SIAM J. MATRIX ANAL. APPL.
Vol. 31, No. 2, pp. 694–720
© 2009 Society for Industrial and Applied Mathematics

COMBINATORIAL PRECONDITIONERS FOR SCALAR ELLIPTIC FINITE-ELEMENT PROBLEMS*

HAIM AVRON†, DORON CHEN‡, GIL SHKLARSKI†, AND SIVAN TOLEDO§

Abstract. We present a new preconditioner for linear systems arising from finite-element discretizations of scalar elliptic partial differential equations (PDE's). The solver splits the collection $\{K_e\}$ of element matrices into a subset of matrices that are approximable by diagonally dominant matrices and a subset of matrices that are not approximable. The approximable K_e's are approximated by diagonally dominant matrices L_e's that are assembled to form a global diagonally dominant matrix L. A combinatorial graph algorithm then approximates L by another diagonally dominant matrix M that is easier to factor. Finally, M is added to the inapproximable elements to form the preconditioner, which is then factored. When all the element matrices are approximable, which is often the case, the preconditioner is provably efficient. Approximating element matrices by diagonally dominant ones is not a new idea, but we present a new approximation method which is both efficient and provably good. The splitting idea is simple and natural in the context of combinatorial preconditioners, but hard to exploit in other preconditioning paradigms. Experimental results show that on problems in which some of the K_e's are ill conditioned, our new preconditioner is more effective than an algebraic multigrid solver, than an incomplete-factorization preconditioner, and than a direct solver.

Key words. preconditioning, finite elements, support preconditioners, combinatorial preconditioners

AMS subject classifications. 65Y20, 65N22, 65F10

DOI. 10.1137/060675940

1. Introduction.

Symmetric semidefinite matrices that are diagonally dominant are relatively easy to precondition. This observation has led two groups of researchers to propose linear solvers that are based on element-by-element approximation of a given coefficient matrix by a diagonally dominant matrix. The diagonally dominant approximation is then used to construct a preconditioner, which is applied to the original problem. Before we describe their proposals and our new contributions, however, we describe the formulation of the problem.

These existing techniques, as well as our new algorithm, use a given finite-element discretization of the following problem: Find $u\colon \Omega \longrightarrow \mathbb{R}$ satisfying

$$\begin{aligned} \nabla \cdot (\theta(z)\nabla u) &= -f && \text{on } \Omega \\ u &= u_0 && \text{on } \Gamma_1 \ , \\ \theta(z)\partial u/\partial n &= g && \text{on } \Gamma_2 \ . \end{aligned} \tag{1.1}$$

The domain Ω is a bounded open set of $\mathbb{R}^d$ and Γ_1 and Γ_2 form a partition of the boundary of Ω. The *conductivity* θ is a spatially varying d-by-d symmetric positive

*Received by the editors November 27, 2006; accepted for publication (in revised form) by E. Ng July 17, 2008; published electronically June 17, 2009. This research was supported by an IBM Faculty Partnership Award, by grant 848/04 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and by grant 2002261 from the United-States–Israel Binational Science Foundation.

http://www.siam.org/journals/simax/31-2/67594.html

†Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel (haima@tau.ac.il, shagil@tau.ac.il).

‡IBM Haifa Research Lab, Tel-Aviv Site, Israel (cdoron@il.ibm.com).

§Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel and MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA 02139 (stoledo@tau.ac.il).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

definite matrix, f is a scalar *forcing function*, u_0 is a *Dirichlet boundary condition*, and g is a *Neumann boundary condition*.

We assume that the discretization of (1.1) leads to an algebraic system of equations

$$Kx = b\ .$$

The matrix $K \in \mathbb{R}^{n\times n}$ is called a *stiffness matrix*, and it is a sum of *element matrices*, $K = \sum_{e\in E} K_e$. Each element matrix K_e corresponds to a subset of Ω called a *finite element*. The elements are disjoint except perhaps for their boundaries and their union is Ω. We assume that each element matrix K_e is symmetric, positive semidefinite, and zero outside a small set of n_e rows and columns. In most cases n_e is uniformly bounded by a small integer (in our experiments n_e is 4 or 10). We denote the set of nonzero rows and columns of K_e by $\mathcal{N}_e$.

For the problem (1.1), all the element matrices are singular with rank $n_e - 1$ and null vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$. This is a key aspect of techniques for approximating element matrices by diagonally dominant ones: the methods only works when element matrices are either nonsingular or are singular with rank $n_e - 1$ and null vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

Reitzinger and his collaborators were the first to propose element-by-element approximation by symmetric and diagonally dominant (SDD) matrices [22, 37, 28]. They proposed two element-approximation techniques. In one technique, the approximation problem is formulated as an optimization problem, in which one tries to minimize the generalized condition number of a given element matrix K_e and an SDD matrix L_e; a generic optimization algorithm is then used to find L_e. The second technique uses symbolic algebra to formulate the approximation problem, and uses symbolic-algebra algorithms to find L_e. Both methods are quite expensive and slow. A row or a column that is zero in K_e is also zero in L_e, so the L_e's are also very sparse. Once the L_e's are found, they are assembled. When all the L_e's are SDD, their sum L is also SDD. The matrix L is then used to construct an algebraic multigrid solver, which is used as a preconditioner for K.

Boman, Hendrickson, and Vavasis proposed a different element-by-element approximation technique [10]. They used geometric information about the elements and the values of θ to directly construct L_e. The approximation algorithm is inexpensive. They show that under certain conditions on the continuous problem and on the finite-element mesh, the approximations are all good. They proposed to use L to construct a combinatorial preconditioner rather then apply multigrid to L. This proposal was based on the observation that over the last decade, several provably good combinatorial graph algorithms for constructing preconditioners for SDD matrices have been developed [46, 21, 20, 7, 43, 44, 18, 45, 29]. Some of them, as well as some combinatorial heuristics, have been shown to be effective in practice [15, 25, 36, 38, 19, 35].

In this paper, we extend this paradigm in two ways. First, we propose a novel, effective, and purely algebraic method for approximating an element matrix K_e by an SDD matrix L_e. Our approximation algorithm is relatively inexpensive and provably good: the spectral distance between K_e and L_e is within a $n_e^2/2$ factor of the best possible for an SDD approximation of K_e, where n_e is the number of nonzero rows and columns in K_e. In particular, this means that our algorithm produces good approximations whenever the algorithm of Boman et al. does. Furthermore, there exist element matrices that are ill conditioned and that are far from diagonal dominance (in the sense that a small perturbation of their entries cannot make them diagonally dominant), which our algorithm approximates well.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Vavasis has shown [49] that some of the results in this paper can be used to find an optimal approximation of an element matrix K_e by an SDD matrix L_e. Since his method is computationally expensive, it is not clear if it is effective in a practical solver. Our code uses our provably good, cheap to compute, but suboptimal approximation instead.

Our second contribution to this paradigm is a technique to handle problems in which some of the element matrices cannot be well approximated by SDD matrices. This may arise because of anisotropy in θ, or because of ill-shaped elements, for example. Our algorithm splits the elements into two sets: the set $E(t)$ in which L_e is a good approximation of K_e, and the rest (where t is a parameter that determines how good we require approximations to be). We then scale and assemble the good element-by-element approximations to form $L = \sum_{e \in E(t)} \alpha_e L_e$. Next, we use a combinatorial graph algorithm to construct an easy-to-factor approximation M of L. Finally, we scale M and add γM to the inapproximable elements to form $\gamma M + \sum_{e \notin E(t)} K_e$. We factor this matrix and use it as a factored preconditioner for $K = \sum_e K_e$.

The splitting idea is simple, but there is no easy way to exploit it in most preconditioning paradigms. In Reitzinger's method, for example, one could build an algebraic multigrid solver for $L = \sum_{e \in E(t)} \alpha_e L_e$, but how would one incorporate the inapproximable elements into this solver? There is no obvious way to do this. An algebraic multigrid solver for $\gamma M + \sum_{e \notin E(t)} K_e$ is unlikely to be much better as a preconditioner than an algebraic multigrid solver for K, because $\gamma M + \sum_{e \notin E(t)} K_e$ is far from diagonal dominance unless all the elements are approximable.

The reason that the splitting idea works well with combinatorial preconditioners is that combinatorial preconditioning algorithms sparsify the SDD matrix L that is given to them as input. The sparsification makes the Cholesky factorization of the sparsified M much cheaper and sparser than the factorization of L. If most of the elements are approximable, adding $\sum_{e \notin E(t)} K_e$ to γM is likely to yield a preconditioner that is still cheap to factor.

Once the Cholesky factor of the preconditioners is computed, we use it in a preconditioned symmetric Krylov-subspace solver such as conjugate gradients [17, 24], SYMMLQ, or MINRES [33]. For most of the combinatorial algorithms that we can use to construct M, it is possible to show that the preconditioner is spectrally close to K. The spectral bounds give a bound on the number of iterations that the Krylov-subspace algorithm performs.

Experimental results that explore the performance and behavior of our solver show that the solver is highly reliable. In particular, on some problems other solvers, including an algebraic-multigrid solver and an incomplete Cholesky solver, either fail or are very slow; our solver handles these problems without difficulty.

The rest of the paper is organized as follows. Section 2 presents our element-approximation method. The scaling of the element-by-element approximation is presented in section 3. The combinatorial sparsification phase is described in section 4, and the handling of inapproximable elements in section 5. The costs associated with the different phases of the solver are described in section 6. Experimental results are presented in section 7. We mention some open problems that this research raises in section 8.

2. Nearly optimal element-by-element approximations. In this section we show how to compute a nearly optimal SDD approximation L_e to a given symmetric positive semidefinite matrix K_e that is either nonsingular or has a null space consisting only of the constant vector.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

2.1. Defining the problem. Let $\mathbb{S}$ be a linear subspace of $\mathbb{R}^n$ (the results of this section also apply to $\mathbb{C}^n$, but we use $\mathbb{R}^n$ in order to keep the discussion concrete). We denote by $\mathbb{R}^{\mathbb{S}} \subseteq \mathbb{R}^{n \times n}$ the set of symmetric positive (semi)definite matrices whose null space is exactly $\mathbb{S}$.

DEFINITION 2.1. *Given two matrices A and B in $\mathbb{R}^{\mathbb{S}}$, a* finite generalized eigenvalue *λ of (A, B) is a scalar satisfying $Ax = \lambda Bx$ for some $x \notin \mathbb{S}$. The* generalized finite spectrum *$\Lambda(A, B)$ is the set of finite generalized eigenvalues of (A, B), and the* generalized condition number *$\kappa(A, B)$ is*

$$\kappa(A, B) = \frac{\max \Lambda(A, B)}{\min \Lambda(A, B)} .$$

(This definition can be generalized to the case of different null spaces, but this is irrelevant for this paper.) We informally refer to $\kappa(A, B)$ as the spectral distance between A and B.

We refer to the following optimization problem as the *optimal symmetric semidefinite approximation problem.*

PROBLEM 2.2. *Let A be a symmetric positive (semi)definite matrix with null space $\mathbb{S}$ and let $B_1, B_2, \ldots, B_m$ be rank-1 symmetric positive semidefinite matrices. Find coefficients $d_1, d_2, \ldots, d_m$ that minimize the generalized condition number of A and*

$$B_{\text{opt}} = \sum_{j=1}^{m} d_j^2 B_j$$

under the constraint null(B_{opt}) = null(A), or decide that the null spaces cannot match under any d_j's. We can assume without loss of generality that all the B_j's have unit norm.

A slightly different representation of the problem is useful for characterizing the optimal solution. Let $B_j = Z_j Z_j^T$, where Z_j is a column vector. Let Z be an n-by-m matrix whose columns are the Z_j's. Then

$$\sum_{j=1}^{m} d_j^2 B_j = \sum_{j=1}^{m} d_j^2 Z_j Z_j^T = ZDD^T Z^T$$

where D is the m-by-m diagonal matrix with d_j as its jth diagonal element. This yields an equivalent formulation of the optimal symmetric semidefinite approximation problem.

PROBLEM 2.3. *Given a symmetric positive (semi)definite n-by-n matrix A with null space $\mathbb{S}$ and an n-by-m matrix Z, find an m-by-m diagonal matrix D such that null$(ZDD^T Z^T) = \mathbb{S}$ and that minimizes the generalized condition number $\kappa(A, ZDD^T Z^T)$, or report that no such D exists.*

We are interested in cases where range(Z) = range(A), where the problem is clearly feasible.

Vavasis has pointed out [49] that minimizing $\kappa(Z^T D^T DZ, A)$ is equivalent to finding a diagonal nonnegative matrix S that minimizes $\max \Lambda(Z^T SZ, A)$ such that $\min \Lambda(Z^T SZ, A) \geq 1$, and taking $D = S^{1/2}$. He showed that this optimization problem can be solved as a convex semidefinite programming problem: find the minimum t and the corresponding S for which the constraints $\min \Lambda(tA, Z^T SZ) \geq 1$ and $\min \Lambda(Z^T SZ, A) \geq 1$ can be satisfied simultaneously. Convex semidefinite programming problems can be solved in polynomial time. An algorithm based on this

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

observation can find an optimal SDD approximation, but applying it might be costly. We have decided to pursue a different direction by finding a suboptimal but provably good approximation using a fast and simple algorithm.

2.2. From generalized condition numbers to condition numbers. The main tool that we use to find nearly optimal solutions to Problem (2.2) is a reduction of the problem to the well-studied problem of scaling the columns of a matrix to minimize its condition number.

DEFINITION 2.4. *Given a matrix A, let $\sigma_{\max}$ be the largest singular value of A and $\sigma_{\min}$ be the smallest nonzero singular value of A. The* condition number *of A is $\kappa(A) = \sigma_{\max}/\sigma_{\min}$. If $A \in \mathbb{R}^{\mathbb{S}}$, then $\kappa(A) = \kappa(A, P_{\perp\mathbb{S}})$, where $P_{\perp\mathbb{S}}$ is the orthogonal projector onto the subspace orthogonal to $\mathbb{S}$.*

The following lemma, which is a generalization of [9, Theorem 4.5], is the key to the characterization of B_{opt}.

LEMMA 2.5. *Let $A = UU^T$ and $B = VV^T$, where U and V are real valued matrices of order $n \times m$. Assume that A and B are symmetric, positive semidefinite and null(A) = null(B). We have*

$$\Lambda(A, B) = \Sigma^2\left(V^+U\right)$$

and

$$\Lambda(A, B) = \Sigma^{-2}\left(U^+V\right) .$$

In these expressions, $\Sigma(\cdot)$ is the set of nonzero singular values of the matrix within the parentheses, Σ^ℓ denotes the same singular values to the ℓth power, and V^+ denotes the Moore–Penrose pseudoinverse of V.

Proof. Both U and V have n rows, so U^+ and V^+ have n columns. Therefore, the products V^+U and U^+V exist. Therefore,

$$\begin{aligned}
\Sigma^2\left(V^+U\right) &= \Lambda\left(V^+UU^T\left(V^+\right)^T\right) \\
&= \Lambda\left(\left(V^+\right)^T V^+UU^T\right) \\
&= \Lambda\left(\left(VV^T\right)^+ UU^T\right) \\
&= \Lambda\left(B^+A\right) .
\end{aligned}$$

($\Lambda(\cdot)$ denotes the set of eigenvalues of the argument.) For the second line we use the following observation. If X is n-by-k and Y is k-by-n, then the nonzero eigenvalues of XY and YX are the same. The second line follows from this observation for $X = \left(V^+\right)^T$ and $Y = V^+UU^T$. The third line follows from the equality $\left(XX^T\right)^+ = \left(X^+\right)^T X^+$ for an order n-by-k matrix X [6, Proposition 6.1.6].

It is sufficient to show that $\Lambda\left(B^+A\right) = \Lambda(A, B)$ in order to prove the first part of the lemma. Let $\lambda \in \Lambda(A, B)$, then there exists a vector $x \perp \text{null}(B) = \text{null}(A)$, such that $Ax = \lambda Bx$. Since B is symmetric, $x \in \text{range}(B) = \text{range}(B^T)$. Therefore, $B^+Ax = \lambda B^+Bx = \lambda x$. The last equality follows from the fact that B^+B is a projector onto range(B^T) [6, Proposition 6.1.6]. Therefore, $\Lambda\left(B^+A\right) \supseteq \Lambda(A, B)$. Let $\lambda \in \Lambda(B^+A)$, then there exists a vector x, such that $B^+Ax = \lambda x$. Therefore, $Ax = BB^+Ax = \lambda Bx$. The first equality follows from the fact that range(A) = range(B) and [6, Proposition 6.1.7]. Therefore, $\Lambda\left(B^+A\right) \subseteq \Lambda(A, B)$, which shows the equality.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

The second result $\Lambda(A,B) = \Sigma^{-2}(U^+V)$ follows from replacing the roles of A and B in the analysis above and from the equality $\Lambda(A,B) = \Lambda^{-1}(B,A)$. The reversal yields

$$\Lambda(A,B) = \Lambda^{-1}(B,A) = \left(\Sigma^2\left(U^+V\right)\right)^{-1} = \Sigma^{-2}\left(U^+V\right) . \qquad \square$$

This lemma shows that Problems (2.2) and (2.3) can be reduced to the problem of scaling the columns of a single matrix to minimize its condition number. Let $A = UU^T$ and let Z satisfy range(Z) = range(A). (If A is symmetric positive semidefinite but U is not given, we can compute such a U from the Cholesky factorization of A or from its eigendecomposition.) According to the lemma,

$$\Lambda\left(A, ZDD^TZ^T\right) = \Sigma^{-2}\left(U^+ZD\right) .$$

Therefore, minimizing $\kappa(A, ZDD^TZ^T)$ is equivalent to minimizing the condition number $\kappa(U^+ZD)$ under the constraint range(ZD) = range(Z).

The other equality in Lemma 2.5 does not appear to be useful for such a reduction. According to the equality

$$\Lambda\left(A, ZDD^TZ^T\right) = \Sigma^2\left((ZD)^+U\right),$$

but unfortunately, there does not appear to be a way to simplify $(ZD)^+U$ in a way that makes D appear as a row or column scaling. (Note that in general, $(ZD)^+ \neq D^+Z^+$.)

The problem of scaling the columns of a matrix to minimize its condition number has been investigated extensively. Although efficient algorithms for minimizing the condition number of a rectangular matrix using diagonal scaling do not exist, there is a simple approximation algorithm. It may be possible to use Vavasis's algorithm [49] to solve this problem too, but this is not the concern of this paper.

2.3. Computing the pseudoinverse of a factor of an element matrix. Before we can find a scaling matrix D, we need to compute U^+ from a given element matrix $K_e = UU^T$ and to form U^+Z.

We compute U^+ in one of two ways. If the input to our solver is an element matrix K_e with a known null space, we can compute U^+ from an eigendecomposition of K_e. Let $K_e = Q_e\Lambda_eQ_e^T$ be the *reduced* eigendecomposition of K_e (that is, Q_e is n-by-rank(K_e) and Λ_e is a rank(K_e)-by-rank(K_e) nonsingular diagonal matrix). We have $K_e = Q_e\Lambda_e^{1/2}(Q_e\Lambda_e^{1/2})^T$ so we can set $U = Q_e\Lambda_e^{1/2}$, so $U^+ = \Lambda_e^{-1/2}Q_e^T$.

Many finite-element discretization techniques actually generate the element matrices in a factored form. If that is the case, then some symmetric factor F of $K_e = FF^T$ is given to our solver as input. In that case, we compute a reduced singular-value decomposition SVD of F, $F = Q_e\Sigma_eR_e^T$, where Σ_e is square, diagonal, and invertible, and R_e is square and unitary, both of order rank(F). Since

$$K_e = FF^T = Q_e\Sigma_eR_e^TR_e\Sigma_e^TQ_e^T = Q_e\Sigma_e^2Q_e^T$$

is an eigendecomposition of K_e, we can set $U = Q_e\Sigma_e$ and we have $K_e = UU^T$. In this case $U^+ = \Sigma_e^{-1}Q_e^T$. This method is more accurate when K_e is ill conditioned.

Note that in both cases we do not need to explicitly form U^+; both methods provide a factorization of U^+ that we can use to apply it to Z.

Once we form U^+Z, our solver searches for a diagonal matrix D that brings the condition number of U^+ZD close to the minimal condition number possible. This

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

problem is better understood when U^+Z is full rank. Fortunately, in our case it always is.

LEMMA 2.6. *Let U be a full rank m-by-n matrix, $m \geq n$, and let Z be an m-by-ℓ matrix with range(Z) = range(U). Then U^+Z has full row rank.*

Proof. Since range(Z) = range(U), there exists an n-by-l matrix C such that $Z = UC$. By definition, rank$(Z) \leq$ rank$(C) \leq n$. Moreover, since range(Z) = range(U) and U is full rank, we have that $n =$ rank(Z). Therefore, rank$(C) = n$.

It is sufficient to show that $C = U^+Z$ to conclude the proof of the lemma. Since U is full rank and $m \geq n$, the product U^+U is the n-by-n identity matrix. Therefore, $U^+Z = U^+UC = C$. ∎

Without the assumption range(Z) = range(U), the matrix U^+Z can be rank deficient even if both $U^+$and Z are full rank.

Example 2.7. Let

$$U = \begin{bmatrix} 2 & 0 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix}.$$

The columns of U are orthogonal, range$(Z) \neq$ range(U). This gives

$$U^+ = \begin{bmatrix} 1/3 & -1/6 & -1/6 \\ 0 & -1/2 & 1/2 \end{bmatrix}$$

and

$$U^+Z = \begin{bmatrix} 0 & 1/2 \\ 0 & 1/2 \end{bmatrix},$$

which is clearly not full rank.

2.4. Nearly optimal column scaling. Given a matrix U^+Z, we wish to find a diagonal matrix D that minimizes the condition number of U^+ZD, under the assumption range(Z) = range(U), and under the constraint that range(ZD) = range(Z).

To keep the notation simple and consistent with the literature, in this section we use A to denote U^+Z and we use m and n to denote the number of rows and columns in $A = U^+Z$.

The key result that we need is due to van der Sluis [47], who showed that choosing $\tilde{D}$ such that all the columns of $A\tilde{D}$ have unit 2-norm brings $A\tilde{D}$ to within a factor of $\sqrt{n}$ of the optimal scaling. Van der Sluis, extending an earlier result of Bauer for square invertible matrices [4], analyzed the full-rank case.

Given an m-by-n matrix A, $m \geq n$, and a nonsingular diagonal D van der Sluis defined

$$\kappa_{\text{vdS}}(AD) = \frac{\|AD\|_2}{\min_{x \neq 0} \|ADx\|_2 / \|x\|_2} \tag{2.1}$$

(his original definition is not specific to the 2-norm, but this is irrelevant for us). If A is nonsingular, then $\kappa_{\text{vdS}}(AD) = \kappa(AD)$. He, like Bauer, was interested in finding the diagonal matrix D that minimizes (2.1). This definition of the problem implicitly assumes that A is full rank, otherwise $\kappa_{\text{vdS}}(AD) = \infty$ for any nonsingular diagonal D. Also, if A is full rank, then the minimizing D must give a full-rank AD. If A has more columns than rows, we can use a complementary result by van der Sluis, one that uses row scaling on a matrix with more rows than columns. Van der Sluis

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

result shows that if the rows of $\tilde{D}A$ have unit 2-norm, then $\kappa_{\text{vdS}}(\tilde{D}A)$ is within a factor of $\sqrt{m}$ of the minimum possible. This gives us the result that we need, because $\kappa_{\text{vdS}}(AD) = \kappa_{\text{vdS}}(D^T A^T)$. Shapiro showed that, in the general case, van der Sluis's estimate on $\kappa_{\text{vdS}}(A\tilde{D})$ cannot be improved by more than a $\sqrt{2}$ factor [40]. The following are formal statements of the last two cited results.

LEMMA 2.8 (Part of Theorem 3.5 in [47]). *Let A be an m-by-n full-rank matrix and let $\tilde{D}$ be a diagonal matrix such that in $\tilde{D}A$ all rows have equal 2-norm. Then for every diagonal matrix D we have*

$$\kappa_{\text{vdS}}(\tilde{D}A) \leq \sqrt{m}\kappa_{\text{vdS}}(DA)\,.$$

LEMMA 2.9 (Theorem in [40]). *For every $\epsilon > 0$ there exists an n-by-n nonsingular, real valued matrix A such that if:*

1. *$\tilde{D}$ is a diagonal matrix such that all the diagonal elements of $\tilde{D}A^T A\tilde{D}$ are equal to one, and*
2. *D is a diagonal matrix such that $\kappa(AD)$ is minimized,*

then

$$\kappa\left(A\tilde{D}\right) > \left(\sqrt{n/2} - \epsilon\right)\kappa(AD)\,.$$

As we have shown in Lemma 2.6, that matrix $A = U^+ Z$ whose columns we need to scale is full rank, so van der Sluis's results apply to it.

We note that further progress has been made in this area for square invertible A's, but it appears that this progress is not applicable to our application when $A = U^+ Z$ is rectangular (which is usually the case). Braatz and Morari showed that for a square invertible A, the minimum of $\kappa(AD)$ over all positive diagonal matrices D can be found by solving a related optimization problem, which is convex [12]. Their paper states that this result also applies to the rectangular case [12, Remark 2.5]; what they mean by that comment is that the related optimization problem minimizes $\|AD\|_2\|D^{-1}A^+\|_2$ [11], whereas we need to minimize $\kappa(AD) = \|AD\|_2\|(AD)^+\|_2$.

2.5. Nearly optimal symmetric diagonally dominant approximations. We now turn our attention to the matrices that arise as element matrices in finite-element discretizations of (1.1). Such a matrix K_e has null space that is spanned by the constant vector and by unit vectors e_j for every zero column j of K_e. The part of the null space that is spanned by the unit vectors is irrelevant, so we assume that we are dealing with a matrix A whose null space is spanned by constant vector $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

We wish to approximate a symmetric semidefinite matrix A with this null space (or possibly a nonsingular matrix) by a symmetric diagonally dominant matrix B,

$$B_{ii} \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |B_{ij}|\ .$$

To define a matrix Z such that the expression $ZDD^T Z^T$ can generate any symmetric diagonally dominant matrix, we define the following vectors.

DEFINITION 2.10. *Let $1 \leq i, j \leq n$, $i \neq j$. A length-n* positive edge vector*, denoted $\langle i, -j\rangle$, is the vector*

$$\langle i, -j\rangle_k = \begin{cases} +1 & k = i \\ -1 & k = j \\ 0 & \textit{otherwise.} \end{cases}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

A negative edge vector $\langle i, j\rangle$ *is the vector*

$$\langle i,j\rangle_k = \begin{cases} +1 & k = i \\ +1 & k = j \\ 0 & \textit{otherwise.} \end{cases}$$

A vertex vector $\langle i\rangle$ *is the unit vector*

$$\langle i\rangle_k = \begin{cases} +1 & k = i \\ 0 & \textit{otherwise.} \end{cases}$$

A symmetric diagonally dominant matrix can always be expressed as a sum of outer products of scaled edge and vertex vectors. Therefore, we can conservatively define Z to be the matrix whose columns are all the positive edge vectors, all the negative edge vectors, and all the vertex vectors.

If A is singular and its null space is the constant vector, we can do better. Chen and Toledo provided a combinatorial characterization of the null space of SDD matrices [16].

LEMMA 2.11 ([16]). *Let A be a symmetric diagonally dominant matrix whose null space is the constant vector. Then A is a sum of outer products of scaled positive edge vectors. Furthermore, the null space of a symmetric diagonally dominant matrix with a positive off-diagonal element (corresponding to an outer product of a scaled negative edge vector) cannot be span* $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$.

Therefore, if A is singular with this null space, we only need to include in the column set Z the set of positive edge vectors. If A is nonsingular, we also include in Z negative edge vectors and vertex vectors.

We can also create even sparser Z's; they will not allow us to express every SDD B as $B = ZDD^TZ^T$, but they will have the same null space as A. To define these sparser Z's, we need to view the edge vector $\langle i, -j\rangle$ as an edge that connects vertex i to vertex j in a graph whose vertices are the integers 1 through n. The null space of ZZ^T is the constant vector if and only if the columns of Z, viewed as edges of a graph, represent a connected graph. Therefore, we can build an approximation $B = ZDD^TZ^T$ by selecting an arbitrary connected graph on the vertices $\{1, \ldots, n\}$. By [16, Lemma 4.2], if A is nonsingular, we can include in Z the positive edge vectors of a connected graph plus one arbitrary vertex vector.

If A is well conditioned (apart perhaps from one zero eigenvalue), we can build a good approximation $B = ZDD^TZ^T$ even without the column-scaling technique of Lemma 2.5. In particular, this avoids the computation of the pseudo-inverse of a factor U of $A = UU^T$. Clearly, if A is nonsingular and well conditioned, then we can use I as an approximation: the generalized condition number $\kappa(A, I)$ is $\kappa(A)$. If A has rank $n-1$ and the constant vector is its null vector, then

$$B_{C(n_e)} = \frac{1}{n_e}\begin{bmatrix} n_e-1 & -1 & -1 & \cdots & -1 \\ -1 & n_e-1 & -1 & \cdots & -1 \\ -1 & -1 & n_e-1 & \cdots & -1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & \cdots & n_e-1 \end{bmatrix}$$

yields

$$\kappa(A, B_{C(n_e)}) = \sigma_{\max}(A)/\sigma_{\min}(A), \tag{2.2}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

which we assumed is low ($\sigma_{\max}(A)$ is the largest singular value of A, and $\sigma_{\min}(A)$ is the smallest nonzero singular value of A). The matrix $B_{C(n_e)}$ is the Laplacian of the complete graph, and it is clearly SDD. The identity (2.2) follows from the fact that $B_{C(n_e)}$ is an orthogonal projection onto range(A). The following lemma summarizes this discussion.

LEMMA 2.12. *Let A be a symmetric positive (semi)definite matrix. If A is nonsingular, or if the null space of A is span $\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^T$, then there is an SDD matrix B such that $\kappa(A,B) \leq \kappa(A)$.*

This result may seem trivial (it is), but it is nonetheless important. The global stiffness matrix $K = \sum_e K_e$ is often ill conditioned, but the individual element matrices K_e are usually well conditioned. Since we build the approximation $L = \sum_e L_e$ element by element, this lemma is often applicable: When K_e is well conditioned, we can set L_e to be an extension of $B_{C(n_e)}$ into an n-by-n matrix. The rows and columns of the scaled $B_{C(n_e)}$ are mapped to rows and columns $\mathcal{N}_e$ of L_e and the rest of L_e is zero.

For well-conditioned A's, we can also trade the approximation quality for a sparser approximation than $B_{C(n_e)}$. The matrix

$$B_{S(n_e)} = \frac{1}{n_e}\begin{bmatrix} n_e - 1 & -1 & -1 & \cdots & -1 \\ -1 & 1 & 0 & \cdots & 0 \\ -1 & & 1 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ -1 & & 0 & \cdots & 1 \end{bmatrix}$$

gives

(2.3)

$$\kappa(A, B_{S(n_e)}) \leq \kappa(A, B_{C(n_e)})\kappa(B_{C(n_e)}, B_{S(n_e)}) = \kappa(A)\kappa(B_{S(n_e)}) = \frac{n_e\sigma_{\max}(A)}{\sigma_{\min}(A)}\,.$$

For small n_e, this may be a reasonable tradeoff. The bound (2.3) follows from the observation that the eigenvalues of $B_{S(n_e)}$ are exactly 0, 1, and n_e. We note that besides generating a sparser matrix this kind of approximation preserves structural properties such as planarity. This may be important for certain sparsification algorithms [5, 27] and subsequent factorization algorithms.

When A is ill conditioned, there may or may not be an SDD matrix B that approximates it well. The following two examples demonstrate both cases.

Example 2.13. Let

$$A = \frac{1}{2\epsilon}\begin{bmatrix} 1+\epsilon^2 & -\epsilon^2 & -1 \\ -\epsilon^2 & \epsilon^2 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

for some small $\epsilon > 0$. This matrix has rank 2 and null vector $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$, and its condition number is proportional to $1/\epsilon^2$. Since A is diagonally dominant, there is clearly an SDD matrix B (namely, A itself) such that $\kappa(A,B) = 1$. This matrix is the element matrix for a linear triangular element with nodes at $(0,0)$, $(0,\epsilon)$, and $(1,0)$ and a constant $\theta = 1$. The ill conditioning is caused by the high aspect ratio of the triangle, but this ill-conditioned matrix is still diagonally dominant. Small perturbations of the triangle will yield element matrices that are not diagonally dominant but are close to a diagonally dominant one. If we discretize the same triangle with the same material

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

constant using a quadratic element, the resulting element matrix is still approximable and ill-conditioned, but is far from SDD; it has positive off-diagonal with magnitude proportional to the norm of the element matrix [3].

Example 2.14. Our example of a matrix that cannot be approximated well by an SDD matrix is the element matrix for an isosceles triangle with two tiny angles and one that is almost π, with nodes at $(0,0)$, $(1,0)$, and $(1/2,\epsilon)$ for some small $\epsilon > 0$. The element matrix is

$$A = \frac{1}{2\epsilon}\begin{bmatrix} \frac{1}{4}+\epsilon^2 & \frac{1}{4}-\epsilon^2 & -\frac{1}{2} \\ \frac{1}{4}-\epsilon^2 & \frac{1}{4}+\epsilon^2 & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \end{bmatrix}.$$

This matrix has rank 2 and null vector $\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$. For any SDD matrix B with the same null space, $\kappa(A,B) \geq \epsilon^{-2}/4$ [3].

2.6. A heuristic for symmetric diagonally dominant approximations. In section 2.5 we have shown how to find a nearly optimal SDD approximation B to a symmetric positive (semi)definite matrix A whose null space is spanned by $\begin{bmatrix}1 & \cdots & 1\end{bmatrix}^T$. In this section we show a simple heuristic. We have found experimentally that it often works well. On the other hand, we can show that in some cases it produces approximations that are arbitrarily far from the optimal one. Thus, this section has two related goals: to describe a simple heuristic that often works well, but to point out that it cannot always replace the method of section 2.5.

DEFINITION 2.15. *Let A be a symmetric positive (semi)definite matrix. We define A_+ to be the SDD matrix defined by*

$$(A_+)_{ij} = \begin{cases} a_{ij} & i \neq j \text{ and } a_{ij} < 0 \\ 0 & i \neq j \text{ and } a_{ij} \geq 0 \\ \sum_{k \neq j} -(A_+)_{ik} & i = j\,. \end{cases}$$

Clearly, A_+ is SDD. It turns out that in many cases, $\kappa(A, A_+)$ is small, making A_+ a good approximation for A. The following lemma gives a theoretical bound for $\kappa(A, A_+)$. We omit its proof from space reasons.

LEMMA 2.16 ([3]). *Let A be an SPSD n_e-by-n_e matrix with null(A) = span$[1 \ldots 1]^T$. Then null(A_+) = span$[1 \ldots 1]^T$, and $\kappa(A, A_+) \leq \sqrt{n_e}\kappa(A)$. Moreover, if there exist a constant c and an index i such that $\|A\|_{\inf} \leq cA_{ii}$, then $\kappa(A, A_+) \leq c\kappa(A)$.*

This means that if A is well conditioned and small, then A_+ is always a fairly good approximation. The matrix A_+ is also sparser than A, and similarly to $B_{S(n)}$ its mesh will be planar if the mesh of A was planar.

But when A is ill conditioned, A_+ can be an arbitrarily bad approximation even though A is approximable by some other SDD matrix.

Example 2.17. Let $0 < \epsilon \ll 1$, and let $M \geq \frac{4}{\epsilon}$,

$$A = \begin{bmatrix} 1+M & -1 & 0 & -M \\ -1 & 1+M & -M & 0 \\ 0 & -M & M & 0 \\ -M & 0 & 0 & M \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1-\epsilon & -1+\epsilon \\ 0 & 0 & -1+\epsilon & 1-\epsilon \end{bmatrix}.$$

This matrix is symmetric semidefinite with rank 3 and null vector $\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}^T$. For small ϵ, A is ill conditioned, with condition number larger than $8\epsilon^{-2}$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

The matrix A_+ is a poor approximation of A,

$$\kappa(A, A_+) > \left(1 - \frac{1-\epsilon}{2M+1}\right)\epsilon^{-1} \approx \epsilon^{-1} .$$

Nevertheless, the matrix

$$B = \begin{bmatrix} \epsilon + M & -\epsilon & 0 & -M \\ -\epsilon & \epsilon + M & -M & 0 \\ 0 & -M & \epsilon + M & -\epsilon \\ -M & 0 & -\epsilon & \epsilon + M \end{bmatrix}$$

is a good approximation for A, with $\kappa(A, B) < 9$ for a small enough ϵ. We omit the proofs, which can be found in [3].

3. Scaling and assembling element-by-element approximations. Given a set of approximations $\{L_e\}$ to a set of element matrices $\{K_e\}$, our solver scales the L_e's so that their assembly $L = \sum_e \alpha_e L_e$ is a good approximation of $K = \sum_e K_e$. We can scale them in one of two equivalent ways. The next lemma shows that under these scalings, if every L_e is a good approximation of K_e, then L is a good approximation of K. Both scaling rules require the computation of one extreme eigenvalue for each pair (K_e, L_e).

LEMMA 3.1. *Let $\{K_e\}_{e\in E}$ and $\{L_e\}_{e\in E}$ be sets of symmetric positive (semi)definite matrices with null(K_e) = null(L_e). Let $\alpha_e = \min \Lambda(K_e, L_e)$ and let $\beta_e = \max \Lambda(K_e, L_e)$. Then*

$$\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) \le \max \kappa\left(K_e, L_e\right)$$
$$\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \beta_e L_e\right) \le \max \kappa\left(K_e, L_e\right)$$

Proof. Scaling L_e by α_e transforms the smallest generalized eigenvalue of $\Lambda(K_e, \alpha_e L_e)$ to 1, since

$$\Lambda(K_e, \alpha_e L_e) = \frac{1}{\alpha_e}\Lambda(K_e, L_e) .$$

Scaling L_e clearly does not change the generalized condition number, so $\max \Lambda(K_e, \alpha_e L_e) = \kappa(K_e, L_e)$.

By the splitting lemma [5],

$$\begin{aligned} \min \Lambda\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\ge \min\left\{\min \Lambda(K_e, \alpha_e L_e)\right\}_{e\in E} \\ &= \min\left\{1\right\}_{e\in E} \\ &= 1 . \end{aligned}$$

Also by the splitting lemma,

$$\begin{aligned} \max \Lambda\left(\sum_{e\in E} K_e, \sum_{e\in E} \alpha_e L_e\right) &\le \max\left\{\max \Lambda(K_e, \alpha_e L_e)\right\}_{e\in E} \\ &= \max\left\{\kappa\left(K_e, L_e\right)\right\}_{e\in E} . \end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Combining these two inequalities gives the result. The proof for scaling by β_e is the same. ∎

If we use inexact estimates $\tilde{\alpha_e}$ for the minimum of $\Lambda(K_e, L_e)$, the bound becomes

$$\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \tilde{\alpha_e} L_e\right) \leq \frac{\max_e \{(\alpha_e/\tilde{\alpha_e})\,\kappa\,(K_e, L_e)\}}{\min_e\,(\alpha_e/\tilde{\alpha_e})}\ ,$$

and similarly for estimates of β_e. This shows that $\kappa\left(\sum_{e\in E} K_e, \sum_{e\in E} \tilde{\alpha_e} L_e\right)$ depends on how much the estimates vary. In particular, if the relative estimation errors are close to 1, scaling by the estimates is almost as good as scaling by the exact eigenvalues.

4. Combinatorial sparsification of the assembled SDD approximation. Once we obtain an SDD approximation $L = \sum_e \alpha_e L_e$ of K, we can use a combinatorial graph algorithm to construct an easier-to-factor SDD approximation M of L. Because M is spectrally close to L and L is spectrally close to K, M is also spectrally close to K. By applying [9, Proposition 3.6] to both ends of the generalized spectra, we obtain the following lemma.

LEMMA 4.1. *Let K, L, and M be symmetric (semi)definite matrices with the same dimensions and the same null space. Then*

$$\kappa(K, M) \leq \kappa(K, L)\kappa(L, M)\ .$$

There are several algorithms that can build M from L. All of them view an exactly (but not strictly) SDD matrix L as a weighted undirected graph G_L, to which they build an approximation graph G_M. The approximation G_M is then interpreted as an SDD matrix M. If L is strictly diagonal dominant, the approximation starts from an exactly SDD matrix $L - D$, where D is diagonal with nonnegative elements. From $L - D$ the algorithm builds an approximation $\tilde{M}$; if $\tilde{M}$ is a good approximation to $L - D$, then $\tilde{M} + D$ is a good approximation to L.

LEMMA 4.2. *Let A and B be symmetric positive (semi)definite matrices with the same null space $\mathbb{S}$, and let C be a positive symmetric (semi)definite with a null space that is a subspace, possibly empty, of $\mathbb{S}$. Then*

$$\Lambda(A + C, B + C) \subseteq [\min \Lambda(A, B) \cup \{1\}, \max \Lambda(A, B) \cup \{1\}]\,.$$

Proof. The result is a simple application of the splitting lemma [5], since

$$\begin{aligned}\max \Lambda(A + C, B + C) &\leq \max\{\max \Lambda(A, B), \max \Lambda(C, C)\}\\ &= \max\{\max \Lambda(A, B), 1\}\\ &= \max \Lambda(A, B) \cup \{1\}\ ,\end{aligned}$$

and similarly for the smallest generalized eigenvalue. ∎

This lemma is helpful, since most of the algorithms that construct approximations of SDD matrices provide theoretical bounds on $\Lambda(L - D, \tilde{M})$ that have 1 as either an upper bound or a lower bound. When this is the case, adding D to $L - D$ and to $\tilde{M}$ preserves the theoretical condition-number bound.

The earliest algorithm for this subproblem is Vaidya's algorithm [46, 5, 15]. This algorithm finds a maximum spanning tree in G_M and augments it with suitable edges. The quantity of extra edges that are added to the tree is a parameter in this algorithm. When few or no edges are added, $\kappa(L, M)$ is high (but bounded by $nm/2$, where m is

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

the number of off-diagonal elements in L), but L is cheap to factor (can be factored in $O(n)$ operations when no edges are added to the tree). When many edges are added, $\kappa(L, M)$ shrinks but the cost of factoring L rises. When G_M is planar or nearly planar, then the algorithm is very effective, both theoretically and experimentally. For more general classes of graphs, even with a bounded degree, the algorithm is less effective.

A heuristic for adding edges to the maximum spanning tree was proposed by Frangioni and Gentile [19]. They designed their algorithm for SDD linear systems that arise in the interior-point solution of network-flow problems. There are no theoretical convergence bounds for this algorithm.

Several algorithms are based on so-called *low-stretch spanning trees*. Boman and Hendrickson realized that a low-stretch spanning tree G_M of G_L leads to a better convergence-rate bound than maximum spanning trees [8]. Elkin et al. showed simpler and lower-stretch constructions for low-stretch trees [18]. Spielman and Teng proposed algorithms that create denser graphs G_M (which are still based on low-stretch trees) [43, 44, 45]. The algorithms of Spielman and Teng lead to nearly linear work bound for solving $Lx = b$.

There are other classes of combinatorial preconditioners, for example [21, 20, 29]. It is not clear whether they can be used effectively in our framework.

5. Dealing with inapproximable elements. When some of the element matrices cannot be approximated well by an SDD matrix, we split the global stiffness matrix K into $K = K_{\le t} + K_{>t}$, where $K_{\le t} = \sum_{e \in E(t)} K_e$ is a sum of the element matrices for which we found an SDD approximation L_e that satisfies $\kappa(K_e, L_e) \le t$ for some threshold $t > 0$, and $K_{>t} = \sum_{e \notin E(t)} K_e$ is a sum of element matrices for which our approximation L_e gives $\kappa(K_e, L_e) > t$.

We then scale the approximations in $E(t)$ and assemble them to form $L_{\le t} = \sum_{e \in E(t)} \alpha_e K_e$. We now apply one of the combinatorial graph algorithms discussed in section 4 to construct an approximation $M_{\le t}$ to $L_{\le t}$. Once we have $M_{\le t}$, we add it to $K_{>t}$ to obtain a preconditioner $M_1 = M_{\le t} + K_{>t}$.

This construction gives a bound on $\kappa(K, M_1)$, but it is a heuristic in the sense that M_1 may be expensive to factor. The analysis of $\kappa(K, M_1)$ is essentially the same as the analysis of strictly dominant matrices in section 4: by Lemma 4.2, a theoretical bound $\Lambda(K_{\le t}, M_{\le t}) \subseteq [\alpha, \beta]$ implies $\Lambda(K, M_1) \subseteq [\min\{\alpha, 1\}, \max\{\beta, 1\}]$.

The scaling technique of Lemma 3.1 ensures that either $\alpha, \beta \le 1$ or $1 \le \alpha, \beta$. But the interval $[\alpha, \beta]$ may be quite far from 1. If the interval is far from 1, the bound on $\kappa(K, M_1)$ can be considerably larger than the bound on $\kappa(K_{\le t}, M_{\le t})$. We observed this behavior in practice (experiments not reported here). To avoid this danger, we scale $M_{\le t}$ before adding it to $K_{>t}$; that is, we use a preconditioner $M_\gamma = \gamma M_{\le t} + K_{>t}$. We choose γ as follows. We find some vector v that is orthogonal to $\text{null}(M_{\le t})$ and compute its generalized Raleigh quotient

$$\gamma = \frac{v^T K_{\le t} v}{v^T M_{\le t} v} .$$

The null space of $M_{\le t}$ is determined by the connected components of its graph, so it is easy to quickly find such a v (we use a random v in this subspace). This definition ensures that $\gamma \in [\alpha, \beta]$. Since $\Lambda(K_{\le t}, \gamma M_{\le t}) \subseteq [\alpha/\gamma, \beta/\gamma]$, we have $1 \in [\alpha/\gamma, \beta/\gamma]$.

LEMMA 5.1. *Under this definition of M_γ, $\kappa(K, M_\gamma) \le \beta/\alpha$, where the interval $[\alpha, \beta]$ bounds $\Lambda(K_{\le t}, M_{\le t})$. In particular, if we take α and β to be the extremal generalized eigenvalues of $(K_{\le t}, M_{\le t})$, we obtain*

$$\kappa(K, M_\gamma) \le \kappa(K_{\le t}, M_{\le t}) .$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

We expect that this overall heuristic will be effective when $E \setminus E(t)$ contains only few elements. If only a few elements cannot be approximated, then $K_{>t}$ is very sparse, so the sparsity pattern of $M_\gamma = \gamma M_{\leq t} + K_{>t}$ is similar to that of $M_{\leq t}$. Since $M_{\leq t}$ was constructed so as to ensure that its sparsity pattern allow it to be factored without much fill, we can expect the same to hold for M_γ. If $E \setminus E(t)$ contains many elements, there is little reason to hope that the triangular factor of M_γ will be particularly sparse.

If $E \setminus E(t)$ contains very few elements a different strategy can be used. Instead of introducing the ill-conditioned elements into M_γ they can be ignored. Each ignored element can be considered as an $n_e - 1$ rank perturbation. Results in [2] suggest that this will increase the number of iterations by at most $n_e - 1$. Therefore, if the number of inapproximable elements is small only a few more iterations will be needed. If many inapproximable elements are dropped, convergence can be slow.

6. Asymptotic complexity issues. In this section we explain the asymptotic complexity of the different parts of our solver. We do not give a single asymptotic expression that bounds the work that the solver performs, but comment on the cost and asymptotics of each phase of the solver. The cost of some of the phases is hard to fully analyze, especially when $E(t) \subsetneq E$. The next section presents experimental results that complement the discussion here.

The first action of the solver is to approximate each element matrix K_e by an SDD matrix $\alpha_e L_e$. For a given element type, this phase scales linearly with the number of elements and it parallelizes perfectly. The per-element cost of this phase depends on the approximation method and on the number n_e of degrees of freedom per element. Asymptotically, all the approximation methods require n_e^3 operations per element, but the uniform-clique is the fastest method. This phase also gives us $\kappa(K_e, \alpha_e L_e)$ which we use to decide which elements belong to $E(t)$ and which do not.

The next phase of the solver assembles the scaled SDD approximations in $E(t)$. The cost of this step is bounded by the cost to assemble $K = \sum_e K_e$, which most finite-elements solvers (including ours) perform. The assemblies of K and L perform $O(\sum_e n_e^2)$ operations: fewer than the first phase, but harder to parallelize.

The cost and the asymptotic complexity of the sparsification of L depends on the algorithm that is used. For Vaidya's sparsification algorithm, which our code uses, the amount of work is $O(n \log n + \sum_e n_e^2)$. For the algorithm of Spielman and Teng [43, 44, 45], the work is $O(m^{1.31})$ where $m = \sum_e n_e^2$.

Next, the algorithm assembles the element matrices K_e that are not in $E(t)$ into $M_{\leq t}$. The cost of this phase is also dominated by the cost of assembling K.

The cost of computing the Cholesky factorization of M is hard to characterize theoretically, because the cost depends on the nonzero pattern of M in a complex way. The nonzero pattern of M depends on how many and which elements are not in $E(t)$, and on how much we sparsified $L_{\leq t}$. The number of operations in a phase of the solver is not the only determinant of running time, but also the computational rate. The Cholesky factorization of M usually achieves high computational rates.

The cost of the iterative solver depends on the number of iterations and on the per-iteration cost. The number of iterations is proportional to $\sqrt{\kappa(K, M_\gamma)} \leq t\sqrt{\kappa(L, M_{\leq t})}$. The amount of work per iteration is proportional to the number of nonzeros in K plus the number of nonzeros in the Cholesky factor of M. The sparsification algorithms of Vaidya and Spielman and Teng control the number of iterations, and if $E(t) = E$, then they also control the density of the Cholesky factor.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

7. Experimental results. This section presents experimental results that explore the effectiveness of our solver.

7.1. Setup. Our solver currently runs under MATLAB [31], but it is implemented almost entirely in C. The C code is called from MATLAB using MATLAB's CMEX interface. The element-by-element approximations are computed by C code that calls LAPACK [1]. The assembly of the element-by-element approximations (and possibly the inapproximable elements) is also done in C. The construction of Vaidya's preconditioners for SDD matrices is done by C code [15]. The Cholesky factorization of the preconditioner is computed by MATLAB's sparse `chol` function, which in MATLAB 7.2 calls CHOLMOD 1.0 by Tim Davis. We always order matrices using METIS version 4.0 [26] prior to factoring them. The iterative Krylov-space solver that we use is a preconditioned conjugate gradient written in C and based on MATLAB's `pcg`.; within this iterative solver, both matrix-vector multiplications and solution of triangular linear systems are performed by C code.

In most experiments we compare our solver to an algebraic multigrid solver, BoomerAMG [23]. We use the version of BoomerAMG that is packaged as part of HYPRE 1.2. We compiled it using GCC version 3.3.5, with options `-O` (this option is HYPRE's default compilation option). We note that BoomerAMG is purely algebraic and does not exploit the element-by-element information. There exist variations of algebraic multigrid that do exploit the element structure [13, 14]. We have not experimented with these variants. Our comparison with BoomerAMG is meant mainly to establish a credible baseline for the results and not to explore the behavior of algebraic multigrid solvers.

In some experiments we compare our solver to solvers that are based on incomplete Cholesky preconditioners [30, 32, 39, 48]. To compute these preconditioners, we use MATLAB's built-in `cholinc` routine. Here too, the matrices are preordered using METIS.

Since many of our test problems are ill conditioned, we iterate until the relative residual is at most 10^{-14}, close to $\epsilon_{\text{machine}}$, in order to achieve acceptable accuracy.

We use two mesh generators to partition the three-dimensional problem domains into finite elements. We usually use TETGEN version 1.4 [42]. In a few experiments we use DISTMESH [34], which can generate both two- and three-dimensional meshes.

Running times were measured on a 1.6 GHz AMD Opteron 242 computer with 8 GB of main memory, running Linux 2.6. This computer has two processors, but our solver only uses one. We used a 64-bit version of MATLAB 7.2. This version of MATLAB uses the vendor's BLAS, a library called ACML. The measured running times are wall-clock times that were measured using the `ftime` Linux system call.

7.2. Test problems. We evaluated our solver on several three-dimensional problems. We used both linear and quadratic tetrahedral elements. Table 7.1 summarizes the problems that we used in the experiments. The boundary conditions are always pure Neumann $\partial u/\partial n = 0$, and we removed the singularity by fixing the solution at a fixed unknown (algebraically, we remove the row and column of K corresponding to that unknown). We generate the right-hand side b of the linear system $Kx = b$ by generating a random solution vector x and multiplying it by K to form b.

In all the experiments reported below, except for a single experiment, our solver produced acceptable forward errors. The computed solution $\hat{x}$ satisfied

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \leq 10^{-4} .$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 7.1
Notation for the test problems.

The domain Ω	
C	A 3-dimensional cube
B	A 3-dimensional box with aspect ratio 1-by-1-by-10000
CH	A 1-by-1-by-1 cube with a 1-by-0.1-by-0.79 hole in the middle
SC	A 10-by-10-by-10 cube containing a spherical shell of inner radius 3 and thickness 0.1.
The mesh (the parameter indicates the number n *of mesh points)*	
G	3-dimensional, generated by TETGEN
D	3-dimensional, generated by DISTMESH
The conductivity $\theta(x)$	
U	uniform and isotropic, $\theta = I$ everywhere
J	jump between subdomains but uniform and isotropic within subdomain (e.g., $\theta = 10^4 I$ in the spherical shell of domain SC and $\Theta = I$ elsewhere); the parameter indicates the magnitude of the jump
A	anisotropic within a subdomain (e.g., the spherical shell in SC) and $\theta = I$ elsewhere; θ is always 1 in the x and y directions and the parameter indicates the conductivity in the z direction.
The element type	
L	Linear tetrahedral element, 4-by-4 element matrix
Q	Quadratic tetrahedral element, 10-by-10 element matrix

In one of the experiments with well-conditioned elements and jumping coefficients with ratio 10^8 (section 7.5), when running Vaidya's preconditioner with the goal 0.6, the forward error was $1.24 \cdot 10^{-3}$.

7.3. Choosing the parameter t. We begin with simple problems that are designed to help us choose t, the approximation threshold. The behavior of our solver depends on two parameters, t and the aggressiveness of the combinatorial sparsification algorithm. These parameters interact in complex ways, because both influence the sparsity of the Cholesky factor of M and the number of iterations in the Krylov-space solver. It is hard to visualize and understand the performance of a solver in a two-dimensional (or higher) parameter space. Therefore, we begin with experiments whose goal is to establish a reasonable value for t, a value that we use in all of the subsequent experiments.

Figure 7.1 shows the results of these experiment, which were carried out on two meshes, one generated by DISTMESH and the other by TETGEN. The elements are all linear tetrahedral, and their approximations L_e are built using our nearly optimal approximation algorithm. The graphs on the left show the distributions of $\kappa(K_e)$. With DISTMESH, we see that the elements belong to two main groups, a large group of elements with generalized condition numbers smaller than about 100, and a small set of so-called slivers with much higher condition numbers, ranging from 200 to 10^8. From results not shown here, it appears that for the nonslivers, $\kappa(K_e, L_e)$ is smaller than $\kappa(K_e)$ by roughly a constant factor. For the slivers, $\kappa(K_e, L_e)$ is close to $\kappa(K_e)$. With TETGEN, there are no highly ill-conditioned elements, and the distributions of $\kappa(K_e)$ and $\kappa(K_e, L_e)$ are smoother.

The graphs on the right show the number of iterations that the conjugate gradients algorithm performs for several values of t and various levels of sparsification. In all the graphs in the paper whose horizontal axis is fill in the Cholesky factor, the horizontal axis ranges from 0 to the number of nonzeros in the Cholesky factor of K. When t is small, $K_{>t}$ is relatively dense, so the sparsification algorithm cannot be very effective. Even when we instruct Vaidya's algorithm to sparsify $L_{\le t}$ as much as possible, the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

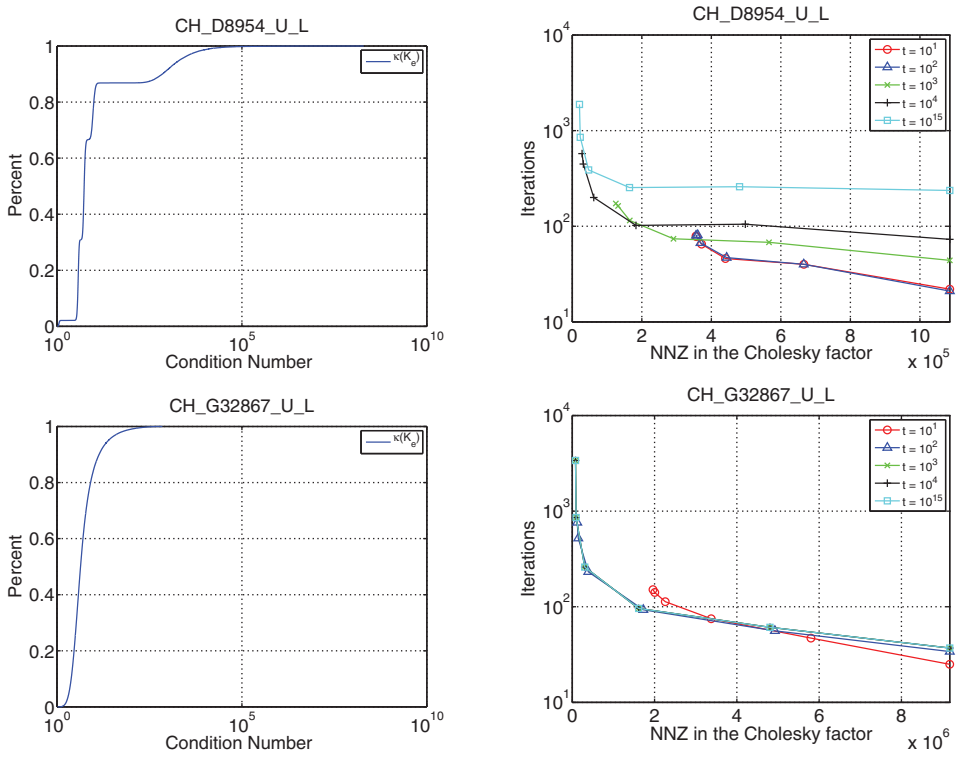


FIG. 7.1. *The distribution of element condition numbers (left graphs) and the number of iterations for several values of t and several sparsification levels (right graphs). The top row shows results on a mesh generated by* DISTMESH, *and the bottom row on a mesh generated by* TETGEN.

Cholesky factor of M remains fairly similar to the Cholesky factor of K. On the other hand, a small t leads to faster convergence. With a large t we can construct M's with very sparse factors, but convergence is very slow. If all the elements are relatively well-conditioned, then there is little dependence on t, as can be seen in the bottom right figure. A value of t near 1000 gives a good balance between high iteration counts caused by using L_e's with fairly high $\kappa(K_e, L_e)$ and the inability to construct a sparse preconditioner caused by a dense $K_{>t}$. We use the fixed value $t = 1000$ in the remaining experiments in order to clarify the role of the other parameter in the algorithm.

We stress that the selection of t in practice should not be static; it should be based on the actual distribution of the generalized condition numbers of the approximation and on analysis similar to the one described in this section.

7.4. Baseline tests. The next set of experiments shows the performance of our solver relative to other solvers on the same problems and the same meshes, for a few relatively easy problems. The graphs in Figure 7.2 compare the running time of our solver to that of an incomplete Cholesky preconditioner, BoomerAMG, and a state-of-the-art direct solver, CHOLMOD. In these graphs the vertical axis represents wall-clock time for all the phases of the solution and the horizontal axis represents the number of nonzeros in the triangular factors of the preconditioner. The rightmost (largest) horizontal coordinate in the graphs always corresponds to the number of nonzeros in a complete sparse Cholesky factor of the coefficient matrix. When the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

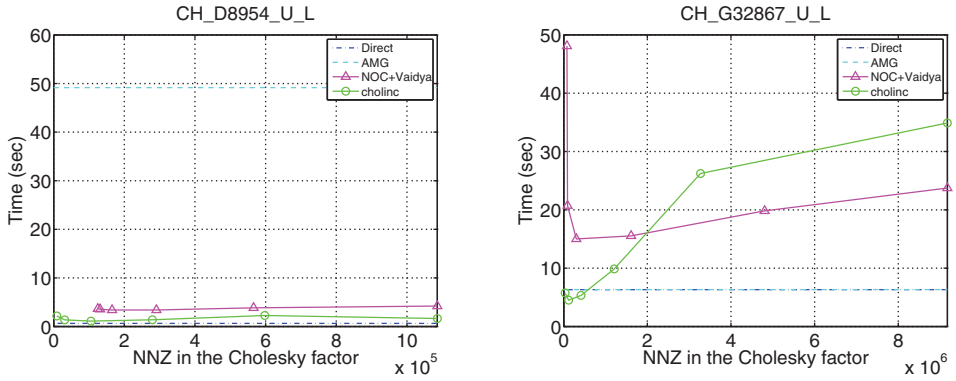


FIG. 7.2. *Running times for our solver, for incomplete Cholesky, for BoomerAMG, and for a direct solver on simple three-dimensional problems. The graph on the left uses a mesh generated by* DISTMESH, *and the one on the right a mesh generated by* TETGEN. *See the first paragraph of section* 7.4 *for a complete explanation of the graphs.*

complete factorization runs out of space, we still use this scaling of the horizontal axis, and we estimate the running time of the complete factorization based on the assumptions that it runs at 10^9 floating-point operations per second. The direct solver and BoomerAMG only give us one data point for each problem; their running times are represented in the graphs by horizontal lines. We ran each preconditioned solver with several values of the parameter that controls the sparsity of the factor (drop tolerance in incomplete Cholesky and the sparsification parameter in Vaidya's preconditioner). Therefore, for each preconditioned solver we have several data points that are represented by markers connected by lines. Missing markers and broken lines indicate failures to converge within a reasonable amount of time. Our algorithm is labeled as "NOC+Vaidya". NOC stands for "nearly optimal clique" because our algorithm uses a clique topology for the approximation. Most of the remaining graphs in this section share the same design.

The graphs in Figure 7.2 compare the running time of the solvers on easy problems with a relatively simple domain and uniform coefficients. The mesh produced by TETGEN leads to a linear system that is easy for all the iterative solvers. With a good drop-tolerance parameter, incomplete Cholesky is the fastest, with little fill. Our solver is slower than all the rest, even with the best sparsity parameter. The mesh produced by DISTMESH causes problems to BoomerAMG, but incomplete Cholesky is faster than our solver.

Although the performance of incomplete Cholesky appears to be good in the experiments reported in Figure 7.2, it sometimes performs poorly even on fairly simple problems. Figure 7.3 shows that on a high-aspect-ratio three-dimensional structure with uniform coefficients, incomplete Cholesky performs poorly: the sparser the preconditioner, the slower the solver. Our solver, on the other hand, performs reasonably well even when its factor is much sparser than the complete factor. On the high-aspect-ratio problem, as well as on any problem of small to moderate size, the direct solver performs well. But as the problem size grows the direct solver becomes slow and tends to run out of memory. The rightmost graph in Figure 7.3 shows a typical example.

Figure 7.4 shows a breakdown of the running time of our solver for one particular problem. The data shows that as the preconditioner gets sparse, the time to factor the preconditioner decreases. The running time of the iterative part of the solver also initially decreases, because the preconditioner gets sparser. This more than offsets

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

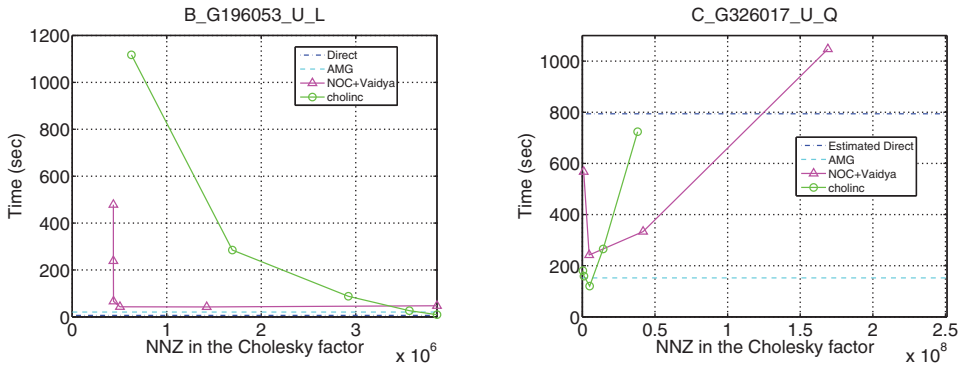
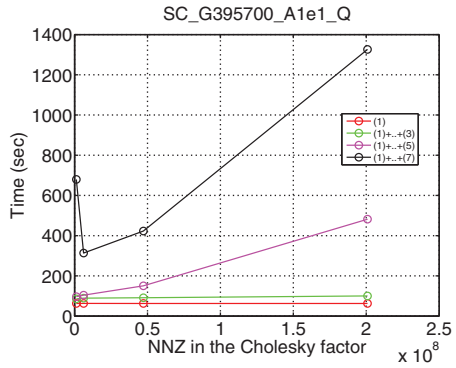


FIG. 7.3. *Experimental results on two additional problems with uniform coefficients; these problems are much larger than those analyzed in Figure* 7.2.



Notation for phases of the solver:
(1) Approximate K_e by L_e
(2) Assembly $L_{\leq t} = \sum_{E(t)} L_e$
(3) Sparsify $L_{\leq t}$ to obtain $M_{\leq t}$
(4) Assembly of $M = M_{\leq t} + K_{>t}$
(5) Order, permute, and factor M
(6) Assembly of $K = \sum_E K_e$
(7) Permute K and iterate

FIG. 7.4. *A breakdown of the running time of our solver, on a particular problem. The graph shows the time consumed by the different phases of the solver. Assembly phases are not separately shown because their running time is negligible.*

the growth in the number of iterations. But when the preconditioner becomes very sparse, it becomes less effective, and the number of iterations rises quickly.

7.5. Well-conditioned elements and jumping coefficients. The next set of experiments explores problems with a large jump in the conductivity θ. We instructed the mesh generators to align the jump with element boundaries, so within each element, there is no jump. This leads to a large $\kappa(K)$, but the conditioning of individual element matrices is determined by their geometry, not by θ. The results, shown in Figure 7.5, show that the jump in θ does not influence any of the four solvers in a significant way.

7.6. Ill-conditioned elements: Anisotropy. Some of the experiments shown in section 7.3 included ill-conditioned elements. The ill-conditioning of those elements resulted from their geometrical shape. Other mesh generators may be able to avoid such element shapes. Indeed, TETGEN did not produce such elements, only DISTMESH did. But in problems that contain anisotropic materials in complex geometries, ill-conditioned elements are hard to avoid.

Figure 7.6 compares the performance of our solver with that of other solvers on a problem in which the conductivity θ is anisotropic in one part of the domain. The

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

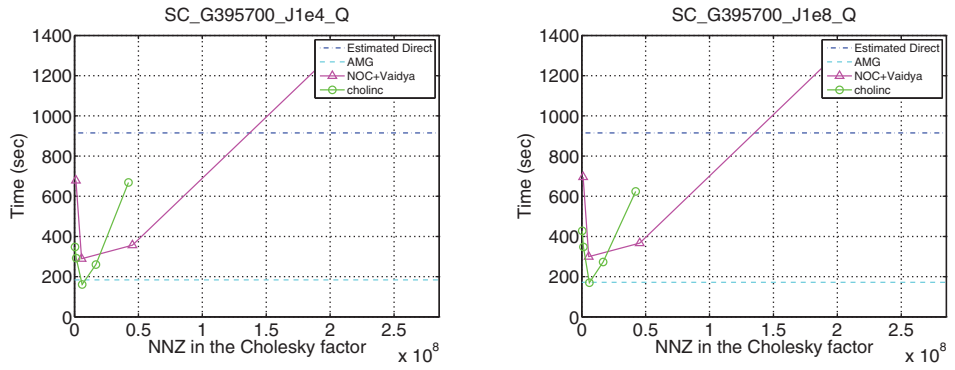


FIG. 7.5. *Running times for problems with jumping coefficients. The average of the ratio* $\kappa(K_e)/\kappa(L_e, K_e)$ *in both experiments is around* 3.6 *and is nearly constant. The condition numbers of the* K_e*'s are always small.*

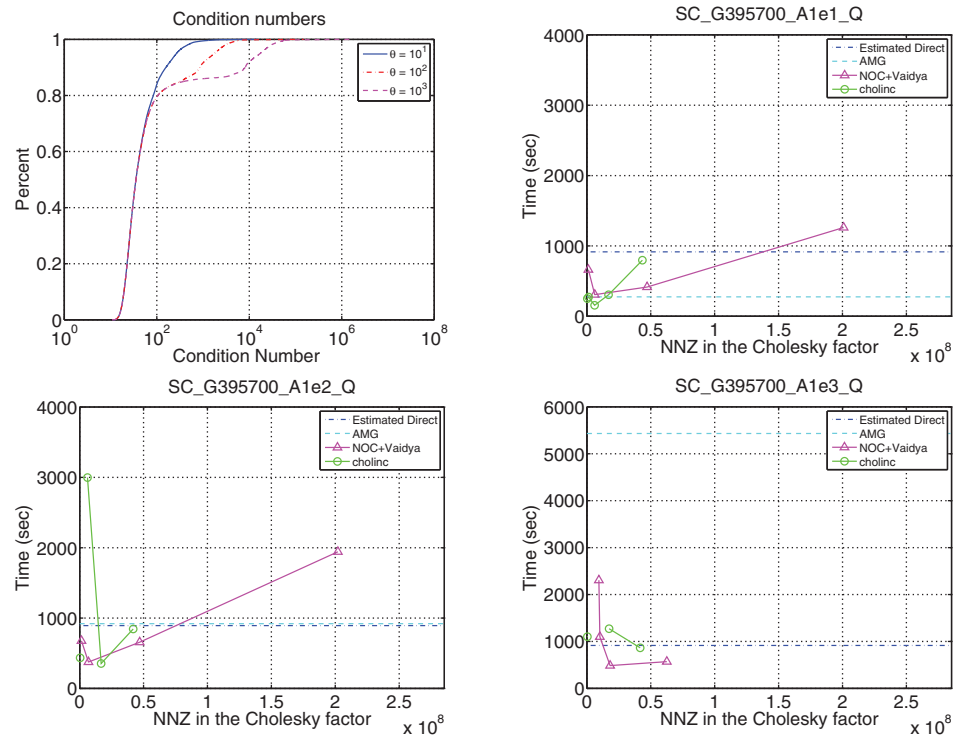


FIG. 7.6. *The behavior of our solver and other solvers on a solid three-dimensional problem that contains a thin spherical shell with anisotropic material. The mesh was generated by* TETGEN. *We report here three experiments with anisotropy rates* 10, 10^2, *and* 10^3 *(*θ *in the legend). The top left graph shows the distribution of the element matrix condition numbers in the different experiments. The ratio* $\kappa(K_e)/\kappa(L_e, K_e)$ *is nearly constant within every experiment. This ratio for the isotropic elements is similar in all the experiments. For the anisotropic elements, the maximal ratio grows from about* 19 *for anisotropy* 10 *to about* 806 *for anisotropy* 10^3.

results clearly show that anisotropy leads to ill-conditioned element matrices. As the anisotropy increases, BoomerAMG becomes slower and incomplete Cholesky becomes less reliable. The anisotropy does not have a significant influence on our solver. In

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

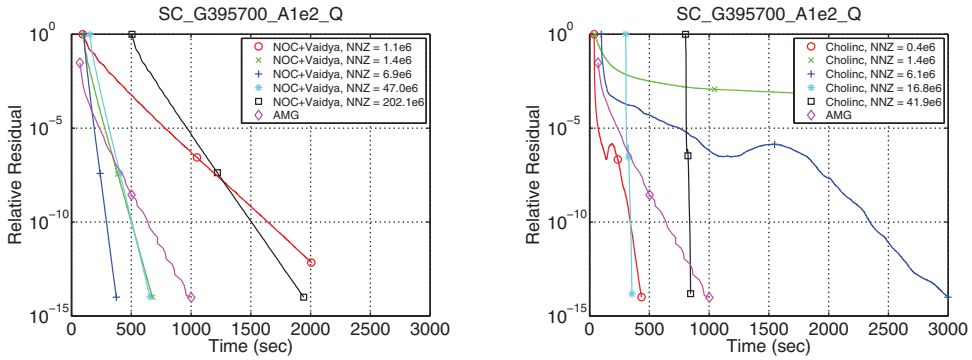


FIG. 7.7. *The relative norm of the residual as a function of the running time (and implicitly, of the iteration count) on one anisotropic problem, for our solver (left) and for incomplete Cholesky (right). The horizontal coordinate in which individual plots start indicates the time to construct and factor the preconditioner.*

experiments not reported here, our solver behaved well even with anisotropy of 10^8. The incomplete-factorization solver becomes not only slow, but also erratic, as the graphs in Figure 7.7 show. The convergence of our preconditioner is always steady, monotonically decreasing, and the convergence rate is monotonic in the density of the preconditioner. The convergence of incomplete Cholesky is erratic, not always monotonic, and sometimes very slow. Furthermore, sometimes one incomplete factor leads to much faster convergence than a much denser incomplete factor. We acknowledge that in some cases, when targeting larger relative residuals (like 10^{-2} or 10^{-5}), incomplete factorization and multigrid preconditioners are more effective than combinatorial preconditioners. This is evident in other combinatorial preconditioners [15, 41]. This is clearly shown in Figure 7.7.

These results show that the ability of our solver to detect inapproximable elements and to treat them separately allows it to solve problems that cause difficulty to other iterative solvers. When there are few such elements, as there are here because the anisotropic shell is thin, these elements do not significantly increase the density of the factor of M. In problems in which most of the elements are inapproximable by SDD matrices, M would be similar to K and the characteristics of our solver would be similar to the characteristics of a direct solver.

This is perhaps the most important insight about our solver. As problems get harder (in the sense that more elements become inapproximable), its behavior becomes closer to that of a direct solver. As problems get harder we lose the ability to effectively sparsify the preconditioner prior to factoring it. But unlike other solvers, our solver does not exhibit slow or failed convergence on these difficult problems.

7.7. Comparisons of different element-by-element approximations. We now explore additional heuristics for approximating K_e. The approximation methods that we compare are as follows:

Nearly Optimal Clique (NOC) $L_e = ZDD^TZ$, where the columns of Z are the full set of edge vectors and D scales the columns of U^+Z to unit 2-norm. This method gives the strongest theoretical bound of the methods we tested on $\kappa(K_e, L_e)$: it is at most $n_e^2/2$ times larger than the best possible for an SDD approximation of K_e. Here and in the next four methods, we set the scaling factor α_e to be $\max \Lambda(K_e, L_e)$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

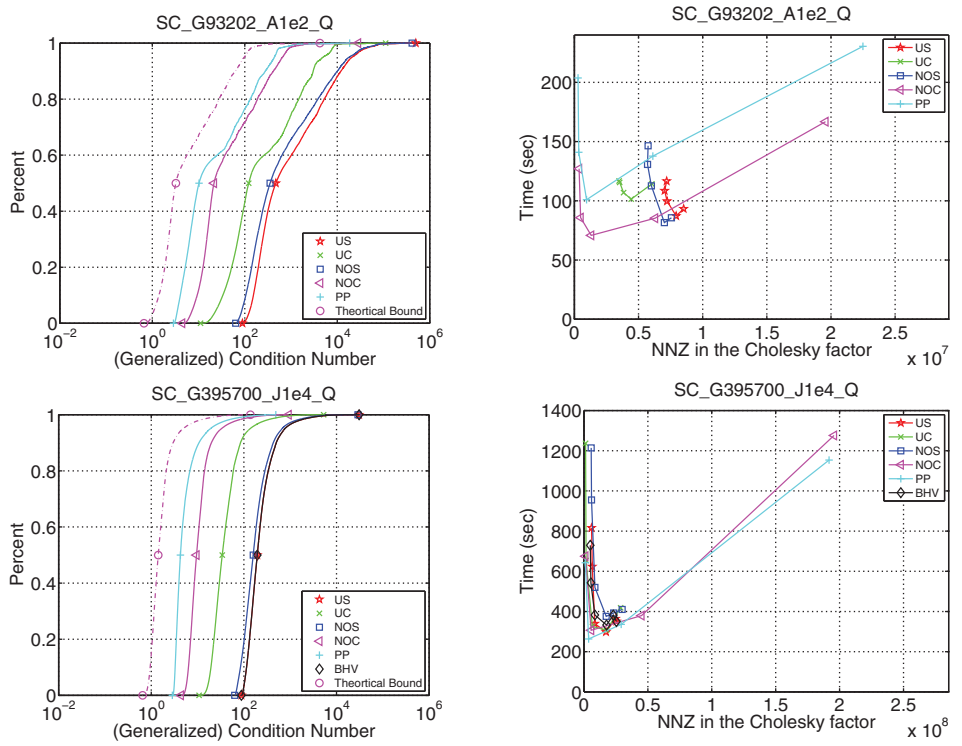


FIG. 7.8. *Different element-approximation methods. The graph on the left shows the distribution of $\kappa(K_e, L_e)$ for each approximation method. The method of Boman et al. is only applicable to well-conditioned elements, so it is not included in the graphs in the top row. The theoretical bound in the left figure is calculated using the fact that the spectral distance of NOC is within $n_e^2/2$ of the best possible. In the bottom left figure, the BHV (diamond markers) plot occludes the US (star markers) plot.*

Nearly Optimal Star (NOS) $L_e = ZDD^TZ$, where the columns of Z are edge vectors that form a star, $\langle 1, -2\rangle, \langle 1, -3\rangle, \ldots, \langle 1, -n_e\rangle$ and D scales the columns of U^+Z to unit 2-norm. Sparser than the first but usually a worse approximation.

Uniform Clique (UC) L_e is the extension of the n_e-by-n_e matrix $B_{C(n_e)}$ to an n-by-n matrix. Computing L_e is cheap, but the approximation is only guaranteed to be good when K_e is very well conditioned. The low cost of this method stems from the facts that (1) $B_{C(n_e)}$ is a fixed matrix, and (2) $\alpha_n = \max \Lambda(K_e)$, so we do not need to estimate an extreme generalized eigenvalue, only a single-matrix eigenvalue.

Uniform Star (US) L_e is the extension of the n_e-by-n_e matrix $B_{S(n_e)}$ to an n-by-n matrix. Sparser than the uniform clique, but more expensive, since we compute an extreme generalized eigenvalue to set α_e.

Positive Part (PP) $L_e = (K_e)_+$, defined in section 2.6.

Boman et al. (BHV) $\alpha_e L_e$ is the Boman–Hendrickson–Vavasis approximation of K_e [10]. In their method, L_e is a uniform star, and the scaling factor α_e is computed from quantities associated with the finite-element discretization that produces K_e.

The results are shown in Figure 7.8. When element matrices are fairly well conditioned (bottom graphs), different approximation methods exhibit similar qualitative

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

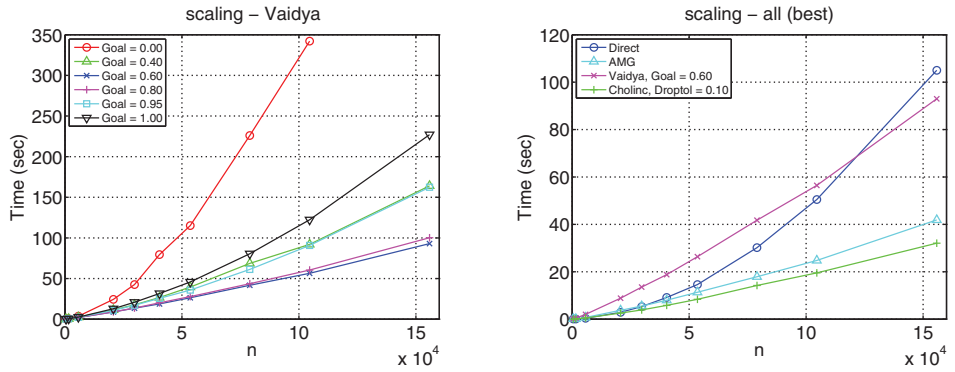


FIG. 7.9. *Solution times as a function of mesh size, on the same physical problem (a cube with uniform coefficients, discretized using linear tetrahedral elements). The graph on the left compares the running times of our solver with different levels of fill, and the graph on the right compares our solver (with the best-case fill level) with BoomerAMG and the direct solver. The fill in our solver is controlled by a parameter called goal in these graphs. A goal of* 1 *does not sparsify the approximation* $M_{\leq t}$*, and a goal of* 0 *sparsifies it as much as possible, resulting in a tree or forest graph structure for* $L_{\leq t}$*.*

behaviors. But when there are ill-conditioned elements, naive approximation methods perform poorly. The results also show that the nearly optimal approximation (which we used in all the other experiments) performs well relative to other approximation methods, but is usually not the best.

The comparision between PP and NOC in the top experiments in Figure 7.8 is interesting. The top left graph indicates that PP is a better approximation in that experiment. This was consistent in additional parameters that we explored and are not reported here: absolute condition number, number of elements that were inapproximable (beyond $t = 1000$), the distribution of the ratio $\kappa(K_e, L_e)/\kappa(K_e)$, sparsity of the preconditioner, and time to compute the approximation.

Nevertheless, the number of iterations with the NOC+Vaidya preconditioner is substantially smaller (about half) of the number of iterations with the PP+Vaidya preconditioner. The bottom line is that NOC+Vaidya performs better than PP+Vaidya. We do not have a good explanation for this; it may be a good subject for future research.

7.8. Running times for growing problem sizes. The results in Figure 7.9 present the growth in running time of our solver as the problem size grows. For very dense and very sparse preconditioners, the growth is highly nonlinear. This is consistent with the theory of sparse direct solvers on one side and with the theory of Vaidya's preconditioners on the other. For intermediate levels of fill, running times grow more slowly, but they still seem to grow superlinearly.

8. Open problems. This paper raises several interesting questions and challenges for further research. We mention three. One challenge is to extend the optimal-scaling method of Braatz and Morari [12] to rank-deficient and rectangular matrices. It may be possible to use the reduction in [49] to solve this problem, but we have not explored this. It is not even clear whether van der Sluis's nearly optimal scaling for rectangular matrices [47] is also nearly optimal for rank-deficient matrices. Another interesting question is to find a reliable and cheap-to-compute estimate of the spectral distance between a given symmetric positive (semi)definite matrix A and the closest

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

SDD matrix to A. Our method can be used to estimate this distance, but for large matrices the method is expensive and its estimates are loose. We have also shown that $\kappa(A)$ is an upper bound on that distance, but this bound can be arbitrarily loose. The third and probably most important challenge is to find better ways to exploit the splitting $K = K_{\leq t} + K_{>t}$. There may be several ways to exploit it. For example, it is probably possible to build better preconditioners by sparsifying $L_{\leq t}$ with the objective to reduce fill in the Cholesky factor of $M_{\leq t} + K_{>t}$; the algorithm that we used for the sparsification phase ignores $K_{>t}$ and only tries to reduce fill in the factor of $M_{\leq t}$.

Acknowledgments. Thanks to the two anonymous referees and the editor for their numerous comments and suggestions. We specifically want to thank one of the referees for suggesting an elegant proof for Lemma 2.5 and for helping us to improve Lemma 2.16. We also want to thank Stephen Vavasis for [49].

Sivan Toledo's work on this problem started a decade ago, in 1996, when he was working with John Gilbert under DARPA contract DABT63-95-C-0087, "Portable parallel preconditioning". The contributions and support of John and DARPA are gratefully acknowledged.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, 3rd ed., SIAM, Philadelphia, 1999. Also available online from http://www.netlib.org.

[2] H. Avron, E. Ng, and S. Toledo, *Using perturbed QR factorizations to solve linear least-squares problems*, SIAM J. Sci. Comput., submitted.

[3] H. Avron, G. Shklarski, and S. Toledo, *On element SDD approximability*, Technical report, Tel-Aviv University, Israel, Apr. 2008, http://www.cs.tau.ac.il/∼haima/element_approximability.pdf.

[4] F. L. Bauer, *Optimally scaled matrices*, Numer. Math., 5 (1963), pp. 73–87.

[5] M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, *Support-graph preconditioners*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 930–951.

[6] D. S. Bernstein, *Matrix Mathematics: Theory, Facts, and Formulas with Applications to Linear Systems Theory*, Princeton University Press, Princeton, NJ, 2005.

[7] E. G. Boman, D. Chen, B. Hendrickson, and S. Toledo, *Maximum-weight-basis preconditioners*, Numer. Linear Algebra Appl., 11 (2004), pp. 695–721.

[8] E. G. Boman and B. Hendrickson, *On spanning tree preconditioners*, unpublished manuscript, Sandia National Laboratories, 2001.

[9] E. G. Boman and B. Hendrickson, *Support theory for preconditioning*, SIAM J. Matrix Anal. Appl., 25 (2004), pp. 694–717.

[10] E. G. Boman, B. Hendrickson, and S. A. Vavasis, *Solving elliptic finite element systems in near-linear time with support preconditioners*, SIAM J. Numer. Anal., 46 (2008), pp. 3264–3284.

[11] R. D. Braatz, *Response to Chen and Toledo on "minimizing the Euclidean condition number"*, private communication, Sept. 2005.

[12] R. D. Braatz and M. Morari, *Minimizing the Euclidean condition number*, SIAM J. Control Optim., 32 (1994), pp. 1763–1768.

[13] 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.

[14] 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.

[15] D. Chen and S. Toledo, *Vaidya's preconditioners: Implementation and experimental study*, Electron. Trans. Numer. Anal., 16 (2003), pp. 30–49.

[16] D. Chen and S. Toledo, *Combinatorial characterization of the null spaces of symmetric H-matrices*, Linear Algebra Appl., 392 (2004), pp. 71–90.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[17] P. Concus, G. H. Golub, and D. P. O'Leary, *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–332.

[18] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng, *Lower-stretch spanning trees*, in Proceedings of the 37th annual ACM symposium on Theory of computing (STOC), Baltimore, MD, 2005, ACM Press, New York, pp. 494–503.

[19] A. Frangioni and C. Gentile, *New preconditioners for KKT systems of network flow problems*, SIAM J. Optim., 14 (2004), pp. 894–913.

[20] K. D. Gremban, *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Oct. 1996. Available as Technical Report CMU-CS-96-123.

[21] K. D. Gremban, G. L. Miller, and M. Zagha, *Performance evaluation of a new parallel preconditioner*, in Proceedings of the 9th International Parallel Processing Symposium, IEEE Computer Society, 1995, pp. 65–69. A longer version is available as Technical Report CMU-CS-94-205, Carnegie-Mellon University.

[22] G. Haase, U. Langer, S. Reitzinger, and J. Schicho, *Algebraic multigrid methods based on element preconditioning*, Int. J. Comput. Math., 78 (2001), pp. 575–598.

[23] V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), pp. 155–177.

[24] M. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bureau Standards, 49 (1952), pp. 409–436.

[25] J. J. Júdice, J. Patricio, L. F. Portugal, M. G. C. Resende, and G. Veiga, *A study of preconditioners for network interior point methods*, Comput. Optim. Appl., 24 (2003), pp. 5–35.

[26] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[27] I. Koutis and G. L. Miller, *A linear work, $O(n^{1/6})$ time, parallel algorithm for solving planar Laplacians*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, LA, January 7–9, 2007, N. Bansal, K. Pruhs, and C. Stein, eds., SIAM, Philadelphia, 2007, pp. 1002–1011.

[28] U. Langer, S. Reitzinger, and J. Schicho, *Symbolic methods for the element preconditioning technique*, Technical report, Johannes Kepler Universität (JKU) Linz, Jan. 2002.

[29] B. M. Maggs, G. L. Miller, O. Parekh, R. Ravi, and S. L. M. Woo, *Finding effective support-tree preconditioners*, in SPAA '05: Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures, ACM Press, New York, 2005, pp. 176–185.

[30] T. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comput., 34 (1980), pp. 473–497.

[31] The MathWorks, *Matlab version* 7.2. software package, Jan. 2006.

[32] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comput., 31 (1977), pp. 148–162.

[33] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[34] P.-O. Persson and G. Strang, *A simple mesh generator in MATLAB*, SIAM Rev., 46 (2004), pp. 329–345.

[35] L. Portugal, F. Bastos, J. Júdice, J. Paixao, and T. Terlaky, *An investigation of interior-point algorithms for the linear transportation problem*, SIAM J. Sci. Comput., 17 (1996), pp. 1202–1223.

[36] L. F. Portugal, M. G. C. Resende, G. Veiga, and J. J. Júdice, *A truncated primal-infeasible dual-feasible interior point network flow method*, Networks, 35 (2000), pp. 91–108.

[37] S. Reitzinger, *Algebraic multigrid and element preconditioning* I, Technical report, J. Kepler Universität (JKU) Linz, Dec. 1998.

[38] M. Resense and G. Veiga, *An efficient implementation of the network interior-point method*, in Network Flows and Matching: The First DIMACS Implementation Challenge, D. Johnson and C. McGeoch, eds., DIMACS Series in Discrete Mathematics and Computer Science 12, AMS, New York, 1993.

[39] Y. Robert, *Regular incomplete factorizations of real positive definite matrices*, Linear Algebra Appl., 48 (1982), pp. 105–117.

[40] A. Shapiro, *Upper bounds for nearly optimal diagonal scaling of matrices*, Linear and Multilinear Algebra, 29 (1991), pp. 145–147.

[41] G. Shklarski and S. Toledo, *Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 7–40.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[42] H. Si, *TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator: Users's Manual for Version* 1.4, Jan. 2006, available online from http://tetgen.berlios.de.

[43] D. A. Spielman and S.-H. Teng, *Solving sparse, symmetric, diagonally-dominant linear systems in time* $0(m^{1.31})$, in Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Oct. 2003, pp. 416–427.

[44] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, ACM Press, New York, 2004, pp. 81–90.

[45] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems*, unpublished manuscript available online at http://arxiv.org/abs/cs/0607105, 2006.

[46] P. M. Vaidya, *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners*, unpublished manuscript. A talk based on this manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computations, Minneapolis, MN, October 1991.

[47] A. van der Sluis, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1969), pp. 14–23.

[48] R. S. Varga, Saff E. B., and V. Mehrmann, *Incomplete factorizations of matrices and connections with H-matrices*, SIAM J. Numer. Anal., 17 (1980), pp. 787–793.

[49] S. Vavasis, private communication, 2007.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.