SIAM J. MATRIX ANAL. APPL.
Vol. 21, No. 4, pp. 1236–1252

SPARSE APPROXIMATE INVERSE SMOOTHER FOR MULTIGRID*

WEI-PAI TANG† AND WING LOK WAN‡

Abstract. Various forms of sparse approximate inverses (SAI) have been shown to be useful for preconditioning. Their potential usefulness in a parallel environment has motivated much interest in recent years. However, the capability of an approximate inverse in eliminating the local error has not yet been fully exploited in multigrid algorithms. A careful examination of the iteration matrices of these approximate inverses indicates their superiority in smoothing the high-frequency error in addition to their inherent parallelism. We propose a new class of SAI smoothers in this paper and present their analytic smoothing factors for constant coefficient PDEs. The following are several distinctive features that make this technique special:

- By adjusting the quality of the approximate inverse, the smoothing factor can be improved accordingly. For hard problems, this is useful.
- In contrast to the ordering sensitivity of other smoothing techniques, this technique is ordering independent.
- In general, the sequential performance of many superior parallel algorithms is not very competitive. This technique is useful in both parallel and sequential computations.

Our theoretical and numerical results have demonstrated the effectiveness of this new technique.

Key words. sparse approximate inverse, smoother, multigrid

AMS subject classifications. 15A09, 15A23, 65F10, 65F50, 65Y05

PII. S0895479899339342

1. Introduction.

The effectiveness of multigrid is based on two main components: *smoothing* and *coarse grid correction*. The smoothing process, usually carried out by a relaxation method, dampens the high-frequency error components. The coarse grid correction process, carried out by an interpolation, approximates the low-frequency components on the coarser grids. The idea is that the combination of the two will result in a significant error reduction independent of the problem size [22] and hence lead to an efficient solution procedure.

Relaxation methods such as Richardson, Jacobi, and Gauss–Seidel are often used as smoothers for multigrid, although other iterative methods, for instance, incomplete LU factorization (ILU) [23, 41] and (preconditioned) conjugate gradient [1, 7] methods, or even ODE solvers [26] such as Runge–Kutta methods, have also been used for specific problems. Relaxation methods are particularly useful for multigrid since they are simple, and they all have the common property of removing the high-frequency error components [5, 22].

*Received by the editors January 18, 1999; accepted for publication (in revised form) by G. Golub October 6, 1999; published electronically April 18, 2000.

http://www.siam.org/journals/simax/21-4/33934.html

†Department of Computer Science, University of Waterloo, Waterloo, ON, Canada N2L 3G1 (wptang@bryce1.math.uwaterloo.ca). This author was supported by the Natural Sciences and Engineering Research Council of Canada, by the Information Technology Research Centre, which is funded by the Province of Ontario, and by NASA under contract NAS 2-96027 between NASA and the Universities Space Research Association (USRA).

‡SCCM Program, Gates Building 2B, Stanford University, Stanford, CA 94305-9025 (wan@sccm.stanford.edu). Former address: Department of Mathematics, University of California, Los Angeles, CA 90095-1555. This author was partially supported by the NSF under contract ASC-9720257, NASA Ames under contract NAG2-1238, Sandia National Laboratory under contract LG-4440, Lawrence Livermore National Laboratory under contract W07405-Eng-48, Department of Energy under the Accelerated Strategic Computing Initiative (ASCI), and the Alfred P. Sloan Foundation as a Doctoral Dissertation Fellow.

In practice, Gauss–Seidel is usually the most effective smoother among relaxation methods. A drawback, however, is that Gauss–Seidel is a very sequential algorithm. A parallel version may be obtained by a special ordering of the unknowns, for example, red-black ordering for the five-point stencil operator on a square grid. The Jacobi method, on the other hand, is a very parallel method, but its smoothing efficiency is not as good as that of the Gauss–Seidel method. It is also well known that Gauss–Seidel smoothers do not work well for anisotropic problems and discontinuous coefficient problems. In this paper, we propose a new class of smoothers derived from sparse approximate inverse (SAI) preconditioners. It has a smoothing efficiency similar to Gauss–Seidel and it is independent of ordering. Moreover, for hard problems, we can improve the smoothing efficiency by adaptively adjusting the quality of the approximate inverse, for instance, by adding more nonzeros. Consequently, this new technique is more robust than using Gauss–Seidel smoothers. Our numerical tests verify this statement.

We remark that Huckle [25] independently experimented with an SAI smoother for multigrid by modifying the standard Gauss–Seidel iteration. Specifically, instead of using the exact inverse of the lower triangular matrix, he used an SAI computed by the techniques described in [20]. In our approach, we do not restrict ourselves only to Gauss–Seidel. Indeed, we replace the Gauss–Seidel smoother entirely by an SAI smoother. The resulting multigrid is efficient, and we have more flexibility in improving the smoothing quality for hard problems.

In section 2, we describe the construction of the SAI smoother. In section 5, we analyze the smoothing property of the proposed smoother both analytically and numerically for constant coefficient PDEs. Comparison to the Gauss–Seidel smoother is also given. Some techniques to improve the smoothing character for anisotropic problems are presented. Finally, in section 6, we show the effectiveness of the SAI as a smoother for multigrid on a variety of problems, including anisotropic problems, discontinuous coefficient problems, and unstructured grid problems.

2. SAI smoothers. Various techniques have been proposed for an effective SAI preconditioner [2, 4, 9, 11, 18, 20, 29, 30, 37]. However, the goal of constructing an effective smoother is very different from finding a good preconditioner. For a powerful preconditioner, the capability for removing both the high- and low-frequency errors is essential. In contrast, a good smoother may just dampen the high-frequency errors effectively. In this respect, much of the weaknesses [37] of SAI preconditioners become the strengths for SAI smoothers. Our proposal is to explore this to construct a powerful smoother.

Most SAI approaches seek a sparse matrix M so that the error of the residual

$$\mathcal{E} = MA - I$$

is minimized in some measure. The sparsity constraint often limits the effectiveness of M as a preconditioner due to the locality of the sparse approximation. This lack of global approximation has created many difficulties for a powerful preconditioner. Various additional techniques are required to improve the quality of an SAI preconditioner [8, 37]. The requirement for a good smoother, however, can take advantage of the locality of the sparse approximation. In the following sections, we propose a systematic approach for constructing sparsity patterns and a modified least squares formulation for SAI smoothers. The cost of this simpler form of *approximate inverse–local least squares approximation* is much less expensive than other typical SAI approaches, and yet it possesses a very effective smoothing property which we will show

in section 5.

2.1. Modified Frobenius norm construction. The construction of our SAI smoothers is based on the Frobenius norm approach described by Benson [2] and Benson and Frederickson [3]. Consider the left preconditioned linear system

$$MAy = Mb,$$

where M is a left preconditioner. An SAI preconditioner M is defined as a solution to the following minimization problem:

$$\min_M \|MA - I\|_F^2$$

subject to some constraints on the number and position of the nonzero entries of M. The minimization problem is equivalent to n independent least squares problems which can be solved in parallel:

$$\min_{m_j} \|A^T m_j - e_j\|_2, \qquad j = 1, \ldots, n, \tag{2.1}$$

where m_j is the jth row of M and e_j is the jth column of I.

In practice, the size of the matrix A may be too large for an effective solution of (2.1). However, since we are looking for a sparse solution M, the number of nonzeros of m_j is small and can be bounded. Thus, the matrix A in (2.1) can be replaced by a submatrix of itself, A_j, whose columns correspond to the nonzero entries of m_j. Let the compressed dense vector of m_j be m'_j, which contains only the nonzeros of m_j. The least squares problem (2.1) becomes

$$\min_{m_j} \|A_j^T m'_j - e_j\|_2, \qquad j = 1, \ldots, n. \tag{2.2}$$

Our modified approach simplifies (2.2) further by selecting a subset of nonzero rows of A_j. In the following, we describe a systematic procedure of constructing submatrices of A for defining the m_j's. In contrast to many other SAI approaches, this modified Frobenius norm approach has direct control of the number and location of the nonzeros of the m_j's as well as the complexity of the least squares problems a priori.

2.2. Algorithm. The submatrix construction algorithm is described by using graph theory notation. A sparse matrix A can be represented by a digraph $\mathcal{G} = (\mathcal{O}, \mathcal{E})$ [14]. For multigrid methods, the graph often is the mesh of the PDE solution domain. Define $\mathcal{L}_k(o_i)$, the k-level neighbor set of node o_i, the nodes of which are a distance $k+1$ or less from o_i. The use of the level concept to define the sparsity pattern for incomplete factorizations is widely present in ILU preconditioning [21, 12]. The 0-level neighbor set $\mathcal{L}_0(o_i)$ contains all the nodes which directly connect to node o_i. For PDE problems with a second-order finite difference/finite element discretization, $\mathcal{L}_0(o_i)$ is just the set of stencil points. Similarly, define $\mathcal{W}_k(o_i)$, the kth wavefront of node o_i, as the set of nodes which are a distance $k+1$ from node o_i.

Remark. It is known that the inverse of a matrix arising from a PDE problem is an approximation of the Green's function for that PDE. A row of this inverse for the grid node o_i would be an approximation of the Green's function when one variable is fixed at the given location. For many two-dimensional elliptic PDEs, this row can be portrayed as a decay surface with a singularity at node o_i. This leads to the wavefront decay behavior where the wavefront center is the corresponding node o_i. Let A_{ij}^{-1} be

the element of the inverse A^{-1} at location (i, j). The size of A_{ij}^{-1} is then associated with the unit contribution from node o_j to node o_i. With the wavefront decay, the set $\mathcal{L}_k(o_i)$ includes the k most influential wavefronts for o_i. This motivates us to choose $\mathcal{L}_k(o_i)$ for the sparsity pattern to approximate the discrete Green's function at node o_i. The computation of these locations is also inexpensive. An effective choice of the sparsity pattern can have a significant impact on the quality and the cost of the SAI.

In section 2.1, we mentioned that further saving is required to yield an effective approximate inverse. The least squares problems (2.2) have to be replaced by some much smaller tasks. Different algorithms for constructing an approximate inverse lead to different approaches for choosing the sparsity pattern of m_j and a submatrix of A_j in (2.2). After the submatrix $\widetilde{A}_j$ is chosen, (2.2) is replaced by a smaller least squares problem,

$$\min_{m_j} \|\widetilde{A_j^T} m'_j - e_j\|_2.$$

In constructing an effective SAI smoother, two important issues arise:

- The sparsity pattern of m_j. Based on the wavefront decay observation, we will choose $\mathcal{L}_k(o_j)$. A larger k is related to the higher cost for constructing the SAI smoother. For a regular problem, $k = 0$ is often good enough.
- What kind of local error should be removed effectively with this smoother? If we choose the submatrix of A_j^T corresponding to the rows for the nearest neighbors $\mathcal{L}_l(o_j)$, the resulting smoother will remove the error in this neighborhood more effectively, since the errors of this particular SAI on these locations are optimized by the least squares problem. For anisotropic problems, the error is not distributed evenly. This neighborhood measuring by a simple distance is not physically correct, and a more sophisticated technique should be introduced (see section 4). It is clear that a higher level l provides an approximation which is good for more neighbors. Notice that l is usually larger than k. Our experience indicates that $k = l$ is not a good choice.

Now, we define the rectangular submatrix[1]

$$A_{k,l} \equiv A(\mathcal{L}_k(o_j), \mathcal{L}_l(o_j)) \tag{2.3}$$

as the (k, l)-level local matrix of node o_j. This matrix takes rows corresponding to nodes $\mathcal{L}_k(o_j)$ and columns corresponding to nodes $\mathcal{L}_l(o_j)$ from A.

We introduce the (k, l)-*level local least squares approximation* of an inverse as follows. For each node o_j (row) of the approximation, the least squares solution m_{o_j} of

$$A_{k,l}^T m'_{o_j} = e_{o_j} \tag{2.4}$$

is taken for the nonzero values at the corresponding location $\mathcal{L}_k(o_j)$ of the sparse approximation of the discrete Green's function corresponding to node o_j. Here, e_{o_j} is a unit basis vector of size $|\mathcal{L}_l(o_j)|$, with one in the position corresponding to o_j and zeros elsewhere. In other words, we inject each element of the least squares solution m'_{o_j} back into a zero vector m_{o_j} at the corresponding locations in $\mathcal{L}_k(o_j)$ and use this sparse row to approximate the row o_j of A^{-1}.

[1]The MATLAB notation is adopted for extracting a submatrix from a given matrix A.

Remark. The first SAI approach proposed by Benson [2] is precisely the $(0,0)$-level local least squares approximation. However, it was not a good choice for either preconditioner or smoother.

The relaxation method

$$x^{\mu+1} = x^{\mu} + M(b - Ax^{\mu})$$

resulting from the SAI M constructed by the (k,l)-level local least squares approximation is called the (k,l)-level SAI smoothing.

2.3. Example: Constant coefficient PDEs. We illustrate the (0,1)-level local least squares approximation method by a model constant coefficient second-order elliptic PDE:

$$w_1 u_{xx} + w_2 u_{yy} + s u_x + t u_y = f(x,y), \qquad (x,y) \in \Omega,$$
$$u|_{\Gamma} = g(x,y)$$

on a square domain Ω. Suppose we use a $p \times q$ regular grid. The discretization matrix A is penta-diagonal and of size $(p \times q)^2$, using the conventional 5-point stencil finite difference method. Moreover, the resulting difference equation can be generally written as

$$b u_{i-1,j} + d u_{i,j-1} + a u_{i,j} + c u_{i+1,j} + e u_{i,j+1} = h^2 f_{i,j}, \qquad 1 \le i \le p, \quad 1 \le j \le q,$$

where a, b, c, d, and e are constants. Here we present the picture of the stencil and the construction of a $(0,1)$-level submatrix corresponding to the center node on a 5×5 grid in Figure 2.1. The vertical (horizontal) dashed lines indicate the columns (rows) to be extracted from the original matrix to form the $(0,1)$-level submatrix. The arrows indicate the position of the center node. The circulated nonzeros on the cross points of the dashed lines will appear in the submatrix. In Figure 2.1, the $(0,1)$-level local least squares problem for the center node o_j is

$$\begin{pmatrix}
a & & & & c \\
 & a & & & d \\
 & & a & & b \\
 & & & a & e \\
b & e & c & d & a \\
c & & & & \\
d & c & & & \\
 & d & & & \\
 & b & d & & \\
 & & b & & \\
 & & e & b & \\
 & & & e & \\
e & & & c &
\end{pmatrix}
\begin{pmatrix} x_W \\ x_N \\ x_E \\ x_S \\ x_C \end{pmatrix}
=
\begin{pmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix}, \tag{2.5}$$

which yields

$$(m_{o_j})_i = \begin{cases} x_W & \text{if } i = o_j - 1, \\ x_N & \text{if } i = o_j + 5, \\ x_E & \text{if } i = o_j + 1, \\ x_S & \text{if } i = o_j - 5, \\ x_C & \text{if } i = o_j, \\ 0 & \text{otherwise.} \end{cases}$$

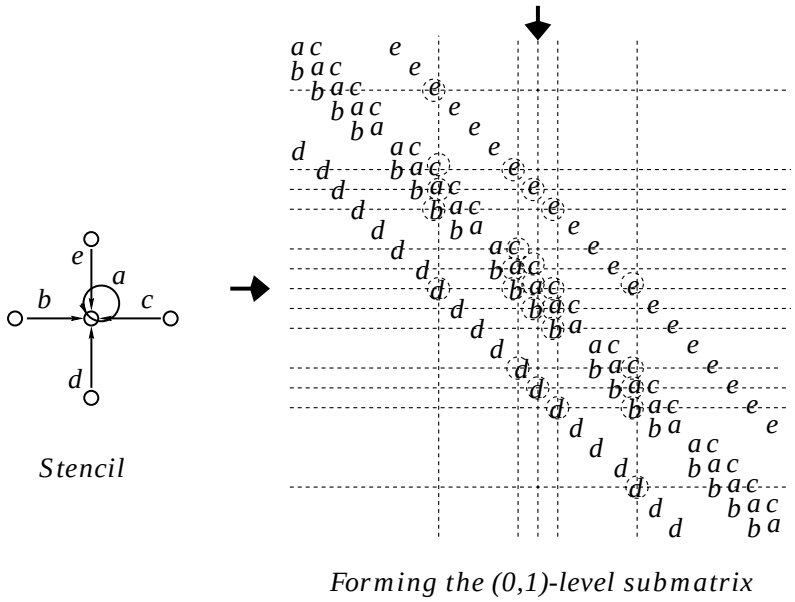


FIG. 2.1. *A stencil and the construction of a submatrix.*

2.4. Simplified SAI smoothers. Computing these local least squares solutions is an easy task and can be implemented in parallel effectively. For any constant coefficient PDE on a regular mesh, we may further simplify the computations as follows. We observe that the local least squares problems, and hence the least square solutions, corresponding to the interior points are identical (cf. (2.5)). However, they may be different near the boundary. We simply use the least squares solutions obtained from the interior points for the solutions at the boundary. Hence, we need to solve only one small least squares problem to obtain the entire approximate inverse. Our tests indicate that this simplified approach does not bring any noticeable penalty in performance for regular PDEs.

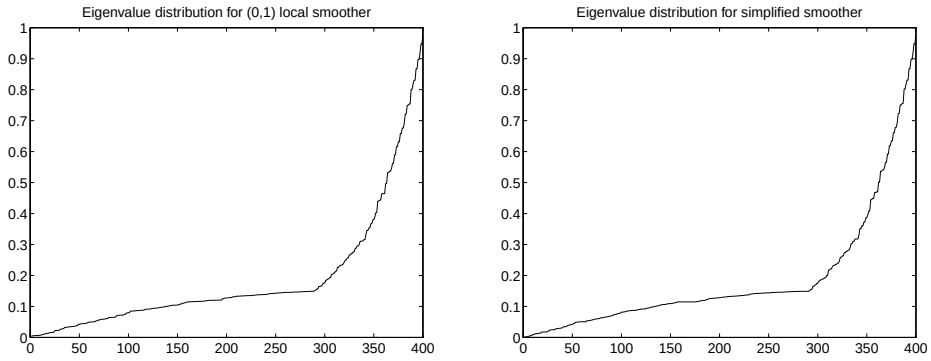
As an illustration, consider solving the following Poisson equation on a 20×20 rectangular grid:

$$\Delta u(x, y) = 1, \qquad (x, y) \in \Omega.$$

In Figure 2.2, we present the plots of the eigenvalues of the iteration matrices associated with the (0,1)-level SAI smoother and its simplified version. The eigenvalues in Figure 2.2(a) were calculated by MATLAB using the *eig* function and those in Figure 2.2(b) by the analytic formula (5.2) derived in section 5. There are no visible differences.

Finally, we would like to mention a number of other techniques which can also be adopted to save the cost in computing the SAI. A follow-up paper will discuss these in lengths. For example,

- the shape of the Green's function for different positions inside the solution region does not vary much for a PDE with constant coefficients; when an unstructured mesh is used, we may compute one approximation of the discrete Green's function accurately and use interpolation to obtain the approximation for all other mesh points;



(a) $(0,1)$-level SAI smoother (b) Simplified smoother

FIG. 2.2. *Comparison of the eigenvalue distributions.*

- for smooth variable coefficient PDEs, the use of one local least squares solution for several of its neighbors is feasible;
- for anisotropic problems, a higher level SAI smoother is required; however, the cost of the local least-squares problems grows rapidly; an a priori drop tolerance technique [37] can significantly reduce the computations.

3. Computation complexity. The use of SAI smoothers consists of two phases: the setup phase for constructing the sparse approximate inverse by solving the n independent least squares problems (2.2) and the application phase, which applies the SAI matrix to a vector. We analyze the complexity of each phase in the following sections.

3.1. Setup phase. The computational cost involved depends on the size of the least squares matrix $A_{k,l}$, which in turn depends on the size of $|\mathcal{L}_k(o_j)|$ and the number of levels k used. In general, it is hard to estimate $|\mathcal{L}_k|$ for a general sparse matrix A. Thus, we restrict our discussion to matrices arising from 5-point stencil discretizations on rectangular grids. First, we have the formula for $|\mathcal{L}_k|$.

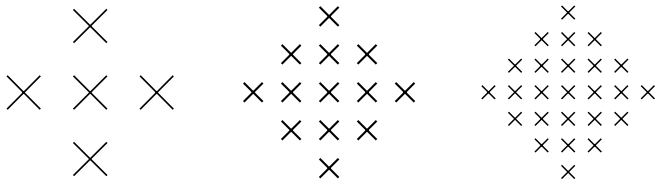
THEOREM 3.1. *For nodes o_j away from the boundary, the number of the first k level nodes is*

$$|\mathcal{L}_k(o_j)| = 2k^2 + 6k + 5.$$

Proof. The formula can be easily derived by examining the first few levels of neighboring nodes as shown in Figure 3.1. First, we observe that there are $2k+3$ rows for the level k pattern. Second, the number of nodes in the central row increases by 2 for each increasing level, and hence is equal to $2k + 3$. The number of nodes in the upper $k + 1$ rows starts from 1 and increases by 2 for each row below, and hence it has a total of $(k + 1)^2$ nodes. Thus, altogether, we have

$$|\mathcal{L}_k(o_j)| = 2(k + 1)^2 + (2k + 3) = 2k^2 + 6k + 5. \qquad \square$$

Although $|\mathcal{L}_k|$ grows quadratically with respect to k, we remark that our numerical experiments in section 6 show that small values of k are sufficient to give rise to an effective SAI smoother. In fact, $k = 1$ is good enough for smooth coefficient PDEs.



(a) level=0 (b) level=1 (c) level=2

FIG. 3.1. *Different level of stencil nodes.*

The following is a well-known result [16] for estimating the complexity of the QR factorization method for solving least squares problems. For simplicity, we assume that $|\mathcal{L}_k(o_j)| \equiv |\mathcal{L}_k|$ is constant for all o_j.

THEOREM 3.2. *The complexity of solving* (2.4) *is*

$$2|\mathcal{L}_l||\mathcal{L}_k|^2 - \frac{2}{3}|\mathcal{L}_k|^3.$$

For general purpose, (0,1)-level SAI smoother is sufficient. In that case, $|\mathcal{L}_0| = 1$ and $|\mathcal{L}_1| = 5$. Thus we need about 10 flops to solve each least squares problem (2.4), and hence the total cost is about 2 Gauss–Seidel iterations.

There are several ways to reduce the cost of the setup phase. For instance, we may employ the simplified version of SAI smoothers described in section 2.4 for constant coefficient PDEs. We need to solve only one least squares problem to obtain the entire approximate inverse, and its cost is clearly negligible.

In many applications where we may have to solve the linear system $Ax^{(i)} = b^{(i)}$ repeatedly with different right-hand sides, the setup cost becomes less significant. Moreover, in a parallel computation environment, the setup phase can be implemented easily in parallel.

3.2. Application phase. Applying an SAI to a vector is simply a matrix-vector multiplication. Thus, its complexity depends on the number of elements per row, which is determined by $|\mathcal{L}_k|$.

THEOREM 3.3. *The complexity of applying the SAI smoother is*

$$n(2k^2 + 6k + 5).$$

Remark. For the recommended (0,1)-level SAI smoother, the complexity is the same as one step of Gauss–Seidel iteration. In a parallel computation environment, the application of the SAI smoother can be done in parallel easily, whereas Gauss–Seidel is a very sequential process.

4. Adaptivity and anisotropic problems. In addition to simple construction and easy parallel implementation, SAI smoothers offer the capability to improve their quality by adjusting the number of levels (i.e., the sparsity of the approximate inverse) used. Relaxation methods, for instance, Gauss–Seidel, do not have such a feature, which means users have to look for other smoothers whenever relaxation smoothers fail to give fast multigrid convergence. For SAI smoothers, we may tune a parameter—the number of levels—to improve the resulting multigrid convergence; in the extreme case when the number of levels equals the matrix size, the SAI is the exact inverse, yielding

one step multigrid convergence. Thus, SAI smoothers are more robust, flexible and can be applied to a larger class of problems.

The potential disadvantage of using higher level SAI smoothers is the higher cost involved. However, like ILU(k) methods, the issue is not just cost per iteration but also whether the resulting multigrid iteration converges. After all, a costly convergent method is still better than inexpensive nonconvergent methods. Anisotropic problems usually have many relatively small entries in both the matrix A and its inverse. We may significantly reduce the cost by discarding these small entries to obtain a sparser approximate inverse. The resulting higher level SAI smoothers can be shown to be inexpensive and yet very effective for anisotropic problems (section 6).

We remark that there are two typical techniques for multigrid to solve anisotropic problems. One technique, similar to ours, is to use a better smoother, for instance, block relaxation methods. Another approach is to use semicoarsening, a special coarsening technique. Both techniques are usually used for regular grids only and are most effective if the direction of isotropy is in either the x or y direction. Special techniques are needed to handle variable anisotropic problems. Our SAI smoothers, on the other hand, are algebraic and do not require any information about the direction of isotropy. Exactly the same procedure can be applied to variable anisotropic problems as well as to irregular grids. We also mention that there exist counterparts of block relaxation smoothers and semicoarsening for irregular grids. They are generally more complicated in that they require heuristic algorithms to determine the lines of different isotropy. Finally, we remark that our SAI smoothers are not meant to be the ultimate method for solving constant coefficient anisotropic problems on regular grids where standard techniques work well. Rather, they are proposed as robust smoothers which are simple to use, applicable to both regular and irregular grids, effective for both simple and tough (e.g., anisotropic) PDE problems, and easily parallelizable.

5. Smoothing factor analysis. We present an analysis of the smoothing factors for constant coefficient PDEs on two-dimensional rectangular grids (equispaced) in this section. For simplicity, we consider only the simplified (0,1)-level SAI smoothers, which have been shown in section 2.4 to be only a small perturbation of the original (0,1)-level SAI smoothers.

The smoothing factor analysis is based on the following theorem.

THEOREM 5.1. *Given two tridiagonal matrices*

$$B = \text{tridiag}(b, a', c)_{p\times p}, \qquad C = \text{tridiag}(d, a'', e)_{q\times q},$$

where b, c, d, e are nonnegative. Then the eigenvalues of the matrix

$$B \otimes I_{q\times q} + I_{p\times p} \otimes C \tag{5.1}$$

are

$$a + 2\sqrt{bc}\cos\theta_k + 2\sqrt{de}\cos\theta_j, \qquad 1 \le k \le p \;\; 1 \le j \le q, \tag{5.2}$$

where $a = a' + a''$, $\theta_k = \frac{k\pi}{p+1}$, $\theta_j = \frac{j\pi}{q+1}$. The corresponding eigenvectors are

$$\left\{ \left(\frac{c}{b}\right)^{\frac{n-s}{2}} \left(\frac{e}{d}\right)^{\frac{n-t}{2}} \sin(s\theta_k)\sin(t\theta_j) \right\}, \qquad 1 \le s \le p,\ 1 \le t \le q.$$

Proof. This result can be shown by using the well-known analysis for the eigenvalues and eigenvectors of a tridiagonal matrix; for example, see [42, 34]. ∎

Unlike Fourier analysis for PDEs with periodic boundary condition, Theorem 5.1 shows that the sine functions $\{\sin\frac{ks\pi}{m+1}\sin\frac{jt\pi}{n+1}\}$ are not the eigenvectors for general constant coefficient PDEs with Dirichlet boundary condition; the eigenvectors may depend on the coefficients a, b, c, d, and e. To give further insight into Theorem 5.1, we continue the analysis further with the assumption that A is symmetric; i.e., $b = c$ and $d = e$.

Denote by M the simplified SAI of the matrix A. Then M can also be written in the form of (5.1).

THEOREM 5.2. *The eigenvalues of the iteration matrix $I - MA$ are*

$$1 - ((x_C + 2x_W\cos\theta_k + 2x_N\cos\theta_j)\ (a + 2b\cos\theta_k + 2d\cos\theta_j))\,,$$

where $1 \le k \le p, \quad 1 \le j \le q$.

Proof. Consider the left-hand side of (2.5). By the symmetry of A, if we switch x_W and x_E, we will get exactly the same coefficient least squares matrix. Thus $x_E = x_W$. Similarly, we also have $x_N = x_S$. Hence M is also symmetric. By Theorem 5.1, both A and M have the same set of eigenvectors and so they can be diagonalized at the same time. The eigenvalues of the iteration matrix are then given by Theorem 5.1. ∎

Remark. This analysis can also be generalized to three-dimensional problems.

5.1. Local mode analysis. We carry out the classical local mode analysis [5] to analyze the smoothing efficiency of the (0,1)-level SAI smoothers for Poisson equations. By a direct computation and Theorem 5.1, we have the following two lemmas.

LEMMA 5.3. *Suppose A is the standard Laplacian operator. Then the least squares solution of* (2.5) *is*

$$x_W = x_N = x_E = x_S = \frac{3}{61}, \qquad x_C = \frac{17}{61}.$$

LEMMA 5.4. *Let M be the $(0,1)$-level sparse approximate inverse of the standard Laplacian operator. Then the eigenvalues of M are*

$$-\frac{17}{61} - \frac{6}{61}\cos\theta_k - \frac{6}{61}\cos\theta_j.$$

Remark. Lemma 5.4 shows that $M < 0$, which implies M is invertible.

As a consequence of Theorem 5.2, we have the following result.

LEMMA 5.5. *The eigenvalues of the iteration matrix of the $(0,1)$-level smoother are*

$$\lambda(\theta_k,\theta_j) = 1 - \left(\frac{17}{61} + \frac{6}{61}\cos\theta_k + \frac{6}{61}\cos\theta_j\right)(4 - 2\cos\theta_k - 2\cos\theta_j). \tag{5.3}$$

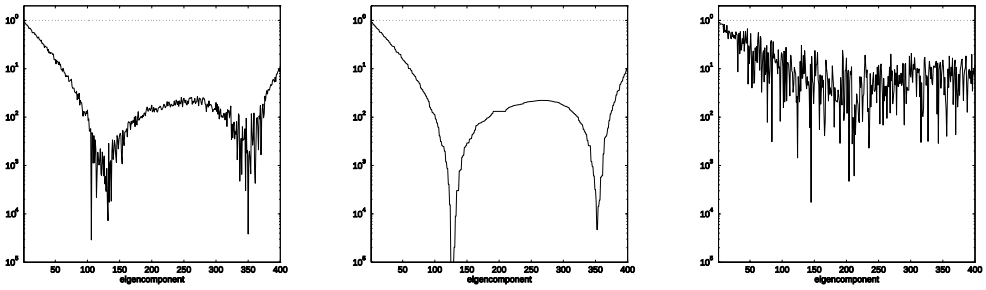
Now, we show the smoothing factor of our SAI smoother. Define the high frequencies:

$$\Theta_H = \left\{(\theta_k,\theta_j) : \frac{p}{2} \le k \le p, \ \ \frac{q}{2} \le j \le q\right\}.$$

We have the following smoothing factor result.

THEOREM 5.6. *Let ρ be the smoothing factor defined as*

$$\rho \equiv \max\{|\lambda(\theta_k,\theta_j)| : (\theta_k,\theta_j) \in \Theta_H\},$$



(a) (0,1)-level smoother (b) Simplified smoother (c) GS smoother

FIG. 5.1. *Errors after two smoothing steps of different smoothers. The dotted line denotes the initial error.*

where $\lambda(\theta_k, \theta_j)$ *is as in Lemma* 5.5. *Then* $\rho \leq \frac{21}{61} \approx \frac{1}{3}$.

Proof. Write $c_k = \cos\theta_k$ and $c_j = \cos\theta_j$. For $(\theta_k, \theta_j) \in \Theta_H$, we have

$$-1 < c_k, c_j < 0. \tag{5.4}$$

Expand the formula in (5.3) and obtain

$$\lambda(\theta_k, \theta_j) = \frac{12}{61}(c_k + c_j)^2 + \frac{10}{61}(c_k + c_j) - \frac{7}{61}.$$

Let $x = c_k + c_j$. By (5.4), $-2 < x < 0$. By a simple calculation, the polynomial $p(x) = \frac{12}{61}x^2 + \frac{10}{61} - \frac{7}{61}$ has a maximum in absolute value at $x = -2$ for $-2 \leq x \leq 0$. Thus

$$\rho \leq \max\{|p(x)| : -2 \leq x \leq 0\} = |p(-2)| = \frac{21}{61}. \qquad \square$$

Thus, the SAI smoothers reduce the high frequencies by a factor of almost 1/3, and hence they are effective smoothers for smooth coefficient PDEs. As a comparison, similar calculations can be carried out for Gauss–Seidel with natural and red-black ordering [22, 40]. The smoothing factors are 1/2 and 1/4, respectively. In section 6, we show that the SAI smoothers are also effective for tough PDEs such as anisotropic problems.

5.2. Spectral analysis. In addition to the above Fourier analysis, we compare the smoothing efficiency of our SAI smoothers and the Gauss–Seidel smoother using a spectral analysis. Figure 5.1 shows the errors after two smoothing steps of different smoothers. The x-axis represents the eigencomponents with respect to the Laplacian matrix, and the y-axis represents their magnitudes. For Laplacian matrices, the small and large eigenvalues correspond to the low and high frequencies, respectively. From the plots, there is no significant difference between the (0,1)-level local smoother and the simplified smoother. Moreover, they both dampen high frequencies more effectively than the Gauss–Seidel smoother.

6. Numerical results. In this section, we demonstrate the effectiveness of the proposed SAI smoothers. Example 1 illustrates that our SAI smoothers perform essentially the same as GS smoothers, if the latter work. It shows that the additional easy

parallel implementation feature of the SAI smoothers do not affect convergence. Besides, for some problems, SAI smoothers may converge while Gauss–Seidel smoothers do not.

In addition to the gain of parallel efficiency, our SAI smoothers have the capability of improving themselves by adjusting a parameter. When relaxation smoothers do not work, we can do nothing to make them work. For SAI smoothers, we may increase the number of levels, which in turn increases the density of the sparse approximate inverse, improving their quality for hard problems. We illustrate this point by solving anisotropic coefficient PDE problems in Example 2. We show that the cost of higher level SAI smoothers is only about twice as expensive per iteration, whereas the reduction in the number of iterations is much more than double. Moreover, the SAI smoothers also work well for unstructured grid problems where the standard multigrid techniques for anisotropic problems on regular grids cannot be applied. Thus our SAI smoothers are more robust.

In all the examples, Dirichlet boundary conditions are used, and the right-hand-side function is $f(x) = 1$. In the multigrid procedure, a V-cycle is used with two presmoothing and two postsmoothing steps. Linear interpolation is used for structured grid problems, and a specialized energy-minimizing interpolation [38, 39] is used for unstructured grid problems. The number of multigrid levels is such that the coarsest grid is 3×3 for structured square grid problems and a total of four levels for unstructured grid problems. The iteration was terminated when the relative residual norm was less than 10^{-8}. We are using zeros as the initial guess in all cases. Actually, we may report better numbers of iterations if a random initial guess is used. The results are summarized in table form, where the number of V-cycles and the average convergence rate of the last 10 iterations are shown.

Example 1. We compare the performance of two Gauss–Seidel smoothers with the (0,1)-level SAI smoother and its simplified version, the simplified (0,1)-level SAI smoother by several smooth and rough coefficient PDEs on a 33×33 square grid. The equation for the variable coefficient problem is

$$((1 + x^2)u_x)_x + u_{yy} + \tan^2 y u_y = -100x^2.$$

The equation for the helical spring problem [15] is

$$u_{xx} + u_{yy} + \frac{3}{5 - y}u_x - 2G\lambda = 0,$$

where G and λ are some constants. The discontinuous coefficient PDE is

$$(a(x, y)u_x)_x + (b(x, y)u_y)_y + u_x + u_y = \sin(\pi xy),$$

where the coefficients $a(x, y)$ and $b(x, y)$ are defined as

$$a(x, y) = b(x, y) = \begin{cases} 10^{-3} & (x, y) \in [0, 0.5] \times [0.5, 1], \\ 10^{3} & (x, y) \in [0.5, 1] \times [0, 0.5], \\ 1 & \text{otherwise.} \end{cases}$$

Table 6.1 shows the convergence results. For smooth coefficient problems, all the methods have similar convergence behavior. Since the (0,1)-level is used, the cost per iteration is also the same for all the methods. Besides, we observe that the simplified (0,1)-level SAI performs essentially the same as its original version, while its setup cost involves only one least squares solve.

TABLE 6.1

Smooth and rough coefficient PDEs on a 33×33 *mesh. GS, Gauss–Seidel with natural ordering; GS(rb), Gauss–Seidel with red-black ordering; SAI,* $(0,1)$*-level SAI smoother; SSAI, simplified* $(0,1)$*-level SAI smoother.* ∞ *indicates divergence.*

Problems	Iteration				Convergence rate			
	GS	GS(rb)	SAI	SSAI	GS	GS(rb)	SAI	SSAI
Poisson	9	7	9	9	0.11	0.07	0.11	0.11
Variable coeff.	13	10	12	17	0.22	0.15	0.19	0.35
Helical spring	12	9	12	12	0.20	0.12	0.19	0.20
Discont. coeff.	∞	∞	22	∞	∞	∞	0.40	∞

For the discontinuous coefficient problem, the two Gauss–Seidel smoothers diverge while our (0,1)-level SAI smoother converges. The failure of the simplified SAI smoother is expected since the Green's function of a single point in the interior may not be a good approximation to those with a region of totally different coefficients.

Example 2. We show how we may improve the quality of SAI by using a higher level SAI when (0,1)-level SAI does not work well. We consider two model anisotropic coefficient PDEs.

Problem 1: The single direction anisotropic problem is

$$\begin{aligned} 100u_{xx} + u_{yy} &= 1, \qquad (x,y) \in \Omega, \\ u|_\Gamma &= 0. \end{aligned}$$

Problem 2: This has a more sophisticated anisotropy structure in both the x and y directions:

$$a(x,y)u_{xx} + b(x,y)u_{yy} = 1,$$

where the coefficients $a(x,y)$ and $b(x,y)$ are defined as

$$a(x,y) = \begin{cases} 100 & (x,y) \in [0,0.5] \times [0,0.5] \text{ or } [0.5,1] \times [0.5,1], \\ 1 & \text{otherwise}, \end{cases}$$

$$b(x,y) = \begin{cases} 100 & (x,y) \in [0,0.5] \times [0.5,1] \text{ or } [0.5,1] \times [0,0.5], \\ 1 & \text{otherwise}. \end{cases}$$

The results are shown in Tables 6.2 and 6.3. The first three rows show the convergence results for Problem 1 on 32×32, 64×64, and 128×128 grids, respectively. Similarly, rows 4 to 6 show the results for Problem 2. The last two rows show the results for Problem 1 on two unstructured grids shown in Figure 6.1.

As is well known, multigrid with Gauss–Seidel smoother is very slow for these kinds of problems. Similar results are also obtained for Gauss–Seidel with red-black ordering, and hence they are omitted. For Problem 1 on regular grids, a standard technique[2] for improving the multigrid convergence is to use line (block) relaxation methods. As indicated in the table, the block Jacobi smoother is quite effective for small grids but eventually slows down for bigger grids. Moreover, the block Jacobi smoother for Problem 1 does not work for Problem 2 since the latter has different isotropies in both the x and y directions. We have to manually change the two line relaxation smoothing in the x direction into one line relaxation in the x direction and

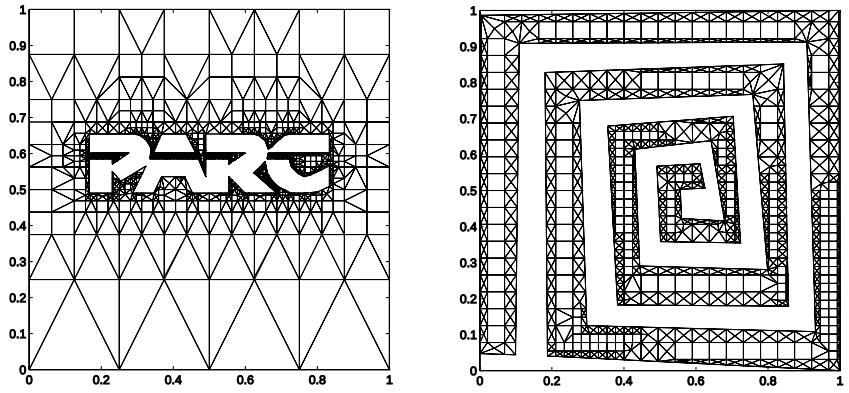
[2]Another technique is to use *semicoarsening*. Since this approach uses a different coarsening rather than an improved smoother, we do not compare this method with our SAI smoother.

TABLE 6.2
Convergence results for anisotropic problems on different grids. GS, Gauss–Seidel with natural ordering; BJ, Block Jacobi. See text for the definitions of SAI$(3,\epsilon)$ *and SAI*$(4,\epsilon)$. $\epsilon = 0.0008$ *for the regular grid problems and* 0.0004 *for the unstructured grid problems.*

Grids	Iteration			
	GS	BJ	SAI(3,ϵ)	SAI(4,ϵ)
Problem 1 (32×32)	> 100	9	25	18
Problem 1 (64×64)	> 100	31	33	24
Problem 1 (128×128)	> 100	> 100	37	27
Problem 2 (32×32)	81	16	15	12
Problem 2 (64×64)	> 100	29	28	22
Problem 2 (128×128)	> 100	67	39	32
Parc	92	–	27	22
Spiral	74	–	18	14

TABLE 6.3
Convergence rates for anisotropic problems on different grids. GS, Gauss–Seidel with natural ordering; BJ, Block Jacobi. See text for the definitions of SAI$(3,\epsilon)$ *and SAI*$(4,\epsilon)$. $\epsilon = 0.0008$ *for the regular grid problems and* 0.0004 *for the unstructured grid problems.*

Grids	Convergence rate			
	GS	BJ	SAI(3,ϵ)	SAI(4,ϵ)
Problem 1 (32×32)	0.89	0.13	0.52	0.40
Problem 1 (64×64)	0.91	0.62	0.61	0.50
Problem 1 (128×128)	0.92	0.88	0.64	0.53
Problem 2 (32×32)	0.82	0.34	0.32	0.23
Problem 2 (64×64)	0.89	0.57	0.55	0.47
Problem 2 (128×128)	0.91	0.66	0.60	0.54
Parc	0.86	–	0.55	0.47
Spiral	0.79	–	0.37	0.27



(a) parc (b) spiral

FIG. 6.1. *The unstructured grids for Example* 2.

one line relaxation in the y direction. The results are still shown under the column BJ. For unstructured grid problems, blocks defined along the direction of the anisotropy are hard, if not impossible, to determine. Thus we do not test the block Jacobi smoother in this case. On the whole, the convergence of the block Jacobi smoother is

TABLE 6.4

Cost of applying different smoothers relative to Gauss–Seidel. Under each smoother, the first column shows the cost per iteration relative to one GS iteration, and the second column shows the total cost = cost per iteration × total number of iterations.

Grids	GS		BJ		SAI(3,ϵ)		SAI(4,ϵ)	
Problem 1 (32×32)	1	> 100	2	18	1.50	38	1.57	28
Problem 1 (64×64)	1	> 100	2	62	1.65	54	1.68	40
Problem 1 (128×128)	1	> 100	2	> 100	1.72	64	1.74	47
Problem 2 (32×32)	1	81	2	32	1.64	25	2.02	24
Problem 2 (64×64)	1	> 100	2	58	1.72	48	1.92	42
Problem 2 (128×128)	1	> 100	2	> 100	1.76	69	1.86	60
Parc	1	92	–	–	1.58	43	1.90	42
Spiral	1	74	–	–	1.19	21	1.30	18

quite reasonable; but it is essentially a regular grid technique.

The previous (0,1)-level SAI smoother is not very effective in this case. We improve the performance by using higher level SAI smoothers (section 2). For higher level SAI, however, the approximate inverse is much denser. We control the amount of fill-in by dropping small elements. SAI(k, ϵ) denotes the $(k, k+1)$-level SAI smoother where the elements in the approximate inverse whose absolute values are below ϵ are dropped. We remark that we may also drop small elements in the matrix A before we compute the approximate inverse. Our experience indicates that the difference between the two dropping strategies is small and hence we consider only the former approach.

Tables 6.2 and 6.3 show that the higher level SAI smoothers perform essentially the same as the block Jacobi smoother for small grids and better for larger grids. More importantly, in contrast to line relaxation smoothers, the construction of the SAI smoothers does not require any information about the lines of different isotropy. Hence the exact same construction procedure can be applied to both Problems 1 and 2. The higher level SAI allows us to capture the anisotropy easily by adjusting only one number, the level of fill-in, with no need to track geometrically the anisotropic directions, which are often hard to determine. Furthermore, it does not matter whether it is a regular grid or irregular grid problem. For the two unstructured grid problems, the SAI smoothers converge much faster than the Gauss–Seidel smoother. We remark that the apparent increase in the number of multigrid iterations with respect to the mesh size is probably due to the fact that the multigrid preconditioner used in each grid size problem is slightly different since the entries dropped in constructing the SAI smoothers are different. Mesh size independent convergence should be observed if no dropping strategy is used.

Table 6.4 shows the costs of different smoothers relative to Gauss–Seidel. Under each smoother, the first column shows the cost per iteration of the smoother relative to one Gauss–Seidel iteration; the second column shows the total cost = cost per iteration × total number of iterations. The cost of each smoother is estimated as follows. The cost of one Gauss–Seidel iteration is estimated as the number of nonzeros in the matrix A. Thus, the relative cost of the SAI smoothers is estimated as the ratio of the number of nonzeros in the sparse approximate inverse to the number of nonzeros in the matrix A. The cost of inverting the diagonal block is about $8n$, assuming each block is a symmetric positive definite tridiagonal matrix [16]. Thus the cost of the block Jacobi smoother is about twice that of Gauss–Seidel.

From the table, it is clear that the higher cost of higher level SAI smoothers re-

sults in a lower total cost due to the much better improvement in the total number of iterations. Hence it is worth the effort to use higher level SAI smoothers for difficult problems such as anisotropic coefficient PDEs.

We note that we may adjust/improve the SAI smoothers locally according to the local behavior of the PDE coefficients. For instance, we may use a higher level SAI approximation on the subdomains with anisotropic PDE coefficients and a simple SAI approximation on the subdomains with isotropic coefficients. This local adaptivity feature is another advantage over methods based on incomplete factorizations. However, for Example 2, the coefficient is anisotropic throughout the entire domain, and higher level SAI approximation is needed everywhere.

Acknowledgments. Both authors would like to thank Tony Chan for his comments and discussions on the development of the sparse approximate inverse smoothers, and Li-Tien Cheng for his careful proofreading. The second author would also like to acknowledge the support and hospitality of the Computer Science Department of the University of Waterloo; this work was initiated during a visit there.

REFERENCES

[1] R. E. Bank and C. C. Douglas, *Sharp estimates for multigrid rates of convergence with general smoothing and acceleration*, SIAM J. Numer. Anal., 22 (1985), pp. 617–633.

[2] M. W. Benson, *Iterative Solution of Large Scale Linear Systems*, M. Sc. Thesis, Lakehead University, Thunder Bay, ON, Canada, 1973.

[3] M. W. Benson and P. O. Frederickson, *Iterative solution of large scale linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–120.

[4] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149.

[5] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

[6] R. Bridson and W.-P. Tang, *Ordering, anisotropy, and factored sparse approximate inverses*, SIAM J. Sci. Comput., 21 (1999), pp. 867–882.

[7] R. H. Chan, T. F. Chan, and W. L. Wan, *Multigrid for Differential-Convolution Problems Arising from Image Processing*, CAM Report 97-20, UCLA, Los Angeles, CA, 1997.

[8] T. Chan, W. -P. Tang, and W. L. Wan, *Wavelet sparse approximate inverse preconditioners,* BIT, 37 (1997), pp. 644–650.

[9] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., 18 (1997), pp. 1657–1675.

[10] S. S. Clift and W.-P. Tang, *Weighted graph based ordering techniques for preconditioned conjugate gradient methods*, BIT, 35 (1995), pp. 30–47.

[11] J. Cosgrove, J. Diaz, and A. Griewank, *Approximate inverse preconditionings for sparse linear systems*, Internat. J. Comput. Math., 44 (1992), pp. 91–110.

[12] D. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, *Towards a cost effective ILU preconditioner with high level fill*, BIT, 32 (1992), pp. 442–463.

[13] G. E. Forsythe and W. R. Wasow, *Finite Difference Methods for Partial Differential Equations,* John Wiley, New York, 1960.

[14] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[15] L. Collatz, *The Numerical Treatment of Differential Equations*, 3rd ed., Springer-Verlag, New York, 1966.

[16] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1989.

[17] N. I. M. Gould and J. A. Scott, *On Approximate-Inverse Preconditioners*, Tech. Report RAL-TR-95-026, Computing and Information System Department, Atlas Center, Rutherford Appleton Lab., Oxfordshire UK, June, 1995.

[18] M. Grote and H. Simon, *Parallel preconditioning and approximate inverse on the connection machine*, in Proceedings of the Scalable High Performance Computing Conference (SHPCC), Williamsburg, VA, IEEE Computer Science Press, Los Alamitos, CA, 1992, pp. 76–83.

[19] M. Grote and H. D. Simon, *Parallel preconditioning and approximate inverses on the connection machine*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing II, R. F. Sincovec, D. E. Keyey, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., SIAM, Philadelphia, 1993, pp. 519–523.

[20] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[21] I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.

[22] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[23] P. W. Hemker, *The incomplete LU-decomposition as a relaxation method in multi-grid algorithms*, in Boundary and Interior Layers—Computational and Asymptotic Methods, J. H. Miller, ed., Boole Press, Dublin, 1980, pp. 306–311.

[24] T. Huckle and M. Grote, *A New Approach to Parallel Preconditioning with Sparse Approximate Inverses*, SCCM Report, Stanford University, Stanford, CA, 1994.

[25] T. Huckle, *Sparse approximate inverses and multigrid methods*, in Proceedings of the Sixth SIAM Conference on Applied Linear Algebra, Snowbird, UT, 1997.

[26] A. Jameson, *Solution of the Euler equation for two-dimensional flow by a multigrid method*, Appl. Math. Comput., 13 (1983), pp. 327–355.

[27] I. E. Kaporin, *New convergence results and preconditioning strategies for the conjugate gradient method*, Numer. Linear Algebra Appl., 1 (1994), pp. 179–210.

[28] L. Y. Kolotilina , *Explicit preconditioning of systems of linear algebraic equations with dense matrices*, J. Soviet Math., 43 (1988), pp. 2566–2573.

[29] L. Kolotilina, A. Nikishin, and A. Yeremin, *Factorized sparse approximate inverse (FSAI) preconditionings for solving* 3*D FE systems on massively parallel computers.* II, in Proceedings of the IMACS International Symposium, Iterative Methods in Linear Algebra, Brussels, R. Beauwens and P. Groen, eds., North-Holland, Amsterdam, 1992, pp. 311–312.

[30] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[31] J. Lifshitz, A. Nikishin, and A. Yeremin, *Sparse approximate inverse (FSAI) preconditionings for solving* 3*D CFD problems on massively parallel computers*, in Proceedings of the IMACS International Symposium, Iterative Methods in Linear Algebra, Brussels, R. Beauwens and P. Groen, eds., North-Holland, Amsterdam, 1992, pp. 83–84.

[32] G. Meurant, *A review on the inverse of symmetric tridiagonal and block tridiagonal matrices*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 707–728.

[33] J. Rice and R. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.

[34] U. Schumann and R. A. Sweet, *Fast Fourier transforms for direct solution of Poisson's equation with staggered boundary conditions*, J. Comput. Phys., 75 (1988), pp. 123–137.

[35] W.-P. Tang, *Schwarz Splitting and Template Operators*, Ph.D. thesis, Stanford University, Stanford, CA, 1987.

[36] W.-P. Tang, *Template Operators and Exponential Decay,* Tech. Report CS-90-46, University of Waterloo, ON, Canada, 1990.

[37] W.-P. Tang, *Toward an effective sparse approximate inverse preconditioner,* SIAM J. Matrix Anal. Appl., 20 (1999), pp. 970–986.

[38] W. L. Wan, *An Energy-Minimizing Interpolation for Multigrid*, CAM Report 97-18, Department of Mathematics, UCLA, Los Angeles, CA, 1997.

[39] W. L. Wan, T. F. Chan, and B. Smith, *An Energy-Minimizing Interpolation for Robust Multigrid*, CAM Report 98-6, UCLA, Los Angeles, CA, 1998.

[40] P. Wesseling, *An Introduction to Multigrid Methods*, John Wiley, New York, 1992.

[41] G. Wittum, *On the robustness of ILU smoothing,* SIAM J. Sci. Comput., 10 (1989), pp. 699–717.

[42] D. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.