Thirteenth International Conference on Domain Decomposition Methods
Editors: N. Debit, M.Garbey, R. Hoppe, J. Périaux, D. Keyes, Y. Kuznetsov

2 Multigraph Algorithms Based on Sparse Gaussian Elimination

R. E. Bank[1], R. K. Smith[2]

Introduction

In this work, we describe a multilevel-multigraph algorithm. An excellent recent survey on algebraic approaches to multilevel iterative methods is given in Wagner [Wag99]. This article also contains an extensive bibliography. The algorithm discussed here is described more fully in [BS00]. Our goal is to develop an iterative solver with the simplicity of use and robustness of general sparse Gaussian elimination, and at the same time to capture the computational efficiency of classical multigrid algorithms. While we do not believe that the current algorithm achieves this goal, it represents an important step in this direction. To guarantee robustness, general sparse Gaussian elimination with minimum degree ordering is a point in the parameter space of our method. This is a well known and widely used method, among the most computationally efficient of general sparse direct methods [GL81].

To obtain simplicity of use and implementation, our algorithms incorporate many technologies and algorithms originally developed for general sparse Gaussian elimination. Besides the minimum degree algorithm, the Reverse Cuthill-McKee ordering is the basis of our coarsening procedure. Our sparse matrix data structures are a generalization of those first introduced in the symmetric Yale Sparse Matrix Package [EGSS82], and our (incomplete) factorization procedure is a generalization of the sparse row elimination scheme used there. To gain computational efficiency, our method offers the possibility to compute an incomplete factorization with the user able to specify a drop tolerance and an absolute bound on the total fill-in. This factorization becomes the smoother in a multilevel procedure similar to the classical multigrid method.

Sparse direct methods typically have two phases. In the *initialization* phase, equations are ordered, and symbolic and numerical factorizations are computed. In the *solution* phase, the solution of the linear system is computed using the factorization. Our procedure, as well as other algebraic multilevel methods, also breaks naturally into two phases. The initialization consists of ordering, incomplete symbolic and numeric factorizations, and the computation of the transfer matrices between levels. In the solution phase, the preconditioner computed in the initialization phase is used to compute the solution using the preconditioned Composite Step Conjugate Gradient (CSCG) or the Composite Step Biconjugate Gradient (CSBCG) method [BC93].

In the spirit of general sparse Gaussian elimination, we have tried to minimize the number of user specified control parameters. In the initialization phase, there are three parameters. The most important is the drop tolerance ($dtol$) for the incomplete factorization. Because the fill-in for the ILU tends to be a very nonlinear and unpredictable function of the drop tolerance, we also allow the user to specify an upper bound on the amount to fill-in the be allowed in the

[1]University of California at San Diego, La Jolla CA 92093, rbank@ucsd.edu. The work of this author was supported by the National Science Foundation under contract DMS-9706090.
[2]Agere Systems, Murray Hill, NJ 07974, kentsmith@agere.com.

incomplete factorization ($maxfil$). Finally, the maximum number of levels in the multilevel procedure ($maxlvl$) can be specified. In the solution phase, the user can specify only two control parameters: the maximum number of iterations ($maxcg$) and an error tolerance (tol) for the convergence criterion.

Our main interest is in developing a solver for discretizations of scalar elliptic problems as in the finite element code *PLTMG* [Ban98]. However, our solver was developed as a stand-alone linear equations solver, and can formally be applied to any structurally symmetric, non-singular, sparse matrix. By structurally symmetric, we mean that the pattern of nonzeros in the matrix is symmetric, although the numerical values of the matrix elements may render it nonsymmetric. Many problems arising in practice naturally have structural symmetry, and of course all can be *made* structurally symmetric by storing some extra zeroes. For certain problems handled by *PLTMG*, the matrices are symmetric and positive definite, but for others, the linear systems are highly nonsymmetric and/or indefinite. Thus in practice, this represents a very broad class of behavior.

Structural symmetry allows for some important simplifications in the implementation. In particular, we can handle linear systems involving symmetric matrices A and nonsymmetric matrices A and A^t within a single, unified code, rather than developing specialized subroutines for each of these three cases. In the nonsymmetric case, linear systems involving A^t arise naturally in the context of the CSBCG algorithm, and hence are important for our solver. This limits the complexity of the code, and also eliminates additional parameters that might be needed to further classify a given matrix. On the other hand, it seems clear that a specialized solver directed at a specific problem or class of problems, and making use of additional knowledge, is likely to outperform our algorithm on that particular class of problems. Although we do not think our method is provably "best" for any particular problem, we believe its generality and robustness, coupled with reasonable computational efficiency, make it an interesting and useful approach for solving linear systems.

Matrix Formulation

Let A be a large sparse, nonsingular $N \times N$ matrix. We assume that the sparsity pattern of A is symmetric, although the numerical values need not be. We consider the solution of the linear system

$$Ax = b. \tag{1}$$

Let B be an $N \times N$ nonsingular smoothing matrix. In our case, B is an approximate factorization of A, i.e.,

$$B = (L + D)D^{-1}(D + U) \approx P^t AP, \tag{2}$$

where L is (strict) lower triangular, U is (strict) upper triangular with the same sparsity pattern as L^t, D is diagonal, and P is a permutation matrix.

Given an initial guess x_0, m steps of the smoothing procedure produce iterates x_k, $1 \le k \le m$, given by

$$\begin{aligned} r_{k-1} &= P^t(b - Ax_{k-1}), \\ B\delta_{k-1} &= r_{k-1}, \\ x_k &= x_{k-1} + P^t\delta_{k-1}. \end{aligned} \tag{3}$$

The second component of the two-level preconditioner is the *coarse grid correction*. Here we assume that the matrix A can be partitioned as

$$\hat{P}A\hat{P}^t = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix}, \tag{4}$$

where the subscripts f and c denote *fine* and *coarse*, respectively. Similar to the smoother, the partition of A in fine and coarse blocks involves a permutation matrix $\hat{P}$. The $\hat{N} \times \hat{N}$ coarse grid matrix $\hat{A}$ is given by

$$\begin{aligned} \hat{A} &= \begin{pmatrix} V_{cf} & I_{cc} \end{pmatrix} \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} \begin{pmatrix} W_{fc} \\ I_{cc} \end{pmatrix} \\ &= V_{cf}A_{ff}W_{fc} + V_{cf}A_{fc} + A_{cf}W_{fc} + A_{cc}. \end{aligned} \tag{5}$$

The matrices V_{cf} and W_{fc}^t are $\hat{N} \times N$ matrices, with identical sparsity patterns; thus $\hat{A}$ has a symmetric sparsity pattern. If $A^t = A$, we require $V_{cf} = W_{fc}^t$, so $\hat{A}^t = \hat{A}$.

Let

$$\hat{V} = \begin{pmatrix} V_{cf} & I_{cc} \end{pmatrix} \hat{P}, \qquad \hat{W} = \hat{P}^t \begin{pmatrix} W_{fc} \\ I_{cc} \end{pmatrix}. \tag{6}$$

In standard multigrid terminology, the matrices $\hat{V}$ and $\hat{W}$ are called *restriction* and *prolongation*, respectively. Given an approximate solution x_m to (1), the coarse grid correction produces an iterate x_{m+1} as follows:

$$\begin{aligned} \hat{r} &= \hat{V}(b - Ax_m), \\ \hat{A}\hat{\delta} &= \hat{r}, \\ x_{m+1} &= x_m + \hat{W}\hat{\delta}. \end{aligned} \tag{7}$$

In typical multilevel fashion, the linear system $\hat{A}\hat{\delta} = \hat{r}$ in (7) is solved by recursion, in our case a multilevel V-cycle. One the coarsest level, we apply the iteration (3). A single cycle takes an initial guess x_0 to a final guess x_3 as follows: x_1 is defined using (3), x_2 is defined using (7), and x_3 is defined using (3). Note in particular that we use only one pre-smoothing and one post-smoothing iteration.

Some Implementation Details

To complete the definition of the method, we must provide algorithms to:

- Compute the incomplete factorization matrix B in (2).
- Compute the permutation matrix P in (2).
- Compute the fine-coarse partitioning matrix $\hat{P}$ in (4).
- Compute the sparsity patterns and numerical values in the prolongation and restriction matrices in (6).

ILU Factorization

Our incomplete $(L+D)D^{-1}(D+U)$ factorization is similar to the row elimination scheme developed for the symmetric YSMP codes [EGSS82, GL81]. Without loss of generality, assume that the permutation matrix $P = I$, so that $A = (L+D)D^{-1}(D+U) + E$, where E is the error matrix.

After k steps of elimination, we have the block factorization

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} D_{11}+L_{11} & 0 \\ L_{21} & I \end{pmatrix} \begin{pmatrix} D_{11}^{-1} & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} D_{11}+U_{11} & U_{12} \\ 0 & I \end{pmatrix} + \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & 0 \end{pmatrix},$$

where A_{11} is $k \times k$ and A_{22} is $N-k \times N-k$. We assume that at this stage the blocks D_{11}, L_{11}, L_{21}, U_{11}, and U_{12} have been computed.

Our goal for step $k+1$ is to compute the first row and column of the approximate Schur complement S, given by

$$\begin{aligned} \ell &= Se_1 = A_{22}e_1 - L_{21}(D_{11}^{-1}U_{12}e_1), \\ u &= S^t e_1 = A_{22}^t e_1 - U_{12}^t(D_{11}^{-1}L_{21}^t e_1). \end{aligned}$$

This is done by a procedure similar to the row elimination scheme employed by the symmetric YSMP codes. After the complete (sparse) vectors ℓ and u are computed, certain entries are dropped (assigned to the error matrix E). In particular, we neglect a pair of off-diagonal elements if

$$\max |L_{ij}|, |U_{ji}| \le dtol\sqrt{|D_{jj}A_{ii}|}, \tag{8}$$

where $j = k+1$; D_{ii} has not yet been computed. The drop tolerance $dtol$ is applied in a symmetric fashion to maintain a symmetric sparsity pattern in the factorization.

It is well known that the fill-in generated through the application of a criterion such as (8) is a highly nonlinear and matrix dependent function of $dtol$. This is especially problematic in the present context, since control of the fill-in is necessary in order to control the work per iteration in the multilevel iteration. Thus, in addition to the drop tolerance $dtol$, the user sets the parameter $maxfil$, which specifies that the total number of nonzeros in U is not larger than $maxfil \cdot N$.

Our basic strategy is to compute the incomplete decomposition using the given drop tolerance. If it fails to meet the given storage bound, we increase the drop tolerance and begin a new incomplete factorization. We continue in this fashion until we complete a factorization within the given storage bound. Of course, such repeated factorizations are computationally expensive, so we develop heuristics which allow us to predict a drop tolerance which will satisfy the storage bound. Thus, should the original factorization fail to satisfy the storage bound, usually only one additional ILU factorization is needed. This is discussed in detail in [BS00].

Finally, we note that there is no comprehensive theory regarding the stability of incomplete triangular decompositions. For certain classes of matrices (e.g., M-matrices), the existence of certain incomplete factorizations has been established; however, in the general case, with potentially indefinite and/or highly nonsymmetric matrices, one must contend in a practical way with the possibility of failure or near failure of the factorization. In our implementation, a failure is revealed by some diagonal entries in D becoming close to zero. Off-diagonal elements L_{ji} and U_{ij} are multiplied by D_{ii}^{-1}, and the solution of $(L+D)D^{-1}(D+U)x = b$ also

involves multiplication by D_{ii}^{-1}. For purposes of calculating the factorization and solution, the value of D_{ii}^{-1} is modified near zero as follows:

$$D_{ii}^{-1} = \begin{cases} 1/D_{ii} & \text{for } |D_{ii}| > \alpha \\ D_{ii}/\alpha^2 & \text{for } |D_{ii}| \leq \alpha \end{cases} .$$

Here α is a small constant; in our implementation, $\alpha = \mu\|A\|$, where μ is the machine epsilon. Although many failures could render the preconditioner well-defined but essentially useless, in practice we have noted that D_{ii}^{-1} is rarely modified for a the large class of finite element matrices which are the main target of our procedure.

Ordering

The minimum degree ordering is used to compute the permutation matrix P in (2). Intuitively, if one is computing an incomplete factorization, an ordering which tends to minimize the fill-in in a complete factorization should tend to minimize the error E in the incomplete factorization. For particular classes of matrices, specialized ordering schemes have been developed; for example, for matrices arising from convection dominated problems, ordering along the flow direction has been used with great success. However, in this general setting, we prefer to use just one strategy for all matrices, to reduce the complexity of the implementation, and to avoid the issue of deciding among various ordering possibilities. We remark that for convection dominated problems, minimum degree orderings perform comparably well to the specialized ones, provided some (modest) fill-in is allowed in the incomplete factorization. For us, this seems to be a reasonable compromise.

Our minimum degree ordering is a standard implementation. We have implemented two small enhancements to the minimum degree ordering; as a practical matter, both involve changes to the input graph data structure that is provided to the minimum degree code. First, we have implemented a drop tolerance similar to that used in the factorization. In particular, the edge in the graph corresponding to off-diagonal entries A_{ij} and A_{ji} is not included in the input data structure if

$$\max |A_{ij}|, |A_{ji}| \leq dtol\sqrt{|A_{jj}A_{ii}|}.$$

This excludes many entries which are likely to be dropped in the subsequent incomplete factorization.

The second modification involves some modest *a priori* diagonal pivoting designed to minimize the number failures (near zero diagonal elements) in the subsequent factorization. This procedure is described in detail in [BS00].

Fine-Coarse Partitioning

Our coarsening scheme is based upon another well-known sparse matrix ordering technique, the Reverse Cuthill-McKee algorithm. This ordering tends to yield reordered matrices with minimal bandwidth, and is widely used with generalized band elimination algorithms [GL81]. Our coarsening procedure is just a simple post-processing step of the basic ordering routine, in which the N vertices of graph are marked as *COARSE* or *FINE*. Initially, all vertices are *UNMARKED*. We proceed through the vertices in RCM order; each *UNMARKED* vertex we

encounter is relabeled *COARSE*, and all of its neighbors are labeled *FINE*. This implicitly defines the matrix $\hat{P}$ given in (4).

Under this procedure, all coarse vertices are surrounded by fine vertices. This implies that the matrix A_{cc} in (4) is a diagonal matrix. For the sparsity patterns of matrices arising from discretizations of scalar partial differential equations as in *PLTMG*, the number of coarse unknowns $\hat{N}$ is typically on the order of $N/4$ to $N/5$.

Computing the Transfer Matrices

We now define the matrices V_{cf} and W_{fc}^t of (5). To define the sparsity structure, we take all the connections of each coarse grid vertex to its fine grid neighbors; that is, the sparsity structures of V_{cf} and W_{fc}^t are the same as the block A_{cf}.

We chose numerical values for V_{cf} and W_{fc} according to the formulae

$$W_{fc} = -R_{ff} D_{ff}^{-1} A_{fc},$$
$$V_{cf} = -A_{cf} D_{ff}^{-1} \tilde{R}_{ff}.$$

Here D_{ff} is a diagonal matrix with diagonal entries equal to those of A_{ff}. In this sense, the nonzero entries in V_{cf} and W_{fc} are chosen as multipliers in Gaussian Elimination. The nonnegative diagonal matrices R_{ff} and $\tilde{R}_{ff}$ are chosen such that nonzero rows of W_{fc} and columns of V_{cf}, respectively, have unit norms in ℓ_1.

Finally, if necessary, the coarsened matrix $\hat{A}$ of (5) is "sparsified" using the drop tolerance and a criterion like (8) to remove small off-diagonal elements. Empirically, applying a drop tolerance to $\hat{A}$ at the end of the coarsening procedure has proved more efficient, and more effective, than trying to independently sparsify its constituent matrices.

Numerical Illustrations

In this section, we present a few numerical illustrations. The problems are all of the form $\mathcal{L}_i u = 1$ in $\Omega = (0,1) \times (0,1)$ with $u = 0$ on $\partial\Omega$. The operators $\mathcal{L}_i$, $1 \le i \le 7$, are given by

$$\begin{aligned}
\mathcal{L}_1 u &= -\Delta u, \\
\mathcal{L}_2 u &= -\Delta u - 1000\, u_x, \\
\mathcal{L}_3 u &= -\Delta u - 1000\, u_x - 1000\, u_y, \\
\mathcal{L}_4 u &= -\Delta u - 1000\, u, \\
\mathcal{L}_5 u &= -\Delta u + 1000\, u, \\
\mathcal{L}_6 u &= -.001 u_{xx} - u_{yy}, \\
\mathcal{L}_7 u &= -\Delta u - 1000\{(y - .5)\, u_x - (x - .5)\, u_y\}.
\end{aligned}$$

These problems are standard PDE's chosen to reflect a wide variety of behavior. We solved these problems on $n \times n$ uniform meshes with $n = 51, 101, 201$; the resulting linear systems are of order $N = n^2$. Uniform meshes were used for standardization, although these problems could be more effectively solved in *PLTMG* using adaptive meshes. A 5×5 mesh, as well as the solutions to the seven problems are shown in Figure 1. Continuous piecewise linear

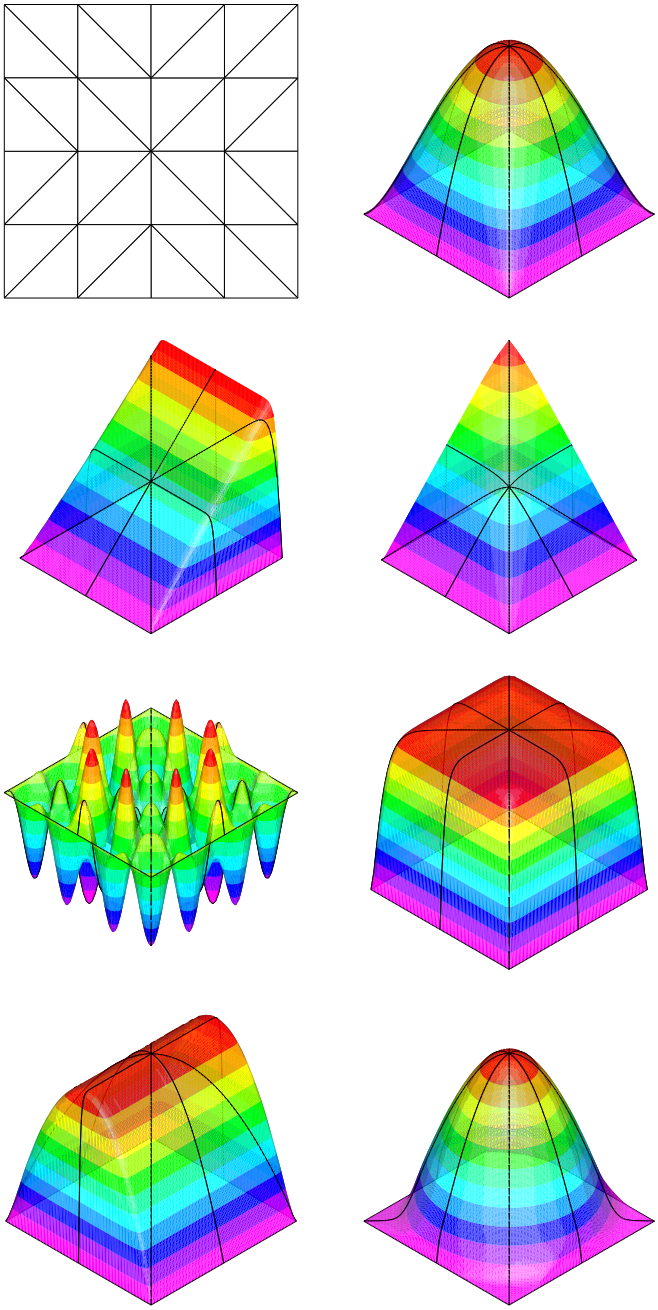

Figure 1: A 5×5 uniform triangulation, and solutions to problems 1-7.

Table 1: Performance comparison.

n	N	Levels	Digits	Cycles	Init.	Solve
Problem 1, $dtol = 10^{-2}$						
51	2601	7	8.4	3	2.1e-1	5.1e-2
101	10201	7	6.8	3	1.0e 0	2.9e-1
201	40401	9	6.2	3	4.5e 0	1.4e 0
Problem 2, $dtol = 10^{-3}$						
51	2601	7	7.5	1	2.5e-1	6.4e-2
101	10201	8	6.1	1	1.2e 0	3.7e-1
201	40401	9	10.3	3	6.5e 0	4.6e 0
Problem 3, $dtol = 10^{-3}$						
51	2601	7	11.6	1	5.8e-1	1.2e-1
101	10201	8	6.5	1	4.9e 0	5.1e-1
201	40401	8	10.0	2	6.8e 0	3.7e 0
Problem 4, $dtol = 10^{-4}$						
51	2601	6	6.7	1	3.7e-1	3.8e-2
101	10201	7	6.1	3	2.3e 0	4.9e-1
201	40401	7	7.0	4	13.4e 0	2.9e 0
Problem 5, $dtol = 10^{-2}$						
51	2601	7	7.6	2	2.7e-1	4.1e-2
101	10201	7	6.4	2	1.2e 0	2.3e-1
201	40401	8	6.7	2	4.4e 0	1.0e 0
Problem 6, $dtol = 10^{-4}$						
51	2601	6	8.6	1	2.0e-1	2.4e-2
101	10201	7	8.2	1	8.9e-1	1.4e-1
201	40401	8	7.5	1	4.1e 0	6.7e-1
Problem 7, $dtol = 10^{-3}$						
51	2601	6	10.3	2	2.9e-1	1.0e-1
101	10201	7	7.7	2	1.6e 0	6.6e-1
201	40401	8	6.6	2	8.2e 0	3.1e 0

finite elements and the usual nodal basis functions are used in *PLTMG* to construct the linear systems.

In Table 1, we summarize the results of the calculation. Here *Levels* refers to the number of levels used in the calculation. In this test, the parameter $maxlvl$ was sufficiently large that it had no effect on the computation. The fill-in control parameter $maxfil$ was also sufficiently large that it had no effect on the computation. The drop tolerance was set as indicated; although not carefully optimized, the tolerance was crudely chosen according to the difficulty of the problem to produce roughly comparable results for all problems. The initial guess for all problems was $x_0 = 0$.

The parameter *Digits* refers to

$$Digits = -\log \frac{\|r_k\|}{\|r_0\|}. \tag{9}$$

In these experiments, we asked for 6 digits of accuracy. The column labeled *Cycles* indicates the number of multigrid cycles (accelerated by composite step conjugate gradients or biconjugate gradients) that were used to achieve the indicated number of digits. Finally, the last two columns, labeled *Init.* and *Solve*, record the CPU time for the initialization and solution phases of the algorithm, respectively. Initialization includes all the orderings, incomplete factorizations, and computation of transfer matrices used in the multigraph preconditioner. Solution includes the time to solve (1) to at least 6 digits given the preconditioner. These experiments were run on an SGI Octane R10000 250mhz, using double precision arithmetic and the f90 compiler.

In analyzing these results, it is clear that our procedure does reasonably well on all of the problems. Although it appears that the rate of convergence is not always independent of N, it seems apparent that the work is growing no faster than logarithmically. CPU times for larger values of N are affected by cache performance as well as the slightly larger number of cycles.

In our next experiment, we illustrate the effect of the parameters $maxlvl$ and $dtol$. For the $\mathcal{L}_1$, $\mathcal{L}_4$, and $\mathcal{L}_7$ and $N = 40401$, we solved the problem for $dtol = 10^{-k}$, $1 \le k \le 4$ and $1 \le maxlvl \le 3$. $\mathcal{L}_4$ and $\mathcal{L}_7$ are the two most challenging problems in this suite. The results are given in Table 2. Although we expect all iterations to eventually converge (at least in exact arithmetic), we terminated the iteration after $maxcg = 25$ steps or when the solution had 6 digits, as measured by (9).

Here we see that, in general, decreasing the drop tolerance or increasing the number of levels improves the convergence behavior of the method. On the other hand, the timings do not always follow the same trend. For example, increasing the number of levels from $maxlvl = 1$ to $maxlvl = 2$ often decreases the number of cycles but increases the time. This is because for $maxlvl = 1$, our method defaults to the standard CG of BCG iteration with the incomplete factorization preconditioner. When $maxlvl > 1$, one pre-smoothing and one post-smoothing step are used for the largest matrix. With the additional cost of the recursion, the overall cost of the preconditioner is more than double the cost for the case $maxlvl = 1$.

We also note that, unlike the classical multigrid method, where the coarsest matrix is solved exactly, in our code we have chosen to approximately solve the coarsest system using just one smoothing iteration using the incomplete factorization. When the maximum number of levels are used, as in Table 1, the smallest system is typically 1×1 or 2×2, and this is an irrelevant remark. However, in the case of Table 2, the fact that the smallest system is not solved exactly significantly influences the convergence.

Table 2: Dependence of convergence of $dtol$ and $maxlvl$.

$dtol$	$maxlvl$	Digits	Cycles	Init.	Solve
Problem 1, $N = 40401$					
10^{-1}	1	3.7	25	1.1	2.7
	2	4.3	25	2.6	6.3
	3	6.2	22	3.7	7.6
10^{-2}	1	5.6	25	1.5	3.3
	2	6.2	13	3.5	4.3
	3	6.1	8	4.2	3.3
10^{-3}	1	6.0	12	2.2	1.8
	2	6.2	6	4.9	2.3
	3	7.0	4	5.6	2.0
10^{-4}	1	6.5	5	3.7	1.0
	2	6.5	2	7.9	1.2
	3	8.5	2	5.6	1.5
Problem 4, $N = 40401$					
10^{-1}	1	3.1	25	1.2	3.0
	2	3.6	25	2.7	7.2
	3	3.9	25	3.8	10.2
10^{-2}	1	3.7	25	1.5	4.4
	2	4.2	25	3.5	11.0
	3	2.5	25	4.2	10.0
10^{-3}	1	6.1	25	3.4	5.0
	2	3.7	25	7.7	11.8
	3	5.7	25	9.6	14.1
10^{-4}	1	6.7	5	8.1	1.3
	2	6.7	4	12.1	2.4
	3	7.0	4	12.9	2.8
Problem 7, $N = 40401$					
10^{-1}	1	3.1	25	1.4	7.9
	2	3.1	25	3.2	18.9
	3	4.5	25	4.0	22.3
10^{-2}	1	5.5	25	1.9	8.7
	2	6.2	10	4.5	9.5
	3	6.4	7	5.2	6.9
10^{-3}	1	7.0	6	2.9	2.3
	2	7.5	4	6.9	3.6
	3	7.7	3	7.8	3.8
10^{-4}	1	6.2	3	3.8	1.2
	2	8.8	2	9.4	2.4
	3	7.4	1	10.5	2.2

Finally we note biconjugate gradient iteration used for nonsymmetric problems requires two matrix multiplies and two preconditioning applications (for the matrix and its transpose), so the overall cost per step is about twice that of the regular conjugate gradient iteration.

References

[Ban98]Randolph E. Bank. *PLTMG: A Software Package for Solving Elliptic Partial DifferentiaEquations, Users' Guide 8.0l*, volume 5 of *Software, Environments and Tools*. SIAM, Philadelphia, 1998.

[BC93]Randolph E. Bank and Tony F. Chan. An analysis of the composite step biconjugate gradient method. *Numerische Mathematik*, 66:295–319, 1993.

[BS00]Randolph E. Bank and R. Kent Smith. An algebraic multilevel multigraph algorithm. Technical report, University of California at San Diego, 2000. submitted to SIAM J. on Scientific Computing.

[EGSS82]S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM J. Sci. Stat. Comput.*, 2:225–237, 1982.

[GL81]Alan George and Joseph Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Wag99]Christian Wagner. Introduction to algebraic multigrid. Technical report, Interdisziplinäres Zentrum für Wissenschaftliches Rechnen,Universität Heidelberg, 1999.