the true near-kernel until the final stages of the process, and because we use them here as generators for the subspace from which we extract the information needed to construct the coarse spaces in our multigrid method.

Several general concepts seem key to the development of an effective adaptive process. We discuss the more important ones here before we describe our particular development of αSA.

Let Relaxation Expose Its Own Weakness. The principle of complementarity between relaxation and coarse-grid correction on which multigrid is based requires that the error components not efficiently reduced by relaxation be represented in the range of interpolation. In the absence of explicit information about the problem to be solved, we can gain information about the nature of errors that the relaxation does not efficiently reduce by applying the method itself to a problem with known solution. The homogeneous equation,

$$A\mathbf{x} = \mathbf{0}, \tag{3.1}$$

serves us well for this purpose. Typically, relaxation applied to this equation quickly yields an error, $\mathbf{x}$, that is nearly the slowest to vanish. What is at work here is that this is just the power method for computing the dominant eigenvector of the iteration matrix. Thus, a few relaxations on (3.1), starting perhaps from a random initial guess, are enough not only to determine whether coarsening is even needed (by measuring the energy convergence factors), but also to produce a candidate that is at least a crude representative of troublesome errors.

Use the Candidate to Enrich Prolongation. Once we have computed a candidate, we want it to be near the range of the prolongator, so that a coarse-grid correction can eliminate it. The best way to make this approximation exact is by ensuring that the candidate is exactly in the range of the prolongator. SA greatly facilitates this process because the tentative prolongator is constructed through *localizing* any given candidate by restricting it to mutually disjoint aggregates. The candidate can then be recovered as a sum over all aggregates of the individual columns of the resulting tentative prolongator. This exactness means that the variational two-level scheme given by (2.3)–(2.5) would exactly eliminate errors in the direction of this candidate. In SA, however, the final prolongator is constructed by applying smoothing to the tentative prolongator. The candidate may no longer be exactly in the range of the smoothed prolongator, but the appropriately smoothed candidate is in this range and it should be at least as good a candidate as the original. In any case, this smoothing is important for achieving scalable multilevel convergence rates.

It is important to acknowledge the role played by the underlying multigrid method in allowing the use of each candidate as a *representative* of many errors with similar local character. Namely, the process of incorporating a candidate in the range of the prolongator results in localization of the candidate and defines a coarse space rich enough to accurately approximate not just the candidate, but also a whole subspace of error components that have a *local* character similar to that of the candidate. This approximation property is analogous to how local finite element bases of piecewise polynomials are used to approximate low-energy (geometrically smooth) components of the PDE itself.

Improve the Candidates with the Emerging Cycle. After just a few relaxation steps, algebraic smoothness of the candidate may already be sufficient to produce coarsening that is effective in a two-level mode, but it may not yet be enough to define accurate interpolation between the coarser levels needed for a V-cycle. More precisely, we expect that, after a few relaxation steps, convergence slows down to yield a convergence factor that approximates the spectral radius of the error propagation matrix. The candidate that reflects this slow convergence may be smooth enough that the induced interpolation approximates all smooth components with accuracy that is sufficient to guarantee optimal two-level multigrid convergence. However, V-cycles

may require more accuracy than just a fixed number of decimal places. Numerical and theoretical results suggest that at least an $O(h^2)$ approximation to the spectral radius of the error propagation matrix may be necessary for typical second-order elliptic problems. How can the candidate be improved to that degree? The answer again is to use the method itself. As a fine-level candidate becomes available, a prolongation operator can be constructed that defines the next coarser-level problem. A coarse-level representation of the candidate is obtained as a byproduct of this construction and can be improved by relaxing it on the coarse level. The prolongation provides a means of using the updated coarse-level candidate to improve the fine-level candidate on a coarser scale. Thus, it is the evolving V-cycle, instead of relaxation by itself, that is used to improve the candidate. But this must be done with care. A straightforward application of this principle to (3.1) would yield a zero candidate when the current candidate is in the range of interpolation because (3.1) is then solved exactly. It is important to recognize that our true goal is not really to solve (3.1) but to find the maximal eigenvector of the error propagation matrix for relaxation. Note that this gives the coarse level a dual role: it must develop its own fast solver (for (2.5)), but it must also improve the fine-level candidate.

Let the V-cycle Expose Its Own Weakness. Many problems, including most scalar second-order elliptic PDEs discretized on quasi-uniform meshes, can be treated by AMG and SA using a single near-kernel component as a candidate. However, several near-kernel components may be required for other applications, such as for elasticity and other systems of PDEs and even for scalar equations with reduced ellipticity or higher-order equations. We seek a set of candidates that is or becomes *rich* in the sense that they combine to *locally* represent *all* troublesome components. How, then, after we have computed the first candidate and the V-cycle that it induces, can we compute a second candidate that is *locally* distinct from the first? Consider, again, the first principle: relaxation applied to the homogeneous equation not only signals when coarsening is needed, but it also produces the first candidate. We take our cue by applying the first V-cycle to the homogeneous equation, which signals when more coarsening is needed. This also produces a new candidate that must be distinct from the first one locally, on average, because it would otherwise be eliminated quickly by the current V-cycle. In fact, this distinction is just in the right sense because the new candidate represents error that is not quickly attenuated by the method. Note that using this *fixed* V-cycle constructed in the first adaptive sweep is different from using the *evolving* V-cycle in the current adaptive sweep: while both are applied to (3.1), the object of using the fixed V-cycle is to expose a new candidate that it cannot effectively reduce, while the objective of using the evolving V-cycle is to simultaneously improve this new candidate and improve itself by using the candidate to enrich the first V-cycle's interpolation. Note, also, that recursion demands that we apply the fixed V-cycle on coarser levels in order to improve the fine-level candidate as a representative of error not reduced by this cycle. Note, finally, that we can iterate on this whole process to introduce several candidates that are mutually distinct in the correct sense.

Maintain Accuracy of Coarsening for All Candidates. SA easily handles multiple candidates by restricting all of them to each aggregate, thereby allowing coarsening to pay attention to new candidates without corrupting the accuracy of the prolongator with respect to the previous candidates. To ensure that the candidate information is used to produce good coarse-level bases, SA uses a local orthogonalization process with a drop tolerance to eliminate local dependencies within every aggregate. One

might thus be tempted to compute several candidates simultaneously, starting perhaps with several random initial guesses. However, it is not at all straightforward to design generally reliable measures to resolve possible dependencies; handling multiple approximate candidates simultaneously risks numerical redundancy on coarser levels, which might lead to spurious near-kernel components and inflated complexity. In contrast, using the V-cycle to compute its own troublesome components one at a time means that these candidates should be separate in the right sense globally and, on average, locally, thus aiding the construction of the prolongation operators. Also, the quality of each computed candidate has an impact on the speed with which further candidates are recovered, so it is important to be reluctant to add new candidates before the current ones are fully improved and exploited. It is likely to be much more robust to have the algorithm prevent redundancy by introducing candidates individually and fully improving them deliberately.

Other Principles. There are several other issues that need to be considered for the design of an effective adaptive strategy. Although we do not explicitly address these principles further, we list them here:

1. The scheme should be optimal in the sense that candidates and coarsening processes should be updated as soon as it is necessary and permissible to do so.
2. Since A is SPD, it is advisable to make use of the energy minimization principle whenever possible (e.g., in measuring convergence).
3. Appropriate measures should be used for making decisions at several stages (e.g., continue to improve the current candidate if the convergence factors for the homogeneous equation are poor and continuing to worsen; otherwise, if the factors have stalled at poor values, add a new candidate and focus on improving it).
4. Do not return from the coarse level until an acceptable solver has been developed for it (because it would otherwise be difficult to determine if poor fine-level V-cycle performance was then a result of an inadequate coarse-level solver or poor quality of prolongation).

4. Adaptive Smoothed Aggregation.

Before describing the algorithm, we emphasize our notational conventions. The transfer operators and coarse-level problems, as well as other components of our multigrid scheme, change as our method adapts. Whenever possible, we use the same symbols for the updated components. Thus, symbol B_l may denote a single column vector in one cycle of the setup procedure or perhaps a two-column matrix in the next step of the setup. The intended meaning should be clear from context.

4.1. Initialization Setup Stage.

The adaptive multigrid setup procedure considered in this paper can be split into two stages. If no knowledge of the near-kernel components of A_1 is available, then we start with the first stage to determine an approximation to one such component. This stage also determines the number of levels, L, to be used in the coarsening process. (Changing L in the next stage based on observed performance is certainly possible, but it is convenient to fix L—and other constructs—early in the setup phase.)

Let $\varepsilon > 0$ be a given convergence factor tolerance.

ALGORITHM 3 (initialization stage).

1. *Set* $l = 1$ *and select a random vector,* $\mathbf{x}_1 \in \mathbb{R}^{n_1}$.
2. *With initial approximation* $\mathbf{x}_1$*, relax* μ *times on* $A_1\mathbf{x} = 0$*, denoting* k*th iter-*

ation by $\mathbf{x}_1^k$:

$$\mathbf{x}_1^\mu \leftarrow (I - R_1 A_1)^\mu \mathbf{x}_1^0, \quad \mathbf{x}_1 \leftarrow \mathbf{x}_1^\mu.$$

3. *If* $\langle A_1 \mathbf{x}_1^\mu, \mathbf{x}_1^\mu \rangle \leq \varepsilon \langle A_1 \mathbf{x}_1^{\mu-1}, \mathbf{x}_1^{\mu-1} \rangle$, *then set* $L = 1$ *and* **stop** *(problem* $A_1 \mathbf{x} = \mathbf{b}_1$ *can be solved fast enough by relaxation alone, so only one level is needed).*
4. *Otherwise, do the following:*
   (a) *Set* $B_l \leftarrow \mathbf{x}_l$.
   (b) *Create a set,* $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$, *of nodal aggregates based on matrix* A_l.
   (c) *Define tentative prolongator* P_{l+1}^l *and candidate matrix* B_{l+1} *using candidate matrix* B_l *and relations* (2.11) *with structure based on* $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$.
   (d) *Define the prolongator:* $I_{l+1}^l = S_l P_{l+1}^l$.
   (e) *Define the coarse matrix:* $A_{l+1} = (I_{l+1}^l)^T A_l I_{l+1}^l$. *If level* $l+1$ *is coarse enough that a direct solver can be used there, skip to step* 5; *otherwise, continue.*
   (f) *Set the next-level approximation vector:* $\mathbf{x}_{l+1} \leftarrow B_{l+1}$.
   (g) *Make a copy of the current approximation:* $\hat{\mathbf{x}}_{l+1} \leftarrow \mathbf{x}_{l+1}$.
   (h) *With initial approximation* $\mathbf{x}_{l+1}$, *relax* μ *times on* $A_{l+1}\mathbf{x} = 0$:

$$\mathbf{x}_{l+1} \leftarrow (I - R_{l+1} A_{l+1})^\mu \mathbf{x}_{l+1}.$$

   (i) *If* $(\frac{\langle A_{l+1}\mathbf{x}_{l+1}, \mathbf{x}_{l+1} \rangle}{\langle A_{l+1}\hat{\mathbf{x}}_{l+1}, \hat{\mathbf{x}}_{l+1} \rangle})^{1/\mu} \leq \varepsilon$, *skip steps* (f)–(i) *in further passes through step* 4.
   (j) *Increment* $l \leftarrow l+1$ *and return to step* 4(a).
5. *Set* $L \leftarrow l+1$ *and update the finest-level candidate matrix:*

$$B_1 \leftarrow I_2^1 I_3^2 \ldots I_{L-1}^{L-2} \mathbf{x}_{L-1}.$$

6. *Create the* V*-cycle based on* B_1 *using the standard SA setup of Algorithm* 2, *with the exception that the aggregates are predetermined in step* 4.

This initialization stage terminates whenever a level is reached in the coarsening process where a direct solver is appropriate. It does not involve level L processing because it is assumed that the coarsest level is handled by a direct solver, making the stopping criterion in step 4(i) automatically true. Note that the candidate matrix is actually a vector in this initial stage because we are computing only one candidate and that this stage provides all of the components needed to construct our initial V-cycle solver, $\mathbf{AMG}_1$.

If the criterion tested in step 4(i) is satisfied, we are assured that the current coarse level $l+1$ can be easily solved by relaxation alone. At that point, we could choose not to coarsen further and use relaxation as a coarsest-level solver. However, it is possible that the general stage of the algorithm described below adds more candidates. In case a new candidate approximates the low-energy modes of the problem better than the candidate obtained in the initial step, the coarse-level matrix may no longer be easily solved by relaxation alone. Thus, we choose to coarsen further, until we are sure that the coarsest problem can be handled well. This offers an added benefit of producing, at the end of the initial stage, a complete aggregation that can be reused in the general stage. Note that if step 4(i) is satisfied, then the approximate solution of the homogeneous problem may be zero. In such a case, we restore the saved original vector, $\hat{\mathbf{x}}_{l+1}$. We choose to skip steps 4(f)–(i) in further coarsening once step 4(i) is satisfied. This amounts to using standard SA coarsening from level $l+1$ down, which guarantees that the candidate computed on level l is exactly represented all the way

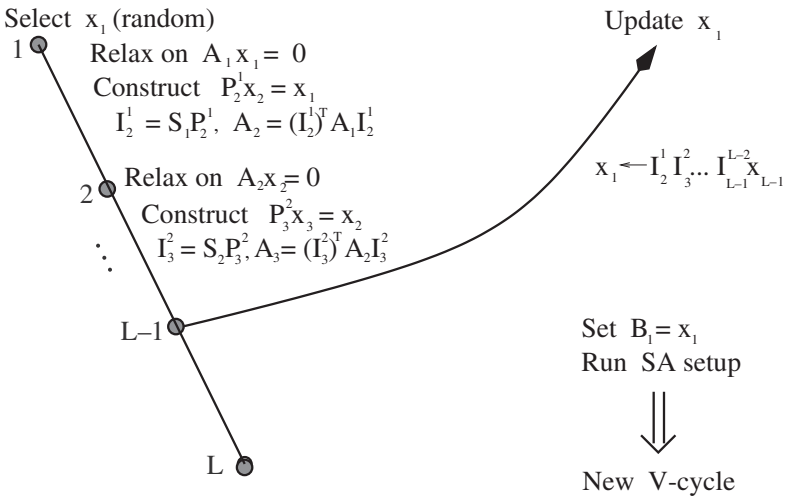

Fig. 4.1 *Initialization stage, Algorithm* 3.

to the coarsest level. Additional modifications to Algorithm 3 are, of course, possible. For instance, the computation of $\mathbf{x}_1^\mu$ in step 2 may be terminated before reaching the specified number of iterations whenever $\langle A_1\mathbf{x}_1^k, \mathbf{x}_1^k\rangle = 0$ for some $k = 1, \ldots, \mu - 1$. In that case, we would update $\mu \leftarrow k$ in step 2. Figure 4.1 illustrates Algorithm 3.

Note 4.1. The initialization stage described in Algorithm 3 is used only if no knowledge of the near-kernel components is provided. In many situations, however, some knowledge may be available and should be used. In such cases, the initialization stage can be omitted and the initial B_1 can be assumed to consist of the given set of vectors. The initial V-cycle would then be constructed exactly as in Algorithm 2 prior to running the main adaptive setup.

As an example, consider a problem of 3D linear elasticity discretized by a standard linear first-order finite element method over an unstructured mesh. In this case, if the discretization package either generates the rigid body modes or supplies the nodal geometry to the solver, then the full set of kernel vectors is presumably available [22] and the adaptive process may be unnecessary. Otherwise, when the full set of rigid body modes is unavailable, it is nevertheless often possible to obtain a subset of the rigid body modes consisting of three independent constant displacements, regardless of the geometry of the mesh. Such a subspace should be used whenever possible to create B_1 and to set up a V-cycle exactly as in the standard SA method. The initialization stage would then be omitted.

Thus, the initialization stage given by Algorithm 3 should be viewed as optional, to be done only if no information can be assumed about the system to be solved. In view of Note 4.1, we can in any case assume that the initial B_1 has at least one column and that a tentative V-cycle is available. This means that we have constructed aggregates $\mathcal{A}_i^l$, transfer operators P_{l+1}^l and I_{l+1}^l, and coarse operators A_{l+1}, $l = 1, \ldots, L-1$.

4.2. General Setup Stage. In each step of the second stage of the adaptive procedure, we apply the current V-cycle to the homogeneous problem to uncover error components that are not quickly attenuated. The procedure then updates its own

transfer operators to ensure that these components are eliminated by the improved method, while preserving the previously established approximation properties. Thus, this stage essentially follows the initialization stage with relaxation replaced by the current V-cycle.

One of the subtleties of this approach lies in the method's attempt to update each level of the evolving V-cycle as soon as its ineffectiveness is exposed. Thus, on the finest level in the second stage, the current V-cycle simply plays the role of relaxation: if it is unable to quickly solve the homogeneous problem (i.e., step 3 fails), then the resulting error becomes a new candidate, and new degrees of freedom are generated accordingly on level 2 (i.e., columns are added to B_1). The level 2-to-L part of the old V-cycle (i.e., the part without the finest level) then plays the role of the level 2 relaxation in the initial setup phase and is thus applied to the homogeneous problem to assess the need to improve its coarser-level interpolation operators. The same is done on each coarser level, l, with the level l-to-L part of the old V-cycle playing the role of the level l relaxation step in the initial setup phase. The process continues until adequate performance is observed or the maximum permitted number of degrees of freedom per node is reached on coarse levels.

The general stage uses the current solver to identify new types of error that the earlier sweeps of the setup cycle may have missed. It is important to note that we are talking here about error *type*. It is not enough for the coarsening process to eliminate only the particular candidate; typically, a fixed percentage of the spectrum of A_1 is algebraically smooth, so elimination of one candidate at a time would require $O(n_1)$ setup cycles to achieve a quickly converging solver. Thus, to avoid this unacceptably large cost, each setup cycle must determine interpolation operators so that the solver eliminates a relatively large set of errors of each candidate's type. Just as each rigid body mode is used locally in standard SA to treat errors of similar type (constants represent errors that are smooth within variables and rotations represent intervariable "smoothness"), so too must each candidate be used in αSA. Moreover, a full set of types must be determined if the solver is to attain full efficiency (e.g., for 2D linear elasticity, three rigid body modes are generally needed). We thus think of each candidate as a sort of *straw man* that represents a whole class of algebraically smooth components. Efficient computation of a full set of straw men is the responsibility of the adaptive process. However, proper treatment of each straw man is the task of the basic solver, which is SA in this case.

We present a general prototype algorithm for the adaptive multigrid setup, assuming that a tentative V-cycle has previously been constructed (cf. Note 4.1). We thus assume that a current hierarchy of nodal aggregates, $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$, and operators, $P_{l+1}^l, I_{l+1}^l, A_{l+1}$, are available for all $l = 1, \ldots, L-1$. Consider, then, a method in which, within each cycle of the adaptive setup, we attempt to update the current V-cycle level by level. One cycle of this adaptive setup traverses from the finest to the coarsest level; on each level l along the way, it updates B_l based on computing a new candidate from the current multigrid scheme applied to the homogeneous problem on level l. Thus, on level l in the setup process, a solver is applied that traverses from that level to level L and back. This gives us the picture of a backward *full multigrid* (FMG) cycle, where the setup traverses from the finest to the coarsest grid and each level along the way is processed by a V-cycle solver (see Figure 4.2). Now, once this new candidate is computed, it is incorporated into the current multigrid scheme and the previously existing V-cycle components are overwritten on level $l+1$ but temporarily retained from that level down. As a result, we redefine level by level the V-cycle components. Once the new B_l (and I_{l+1}^l in (2.8)) are constructed all the

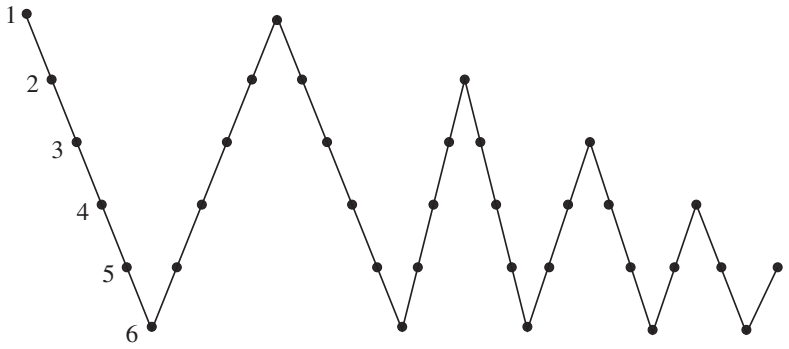

Fig. 4.2 *Self-correcting adaptive cycling scheme given by Algorithm 4, with the solver cycles uncollapsed.*

way to the coarsest level, we can then use them to update the current B_1 and, based on it, construct a new V-cycle on the finest level.

As we apply our current method to the homogeneous problem, the resulting candidate, $\mathbf{x}_l$, tends quickly to an error that is among the slowest to converge in the current method. Our goal in designing the adaptive algorithm is to ensure that $\mathbf{x}_l$ is approximated well by the newly constructed transfer operator. That is, we want to control the constant C_a in the inequality

$$\min_{\mathbf{v}\in\mathbb{R}^{n_{l+1}}} \|\mathbf{x}_l - P_{l+1}^l \mathbf{v}\|^2 \leq \frac{C_a}{\rho(A_l)} \|\mathbf{x}_l\|_{A_l}^2. \tag{4.1}$$

The transfer operators must, therefore, be constructed to give accurate approximations to each candidate as it is computed. This can be guaranteed locally by requiring that, over every aggregate $\mathcal{A}$, we have

$$\min_{\mathbf{v}\in\mathbb{R}^{n_{l+1}}} \|\mathbf{x}_l - P_{l+1}^l \mathbf{v}\|_{\mathcal{A}}^2 \leq C_a \delta_{\mathcal{A}}(\mathbf{x}_l), \tag{4.2}$$

where $\delta_{\mathcal{A}}$ are chosen so that summing (4.2) over all aggregates leads to (4.1), i.e., so that

$$\sum_{\mathcal{A}} \delta_{\mathcal{A}}(\mathbf{x}) = \frac{\langle A_l \mathbf{x}, \mathbf{x}\rangle}{\rho(A_l)}. \tag{4.3}$$

For now, the only assumption we place on $\delta_{\mathcal{A}}(\mathbf{x})$ is that (4.3) holds. An appropriate choice for the definition of $\delta_{\mathcal{A}}(\mathbf{x})$ is given in Note 4.7.

Note 4.2 (relationship to theoretical assumptions). To relate condition (4.1) to the theoretical foundation of SA, we make the following observation. If P_{l+1}^l is constructed so that (4.1) is satisfied for the candidate $\mathbf{x}_l$, the construction of our method automatically guarantees that

$$\min_{\mathbf{v}\in\mathbb{R}^{n_{l+1}}} \|\mathbf{x}_1 - P_2^1 P_3^2 \dots P_{l+1}^l \mathbf{v}\|^2 \leq \frac{C_a}{\rho(A_l)} \|\hat{\mathbf{x}}_l\|_{A_1}^2, \tag{4.4}$$

where $\mathbf{x}_1 = P_2^1 P_3^2 \dots P_l^{l-1} \mathbf{x}_l$ and $\hat{\mathbf{x}}_1 = I_2^1 I_3^2 \dots I_l^{l-1} \mathbf{x}_l$. Since it is easy to show that $\|\hat{\mathbf{x}}\|_{A_1} \leq \|\mathbf{x}\|_{A_1}$, we can then guarantee that (2.13) holds for the particular fine-level

candidate, $\mathbf{x}_1$. Inequality (4.1) is easily satisfied for any component $\mathbf{u}$ for which $\|\mathbf{u}\|_{A_1}$ is bounded away from zero. We can thus focus on the troublesome subspace of components with small energy. Our experience with the standard SA method indicates that for the second- and fourth-order elliptic problems it suffices to ensure that the components corresponding to the kernel of the weak form of the problem are well approximated by the prolongation (the near-kernel components are then well approximated due to the localization and smoothing procedures involved in constructing the SA transfer operators). Further, as the set of candidates constructed during the setup cycle is expected to eventually encompass the entire troublesome subspace, satisfaction of (4.1) for all candidates would imply the satisfaction of (2.13) for any $\mathbf{u} \in \mathbb{R}^{n_1}$. This, in turn, guarantees convergence.

Note 4.3 (locally small components). Each new candidate is the result of applying the V-cycle based on the current B_1, so it must be approximately A_1-orthogonal to all previously computed candidates. This is, however, only a global property that the evolving candidates tend to exhibit. It may be that a candidate is so small on some aggregate, relative to its energy, that its representation there can be ignored. More precisely, we could encounter situations in which

$$\|\mathbf{x}_l\|_{\mathcal{A}}^2 \leq C_a \delta_{\mathcal{A}}(\mathbf{x}_l) \tag{4.5}$$

for a particular aggregate, $\mathcal{A}$, meaning that (4.2) is automatically satisfied no matter what choice we make for P_{l+1}^l. We can, therefore, test for this condition for each candidate on every aggregate. When the test is positive, we can simply remove the candidate's segment from consideration in construction of that aggregate's transfer operator. This elimination can help control coarse-level complexity since small candidate segments are prevented from generating additional columns of P_{l+1}^l and I_{l+1}^l. (This test could be used in the initialization as well as the general setup stage. However, the complexity is generally so low in the initial stage that the elimination is not required there.)

Note 4.4 (reusing previously constructed components). To exploit the work done in the earlier steps of the setup as much as possible, we consider a procedure that reuses parts of P_{l+1}^l that have already been computed. Thus, in each step of the setup, we consider only adding a single new column to P_{l+1}^l. This has the advantages that less work is required and that the storage used to hold the global candidates can be reused as soon as they have been incorporated into P_{l+1}^l.

In this approach, to minimize the complexity of the transfer operators, we seek to ignore locally those components of candidate $\mathbf{x}_l$ that appear to be well approximated by the current transfer operators. This includes the case when $\mathbf{x}_l$ is locally small in the sense of (4.5). To decide whether to ignore $\mathbf{x}_l$ locally in the construction of *new* tentative prolongator P_{l+1}^l, we test how well it is approximated by the *current* tentative prolongator, $\tilde{P}_{l+1}^l$. The following provides a test of how well the range of $\tilde{P}_{l+1}^l$ approximates $\mathbf{x}_l$ over aggregate $\mathcal{A}$:

$$\|\mathbf{x}_l - \tilde{P}_{l+1}^l (\tilde{P}_{l+1}^l)^T \mathbf{x}_l\|_{\mathcal{A}}^2 \leq C_a \delta_{\mathcal{A}}(\mathbf{x}_l). \tag{4.6}$$

(Since $(\tilde{P}_{l+1}^l)^T \tilde{P}_{l+1}^l = I$, then $\tilde{P}_{l+1}^l (\tilde{P}_{l+1}^l)^T$ is the L^2 projection onto the range of $\tilde{P}_{l+1}^l$; thus, (4.6) is just approximation property (4.2) using the tentative prolongator in place of the smoothed one.) If (4.6) is satisfied, then $\mathbf{x}_l$ is assumed to be well approximated by the current transfer operator and is simply ignored in the construction of the new transfer operator on aggregate $\mathcal{A}$. (Practical implications of this local elimination

from the coarsening process are considered in Note 4.6.) If the inequality is not satisfied, then we keep the computed vector, $\mathbf{y} = \mathbf{x}_l - \tilde{P}^l_{l+1}(\tilde{P}^l_{l+1})^T\mathbf{x}_l$, which, by construction, is orthogonal to all the vectors already represented in the current $\tilde{P}^l_{l+1}$. We then normalize via $\mathbf{y} \leftarrow \mathbf{y}/\|\mathbf{y}\|_{\mathcal{A}}$ so that the new P^l_{l+1} has orthonormal columns: $(P^l_{l+1})^T P^l_{l+1} = I$.

To obtain a practical method, several issues must be addressed. These issues are discussed below, where we take advantage of the SA framework to carry out the method outlined in Algorithm 4, as well as to control the amount of work required to keep the evolving coarse-level hierarchy up-to-date. For instance, when using a coarse-level V-cycle constructed by previous applications of the setup stage, we must deal with the fact that the number of vectors approximated on coarse levels in previous cycles is smaller than the number of vectors approximated on the fine levels in the current cycle; hence the current multigrid transfer operators between these two levels are invalid. Also, as suggested in Note 4.4, a candidate may occasionally be eliminated locally over an individual aggregate. This results in varying numbers of degrees of freedom per node on the coarse levels. (Recall that a coarse-level node is defined as a set of degrees of freedom, each representing the restriction of a single candidate to a fine-level aggregate.) To simplify notation, we assume for the time being that the number of degrees of freedom per node is the same for all nodes on a given level (i.e., no candidates are locally eliminated). It is important, however, to keep in mind that we are interested in the more general case. A generalization to varying numbers of degrees of freedom per node could be obtained easily at the cost of a much more cumbersome notation. We briefly remark on the more general case in Note 4.6 below.

Note 4.5 (construction of temporary "bridging" transfer operators). An issue we must consider is the interfacing between the emerging V-cycle on finer levels and the previous V-cycle on coarser levels. Each setup cycle starts by selecting an initial approximation for a new candidate on the finest level (cf. Figure 4.3). This approximation is then improved by applying the error propagation matrix for the previously constructed V-cycle to it. The resulting candidate is used to enrich B_1. This necessitates an update of P^1_2, I^1_2, and A_2 from (2.11) and (2.7) and introduces an additional degree of freedom for the nodes on level 2. Since we now want to run the current solver on level 2 to obtain an improved candidate on that level, we need to temporarily modify P^2_3 and I^2_3 because these transfer operators have not yet been updated to reflect the added degrees of freedom on level 2. Once this modification has been made, a V-cycle on level 2 can be run to compute the new candidate there. This candidate is then incorporated into B_2 and new P^2_3 and I^2_3 are constructed, overwriting the temporary versions, and the new A_3 can be computed using (2.7). To perform the V-cycle on level 3, we then must temporarily modify operators P^3_4 and I^3_4 for the same reason we had to update P^2_3 and I^2_3 above. Analogous temporary modifications to the transfer operators are necessary on all coarser levels, as the setup cycle traverses sequentially through them.

Thus, on stage l of a single cycle of the setup process, all transfer operators defining the V-cycle can be used without change, except for P^l_{l+1} and, consequently, I^l_{l+1} defined through (2.8). We can construct the temporary operator, P^l_{l+1}, by modifying (2.11) as

$$P^l_{l+1}B_{l+1} = \hat{B}_l,$$

where $\hat{B}_l$ is formed by removing the last column from B_l, which consists of the $k+1$ fine-level candidate vectors, including the newly added one (so that the first k columns

represent the same candidates as in the previous cycle). Since tentative prolongator P_{l+1}^l produced in this way is based only on fitting the first k vectors in B_l, the coarse-level matrix A_{l+1} resulting from the previous cycle of the αSA setup (described below) can be used on the next level. Thus, all the coarse operators for levels coarser than l can be used without change. This has the advantage of reducing the amount of work to keep the V-cycle up-to-date on coarser, yet-to-be-traversed levels.

So far, we have considered only the case where all candidates are used locally. In the interest of keeping only the candidates that are essential to achieving good convergence properties, we now consider practical aspects of locally eliminating the candidates where appropriate.

Note 4.6 (eliminating candidates locally as suggested in Note 4.4). When we eliminate a candidate locally over an aggregate as suggested in Note 4.4, the construction of the bridging operator above can be easily modified so that the multigrid hierarchy constructed in the previous setup cycle can be used to apply a level l V-cycle in the current one. Since the procedure guarantees that the previously selected candidates are retained and only the newly computed candidate may be locally eliminated, the V-cycle constructed in the previous setup cycle remains valid on coarser levels as in the case of Note 4.5. The only difference now is that aggregates may have a variable number of associated candidates, and the construction of the temporary transfer operator, P_{l+1}^l, described in Note 4.5 must account for this when removing the column of B_l to construct $\hat{B}_l$.

Note 4.7 (selection of the local quantities $\delta_{\mathcal{A}}(\mathbf{x})$). Our algorithm relies on local aggregate quantities $\delta_{\mathcal{A}}(\mathbf{x})$ to decide whether to eliminate candidate $\mathbf{x}$ in aggregate $\mathcal{A}$, and to guarantee that the computed candidates satisfy the global approximation property (4.1). This leads us to the choice

$$\delta_{\mathcal{A}}(\mathbf{x}) = \left(\frac{\text{card}(\mathcal{A})}{N_l}\right) \frac{\langle A_l \mathbf{x}, \mathbf{x}\rangle}{\rho(A_l)}, \tag{4.7}$$

where $\text{card}(\mathcal{A})$ denotes the number of nodes in aggregate $\mathcal{A}$ on level l, and N_l is the total number of nodes on that level. Note that $\sum_{\mathcal{A}} \delta_{\mathcal{A}}(\mathbf{x}) = \frac{\langle A_l \mathbf{x}, \mathbf{x}\rangle}{\rho(A_l)}$ for any $\mathbf{x}$, so this can be used in local estimates (4.2) to guarantee (4.1).

Having discussed the modifications that may be necessary, we are now ready to give the algorithm for the general stage of αSA. Assume we are given a bound, $K \in \mathbb{N}$, on the number of degrees of freedom per node on coarse levels, convergence factor tolerance $\varepsilon \in (0,1)$, and aggregate quantities $\delta_{\mathcal{A}}(x)$ such that $\sum_{\mathcal{A}} \delta_{\mathcal{A}}(x) = \frac{\langle A_l \mathbf{x}, \mathbf{x}\rangle}{\rho(A_l)}$. Then one step of the general setup stage proceeds as follows.

ALGORITHM 4 (one cycle of the general αSA setup stage).

1. *If the maximum number of degrees of freedom per node on level* 2 *equals* K, **stop** *(the allowed number of coarse-level degrees of freedom has been reached).*
2. *Create a copy of the current* B_1 *for later use:* $\hat{B}^1 \leftarrow B_1$.
3. *Select a random* $\mathbf{x}_1 \in \mathbb{R}^{n_1}$ *and apply* μ *iterations of the current* V*-cycle, denoting the* k*th iteration by* $\mathbf{x}_1^k$:
$$\mathbf{x}_1^\mu \leftarrow \mathbf{AMG}_1^\mu(\mathbf{x}_1^0, \mathbf{0}), \quad \mathbf{x}_1 \leftarrow \mathbf{x}_1^\mu.$$
4. *If* $\langle A_1 \mathbf{x}_1^\mu, \mathbf{x}_1^\mu\rangle \le \varepsilon \langle A_1 \mathbf{x}_1^{\mu-1}, \mathbf{x}_1^{\mu-1}\rangle$, *then* **stop** *(*$A_1\mathbf{x} = \mathbf{b}_1$ *can be solved quickly enough by the current method).*
5. *Update* B_1 *by extending its range with the new column* $\{\mathbf{x}_1\}$:
$$B_1 \leftarrow [B_1, \mathbf{x}_1].$$

6. *For* $l = 1, \ldots, L-2$:
   (a) *Define a new coarse-level matrix* B_{l+1} *and transfer operator* P_{l+1}^{l} *based on* (2.11), *using* B_l *and decomposition* $\{\mathcal{A}_i^l\}_{i=1}^{N_l}$. *In creating* P_{l+1}^{l}, *some local components in* B_l *may be locally eliminated as suggested in Note* 4.4.
   (b) *Construct the prolongator:* $I_{l+1}^{l} = S_l P_{l+1}^{l}$.
   (c) *Construct the coarse operator:* $A_{l+1} = (I_{l+1}^{l})^T A_l I_{l+1}^{l}$.
   (d) *Reorder the columns of* B_{l+1} *so that its last is* $\mathbf{x}_{l+1}$, *and let* $\hat{B}^{l+1}$ *consist of all other columns of* B_{l+1}.
   (e) *Create a "bridge" transfer operator* P_{l+2}^{l+1} *to the coarser level with the old* B_{l+1} *by fitting all the vectors in* B_{l+1} *except the last one; see Note* 4.5.
   (f) *Set the new "bridging" prolongator:* $I_{l+2}^{l+1} = S_{l+1} P_{l+2}^{l+1}$.
   (g) *Make a copy:* $\hat{\mathbf{x}}_{l+1} \leftarrow \mathbf{x}_{l+1}$.
   (h) *Apply* μ *iterations:* $\mathbf{x}_{l+1} \leftarrow \mathbf{AMG}_{l+1}^{\mu}(\mathbf{x}_{l+1}, \mathbf{0})$.
   (i) *If* $(\frac{\langle A_{l+1}\mathbf{x}_{l+1}, \mathbf{x}_{l+1}\rangle}{\langle A_{l+1}\hat{\mathbf{x}}_{l+1}, \hat{\mathbf{x}}_{l+1}\rangle})^{1/\mu} \leq \varepsilon$, *then skip* (d) *through* (j) *in further passes through step* 6.
   (j) *Update the coarse representation of candidate* B_{l+1}:

$$B_{l+1} \leftarrow [\hat{B}^{l+1}, \mathbf{x}_{l+1}].$$

7. *Update the latest fine-level candidate:*

$$\mathbf{x}_1 \leftarrow I_2^1 I_3^2 \ldots I_{L-1}^{L-2} \mathbf{x}_{L-1}. \tag{4.8}$$

8. *Update* B_1 *by extending the old copy with the newly computed* $\mathbf{x}_1$:

$$B_1 \leftarrow [\hat{B}^1, \mathbf{x}_1].$$

9. *Create the* V*-cycle based on the current* B_1 *using the standard SA setup described by Algorithm* 2.

Algorithm 4, which is illustrated in Figure 4.3, starts from a V-cycle on input and produces an improved V-cycle as output. It stops iterating when either the convergence factor for the fine-level iteration in step 3 is acceptable (as measured in step 4) or the maximum number of iterations is reached. Note that, as with the initial stage, this general stage does not involve level L processing because the coarsest level is assumed to be treated by a direct solver. Also as in the initial stage, once a level is reached where the problem can be solved well by the current method, any further coarsening is constructed as in the standard SA.

Before presenting computational results, we consider several possible improvements intended to reduce the necessary number of cycles of the setup and the amount of work required to carry each cycle.

Note 4.8 (improving the quality of existing candidates). Many practical situations, including fourth-order equations and systems of fluid and solid mechanics, require a set of multiple candidates to achieve optimal convergence. In the interest of keeping operator complexity as small as possible, it is imperative that the number of candidates used to produce the final method be controlled. Therefore, ways of improving the quality of each candidate are of interest, to curb the demand for the growth in their number.

When the current V-cycle hierarchy is based on approximating at least two candidates (in other words, the coarse problems feature at least two degrees of freedom per node), this can be easily accomplished as follows.

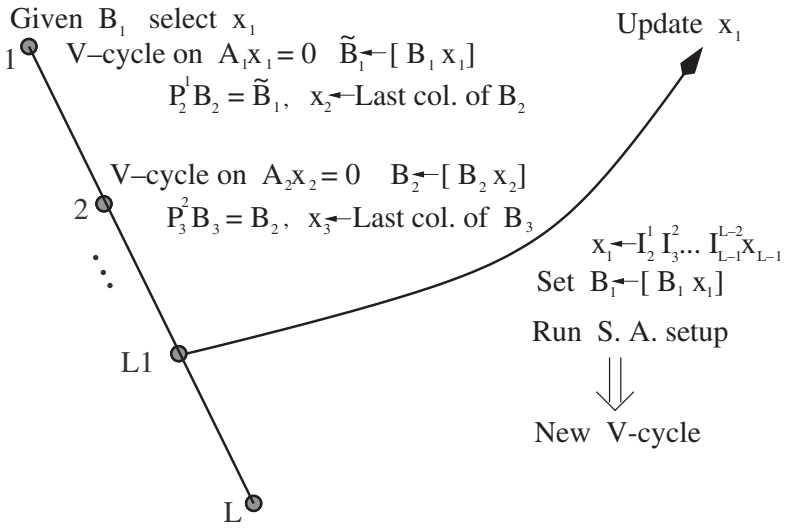

Fig. 4.3 *One step of general setup stage, Algorithm* 4.

Assume that the currently available candidate vectors are $\mathbf{x}_1, \ldots, \mathbf{x}_k$. Consider one such candidate, say, $\mathbf{x_j}$, that we want to improve. We want to run a modified but current V-cycle on the homogeneous problem, $A_1\mathbf{x} = \mathbf{0}$, using $\mathbf{x_j}$ as the initial guess. The modification consists of disabling, in the coarse-grid correction process, the columns of the prolongator corresponding to the given candidate. That is, instead of $\mathbf{x}_l \leftarrow \mathbf{x}_l + I_{l+1}^l \mathbf{x}_{l+1}$ in step 2(c) of Algorithm 1, we use

$$\mathbf{x}_l \leftarrow \mathbf{x}_l + I_{l+1}^l \hat{\mathbf{x}}_{l+1},$$

where $\hat{\mathbf{x}}_{l+1}$ is obtained from $\mathbf{x}_{l+1}$ by setting to zero every entry corresponding to fine-level candidate $\mathbf{x_j}$. Thus, the columns of I_{l+1}^l corresponding to $\mathbf{x_j}$ are not used in coarse-grid correction.

In this way, we come up with an improved candidate vector without restarting the entire setup iteration from scratch and without adding a new candidate. Since we focus on one component at a time and keep all other components intact, this modified V-cycle is expected to converge rapidly.

Note 4.9 (saving work). The reuse of current coarse-level components described in Note 4.5 reduces the amount of work required to keep the V-cycle up-to-date. Additional work can be saved by performing the decomposition of nodes into disjoint aggregates only during the setup of the initial V-cycle and then reusing this decomposition in later cycles. Yet further savings are possible in coarsening, assuming the candidates are allowed to be locally eliminated according to Note 4.4. For instance, we can exploit the second-level matrix structure

$$A_2 = \begin{bmatrix} \tilde{A}_2 & X \\ Y & Z \end{bmatrix},$$

where $\tilde{A}_2$ is the second-level matrix from the previous cycle. Thus, A_2 need not be recomputed and can be obtained by a rank-one update of each block entry in $\tilde{A}_2$. In a similar fashion, the new operators P_{l+1}^l, B_{l+1} do not have to be recomputed in

each new setup cycle by the local QR decomposition noted in section 2. Instead, it is possible to update each nodal entry in $\tilde{P}^l_{l+1}, \hat{B}^{l+1}$ by a rank-one update on all coarse levels, where $\tilde{P}^l_{l+1}, \hat{B}^{l+1}$ are the operators created by the previous setup cycle.

5. Numerical Experiments. To demonstrate the effectiveness of the proposed adaptive setup process, we present results obtained by applying the method to several model problems. In these tests, the solver was stopped when the relative residual reached the value $\epsilon = 10^{-12}$ (unless otherwise specified). $C_a = 10^{-3}$ was used for test (4.6) and parameter ε used in the adaptive setup was 0.1. The relaxation scheme for the multigrid solver was symmetric Gauss–Seidel. While a Krylov subspace process is used often in practice, we present these results for a basic multigrid V-cycle with no acceleration scheme for clarity, unless explicitly specified otherwise.

All the experiments have been run on a notebook computer with a 1.6 GHz mobile Pentium 4 processor and 512 MB of RAM. For each experiment, we report the following. The column denoted by "Iter" contains the number of iterations required to reduce the residual by the prescribed factor. The "Factor" column reports convergence factor measured as the geometric average of the residual reduction in the last 10 iterations. In the "CPU" column, we report the total CPU times in seconds required to complete both the setup and iteration phases of the solver. In the column "RelCPU," we report the relative times to solution, with one unit defined as the time required to solve the problem given the correct near-kernel components. In the "OpComp" column, we report the operator complexity associated with the V-cycle for every run (we define operator complexity in the usual sense [19], as the ratio of the number of entries stored in all problem matrices on all levels divided by the number of entries stored in the finest-level matrix). The "Candidates" column indicates the number of kernel vectors computed in the setup iteration (a value of "provided" means that complete kernel information was supplied to the solver, assuming standard discretization and ignoring scaling). Parameter $\mu_{\max}$ denotes the maximal number of tentative V-cycles allowed in computing each candidate.

In all the cases considered, the problem was modified either by scaling or by rotating each nodal entry in the system by a random angle (as described below). These modifications pose serious difficulties for classical algebraic iterative solvers that are not aware of such modifications.

For comparison, we also include the results for the unmodified problem, with a supplied set of kernel components. Not surprisingly, the standard algorithm (without benefit of the adaptive process) performs poorly for the modified system when the details of this modification are kept from the solver, as we assume here.

Problem 1: Scaled 3D Poisson Problem. We start by considering a diagonally scaled problem,

$$A \leftarrow D^{-1/2} A D^{-1/2},$$

where the original A is the matrix obtained by standard Q1 finite element discretization of the 3D Poisson operator on a cube and D is a diagonal matrix with entries 10^{β}, where $\beta \in [-\sigma, +\sigma]$ is chosen randomly. Table 5.1 shows the results for different values of parameter σ and different levels of refinement. Using the supplied kernel yields good convergence factors for the unmodified problem, but the performance is poor and deteriorates with increased problem size when used with $\sigma \neq 0$. In contrast, the adaptive process, starting from a random approximation, recovers the convergence properties associated with the standard Poisson problem ($\sigma = 0$), even for the scaled case, with convergence that appears bounded independent of the problem size.

Table 5.1 *Misscaled* 3D *Poisson problems with* 68,921 *and* 1,030,301 *degrees of freedom; using* $\epsilon = 10^{-8}$.

σ	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
Poisson problem with 68,921 degrees of freedom							
0	provided	N/A	9	0.100	3.65	1.00	1.038
0	1	5	9	0.100	4.09	1.12	1.038
6	provided	N/A	150	0.871	43.76	11.99	1.038
6	1	5	10	0.126	4.27	1.17	1.038
Poisson problem with 1,030,301 degrees of freedom							
0	provided	N/A	9	0.093	58.43	1.00	1.039
0	1	5	9	0.099	80.05	1.37	1.039
6	provided	N/A	690	0.970	3,252.80	55.67	1.039
6	1	5	9	0.096	88.23	1.51	1.039

Table 5.2 *Scaled* 2D *elasticity problems with* 80,400 *and* 181,202 *degrees of freedom. Iteration counts marked with an asterisk indicate that residual reduction by* 10^{12} *was not achieved before the maximum number of iterations was reached.*

σ	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
2D elasticity problem, 80,400 degrees of freedom							
0	3 provided	N/A	17	0.21	9.16	1.00	1.27
0	3	6	23	0.37	21.16	2.31	1.27
0	3	15	18	0.23	26.65	2.91	1.27
6	3 provided	N/A	299	0.92	133.55	14.58	1.27
6	3	6	25	0.38	22.26	2.43	1.27
6	3	15	18	0.25	27.30	2.98	1.27
2D elasticity problem, 181,202 degrees of freedom							
0	3 provided	N/A	23	0.35	22.85	1.00	1.28
0	3	15	267	0.937	272.14	11.91	1.27
0	4	15	26	0.422	75.18	3.29	1.50
0	4	20	26	0.439	86.60	3.79	1.50
0	5	15	20	0.314	88.20	3.86	1.78
6	3 provided	N/A	5,000*	0.996	4,559.95	199.56	1.28
6	4	15	23	0.367	74.95	3.28	1.50
6	4	20	19	0.302	76.78	3.36	1.50
6	5	10	14	0.173	69.46	3.04	1.78

Problem 2: Scaled 2D Elasticity. Here we consider a diagonally scaled matrix arising in 2D elasticity. Diagonal entries of D are again defined as 10^{β}, with $\beta \in [-\sigma, +\sigma]$ chosen randomly. The original matrix is the discrete operator for the plane-strain elasticity formulation over a square domain using bilinear finite elements on a uniform mesh, with a Poisson ratio of $\nu = 0.3$ and Dirichlet boundary conditions specified only along the "West" side of the domain. The results in Table 5.2 follow a pattern similar to those for the Poisson problem. Note, however, that more than the usual three candidate vectors are now needed to achieve convergence properties similar to those observed with the unmodified problem for which the correct set of three rigid body modes is provided by the user. For the scaled problem, however,

Table 5.3 *Rotated* 2D *elasticity problems with* 80,400 *and* 181,202 *degrees of freedom. Iteration counts marked with an asterisk indicate that residual reduction by* 10^{12} *was not achieved before the limit on the number of iterations was reached.*

Rotated	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
2D elasticity problem with 80,400 degrees of freedom							
NO	3 provided	N/A	17	0.21	9.16 s	1.00	1.27
NO	3	15	18	0.23	26.66	2.91	1.27
YES	3 provided	N/A	1,329	0.99	587.80	64.17	1.27
YES	3	15	19	0.27	27.8464	3.04	1.27
2D elasticity problem with 181,202 degrees of freedom							
NO	3 provided	N/A	23	0.35	22.85 s	1.00	1.28
NO	3	15	18	0.23	66.49	2.91	1.28
YES	3 provided	N/A	$5,000^*$	0.999	3,968.36	173.67	1.28
YES	3	20	135	0.885	170.23	7.45	1.28
YES	4	15	27	0.488	77.46	3.39	1.50
YES	4	20	21	0.395	79.29	3.47	1.50
YES	5	6	18	0.34	60.78	2.66	1.78
YES	5	10	15	0.233	72.66	3.18	1.78

supplying the rigid body modes computed based on the problem geometry leads, as expected, to dismal performance of the standard solver.

Problem 3: Locally Rotated 2D Elasticity. This set of experiments is based again on the 2D elasticity problem, but now each nodal block is rotated by a random angle $\beta \in [0, \pi]$,

$$A \leftarrow Q^T A Q,$$

where Q is a nodal block-diagonal matrix consisting of rotations with random angles. The results in Table 5.3 show that αSA can recover good convergence factors for both the unmodified and the modified systems. Without the adaptive procedure, our basic algebraic solver could not solve the modified matrix problem in a reasonable amount of time.

Problem 4: Locally Rotated 3D Elasticity. This set of experiments demonstrates performance of the method when a higher number of candidates is required. We consider a 3D elasticity problem with local rotations. This is done to maintain locally orthogonal coordinates but is otherwise a random rotation of the three degrees of freedom at each node. The model problem we start from is linearized elasticity discretized using trilinear finite elements over a uniform mesh. Dirichlet boundary conditions are specified on the "West" face of the cube, and the Poisson ratio is set to $\nu = 0.3$. The results in Table 5.4 show that, even for the modified system, the adaptive method can again recover good convergence factors. Furthermore, our current method mimics the convergence of SA for the unmodified problem with the supplied set of rigid body modes. In this set of experiments, we can get close to the ideal iteration counts using just six candidates. We see that using one extra candidate can improve convergence properties and in some cases actually lower the overall cost of the total time to solution. This is done at the price of a small increase in operator complexity. For problems with multiple right sides, the more expensive setup would be performed only once, and using the extra candidate may then be preferred.

Table 5.4 *Rotated* 3D *elasticity problems with* 114,444 *and* 201,720 *degrees of freedom.*

Rotated	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
3D elasticity problem with 114,444 degrees of freedom							
NO	6 provided	N/A	16	0.20	29.97	1.00	1.159
NO	6	15	20	0.27	189.11	6.31	1.159
NO	7	15	17	0.21	215.78	7.20	1.217
YES	6 provided	N/A	587	0.97	913.49	30.48	1.159
YES	6	15	16	0.22	184.32	6.15	1.159
YES	7	10	15	0.20	171.73	5.73	1.217
YES	7	15	15	0.20	210.99	7.04	1.217
3D elasticity problem with 201,720 degrees of freedom							
NO	6 provided	N/A	16	0.20	50.33	1.00	1.153
NO	6	15	21	0.31	319.60	6.35	1.153
NO	7	10	17	0.216	297.95	5.92	1.209
NO	7	15	17	0.209	363.38	7.22	1.209
YES	6 provided	N/A	739	0.97	1,924.62	38.24	1.153
YES	6	15	16	0.23	308.02	6.12	1.153
YES	7	10	15	0.20	301.98	6.00	1.209
YES	7	15	14	0.16	357.85	7.11	1.209

Figures 5.1 and 5.2 illustrate the difference between candidates computed for the locally rotated problem and the original (unrotated) problem, respectively. While the candidate representing the algebraically smooth error corresponding to the original problem exhibits also smoothness in the geometric sense, for the rotated problem such smoothness is observed only at the Dirichlet boundary, where the solution is zero regardless of the rotation. For the sake of visualization, the candidates depicted in Figures 5.1 and 5.2 have been computed by running a smaller problem with 86,490 degrees of freedom.

Problem 5: 3D Elasticity with Discontinuous Coefficients. The final example demonstrates performance of the adaptive method for an elasticity problem featuring discontinuities in the Young modulus. Here we consider a 3D elasticity problem in which the Poisson ratio is fixed at 0.32, while the Young modulus is allowed to vary randomly between the elements. We consider two cases: a case of coefficients varying randomly with uniform distribution in the interval $(1, 10^{\sigma})$ and the case where the distribution is exponential; i.e., the Young modulus is computed as $10^{(\sigma r)}$, where r is generated randomly with uniform distribution in (0, 1). Keeping with the usual practice of employing Krylov method acceleration for problems with coefficient discontinuities, in this experiment we use our adaptive method as a preconditioner in the conjugate gradient method. The iteration was stopped once the initial residual was reduced by a factor of 10^8. Table 5.5 compares the results obtained by using our adaptive scheme, starting from a random initial guess, to the results obtained when the method based on a priori knowledge of the appropriate rigid body modes is employed as a preconditioner. The table indicates that, using the adaptive procedure without a priori knowledge of the problem geometry, we can essentially recover the rates of the method based on the knowledge of the rigid body modes.

The topic of problems with discontinuous coefficients and the appropriate modifications to the basic SA method will be studied in a future paper.

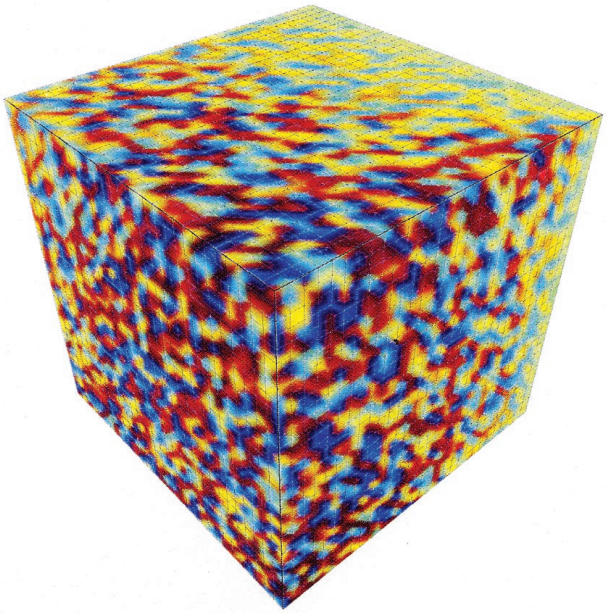

Fig. 5.1 *Candidate representing algebraically smooth error for the* 3D *elasticity problem modified by applying random local rotations.*

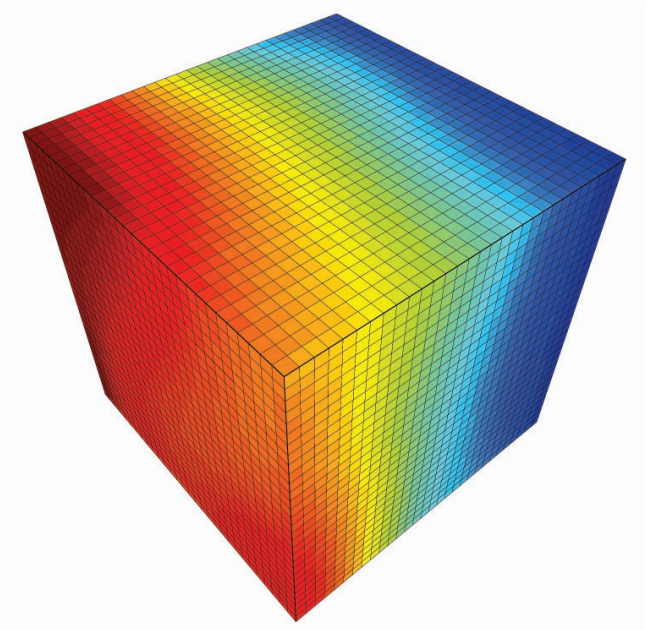

Fig. 5.2 *Candidate computed for the* 3D *elasticity problem without the modification by local rotations exhibits geometric smoothness.*

Table 5.5 3D *elasticity problem,* 201,720 *degrees of freedom, with Young modulus featuring random jumps in* $(1, 10^{\sigma})$.

σ	Candidates	$\mu_{\max}$	Iter	Factor	CPU	RelCPU	OpComp
Elasticity problem with uniformly distributed coefficient jumps							
2	6 provided	N/A	8	0.0734	24.22	1.00	1.15
2	6	10	13	0.2209	219.34	9.05	1.15
2	7	10	11	0.1866	266.29	10.99	1.21
3	6 provided	N/A	8	0.0765	24.23	1.00	1.15
3	6	10	13	0.2154	218.57	9.02	1.15
3	7	10	11	0.1861	265.70	10.96	1.21
4	6 provided	N/A	8	0.0768	24.56	1.00	1.15
4	6	10	12	0.2081	219.26	8.93	1.15
4	7	10	11	0.1648	264.38	10.76	1.21
Elasticity problem with exponentially distributed coefficient jumps							
2	6 provided	N/A	9	0.1146	25.99	1.00	1.15
2	6	10	16	0.3048	225.52	8.68	1.15
2	7	10	12	0.2141	267.72	10.30	1.21
3	6 provided	N/A	14	0.2466	35.68	1.00	1.15
3	6	10	22	0.4179	237.56	6.76	1.15
3	7	10	16	0.3095	275.62	7.84	1.21
4	6 provided	N/A	20	0.3948	49.99	1.00	1.15
4	6	10	30	0.5316	255.43	5.11	1.15
4	7	10	21	0.4040	289.39	5.79	1.21
5	6 provided	N/A	32	0.5545	73.63	1.00	1.15
5	6	10	46	0.6695	292.35	3.97	1.15
5	6	20	36	0.5980	402.75	5.47	1.21
5	7	10	37	0.6020	324.19	4.40	1.21
5	7	15	27	0.4966	381.16	5.17	1.21

Note 5.1. The operator complexities in all of the test problems remain below 2. Moreover, these complexities improve somewhat in three dimensions, compared to two dimensions, due largely to the increased speed of aggregation coarsening. It is also worth mentioning that the increasing size of the coarse matrix block entries due to the increasing number of candidates does not significantly impact the time needed to perform one iteration of the solver, apparently due to the more efficient memory access afforded by blocking.

6. Conclusions. This paper develops a new multilevel method to tackle problems that, to date, have been very difficult to handle by AMG methods. At the expense of a somewhat more costly setup stage and more intricate implementation, we design a method that has, thus far, proved successful at solving such problems. We observe that the convergence properties of the method seem very insensitive to modifications of the algebraic system by scaling or nodal rotation. Moreover, the solver is flexible and can benefit from extra information supplied about the problem. If such information is lacking or incorrect, then αSA can act as a full black-box solver. Despite the growing number of degrees of freedom per coarse-level node as the method evolves, the overall cost of one step of the final iteration grows only modestly because of better utilization of cache memory due to dense matrix operations on the nodal blocks.

Operator complexity remains at reasonable levels and actually seems to improve with increasing spatial dimension.

The construction of the tentative prolongator in the setup phase involves restriction of the candidate functions to an aggregate and subsequent local orthogonalization of these functions. It is therefore suitable for parallel processing as long as the aggregates are local to the processor. Parallelization of the underlying SA solver is in the testing stage. When it is completed, then αSA should also benefit from the parallel speedup. The parallel version is also expected to gain better parallel scalability by replacing the traditionally used Gauss–Seidel relaxation with the polynomial smoothing procedures investigated recently in [1]. The performance of the parallel implementation will depend on the quality of the parallel matrix-vector product.

Future development will concentrate on extending features of the underlying method on which αSA relies and on developing theory beyond the heuristics presented here. Although most decisions are currently made by the code at runtime, much remains to be done to fully automate the procedure, such as determining certain tolerances that are now input by the user. We plan to explore the possibility of setting or updating these parameters at runtime based on the characteristics of the problem at hand. A related work in progress [7] explores adaptive ideas suitable in the context of the standard AMG method.

REFERENCES

[1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, *Parallel multigrid smoothing: Polynomial versus Gauss-Seidel*, J. Comput. Phys., 188 (2003), pp. 593–610.

[2] J. Bramble, J. Pasciak, J. Wang, and J. Xu, *Convergence estimates for multigrid algorithm without regularity assumptions*, Math. Comp., 57 (1991), pp. 23–45.

[3] A. Brandt, *Lecture given at CASC*, Lawrence Livermore National Lab, Livermore, CA, 2001.

[4] A. Brandt, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, UK, 1984, pp. 257–284.

[5] A. Brandt and D. Ron, *Multigrid solvers and multilevel optimization strategies*, in Multilevel Optimization in VLSICAD, Comb. Optim. 14, J. Cong and J. R. Shinnerl, eds., Kluwer Academic, Dordrecht, The Netherlands, 2003, pp. 1–69.

[6] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid based on element interpolation (AMGe)*, SIAM J. Sci. Comput., 22 (2000), pp. 1570–1592.

[7] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. F. McCormick, and J. W. Ruge, *Adaptive Algebraic Multigrid (αAMG)*, SIAM J. Sci. Comput., submitted.

[8] M. Brezina, C. I. Heberton, J. Mandel, and P. Vaněk, *An Iterative Method with Convergence Rate Chosen A Priori*, UCD/CCM report 140, Center for Computational Mathematics, University of Colorado at Denver, Denver, CO, 1999; available online from http://www-math.cudenver.edu/ccmreports/rep140.ps.gz.

[9] M. Brezina and P. Vaněk, *A black-box iterative solver based on a two-level Schwarz method*, Computing, 63 (1999), pp. 233–263.

[10] W. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM, Philadelphia, 2000.

[11] T. Chartier, *Element-Based Algebraic Multigrid (AMGe) and Spectral AMGe*, Ph.D. thesis, University of Colorado at Boulder, Boulder, CO, 2001.

[12] T. Chartier, R. D. Falgout, V. E. Henson, J. Jones, T. Manteuffel, S. McCormick, J. Ruge, and P. S. Vassilevski, *Spectral AMGe (ρAMGe)*, SIAM J. Sci. Comput., 25 (2003), pp. 1–26.

[13] J. Fish and V. Belsky, *Generalized aggregation multilevel solver*, Internat. J. Numer. Methods Engrg., 40 (1997), pp. 4341–4361.

[14] S. F. McCormick and J. W. Ruge, *Algebraic multigrid methods applied to problems in computational structural mechanics*, in State of the Art Surveys on Computational Mechanics, A. K. Noor and J. T. Oden, eds., ASME, New York, 1989, pp. 237–270.

[15] J. W. Ruge, *Algebraic multigrid (AMG) for geodetic survey problems*, in Proceedings of the International Multigrid Conference, Copper Mountain, CO, 1983.
[16] J. W. Ruge, *Final Report on AMG*02, report, Gesellschaft fuer Mathematik und Datenverarbeitung, St. Augustin, 1985, GMD, contract 5110/022090.
[17] J. W. Ruge, *AMG for problems of elasticity*, Appl. Math. Comput., 19 (1986), pp. 293–309.
[18] J. W. Ruge and K. Stüben, *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, in Multigrid Methods for Integral and Differential Equations, The Institute of Mathematics and Its Applications Conference Series, D. J. Paddon and H. Holstein, eds., Clarendon Press, Oxford, UK, 1985, pp. 169–212.
[19] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods, Frontiers Appl. Math. 3, S. F. McCormick, ed., SIAM, Philadelphia, 1987, pp. 73–130.
[20] P. Vaněk, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numer. Math., 88 (2001), pp. 559–579.
[21] P. Vaněk, *Acceleration of convergence of a two-level algorithm by smoothing transfer operator*, Appl. Math., 37 (1992), pp. 265–274.
[22] P. Vaněk, M. Brezina, and R. Tezaur, *Two-grid method for linear elasticity on unstructured meshes*, SIAM J. Sci. Comput., 21 (1999), pp. 900–923.
[23] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.
[24] R. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.