NASA Contractor Report 178223

ICASE REPORT NO. 86-78

ICASE

ITERATIVE METHODS FOR ELLIPTIC FINITE ELEMENT EQUATIONS ON GENERAL MESHES

(NASA-CR-178223) ITERATIVE METHODS FOR ELLIPTIC FINITE ELEMENT EQUATIONS ON GENERAL MESHES Final Report (NASA) 44 p CSCL 12A

N87-18362

Unclas

G3/64 43356

R. A. Nicolaides

Shenaz Choudhury

Contract No. NAS1-18107

November 1986

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

NASA

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

ITERATIVE METHODS FOR ELLIPTIC FINITE ELEMENT EQUATIONS ON GENERAL MESHES

R. A. Nicolaides and Shenaz Choudhury
Department of Mathematics
Carnegie-Mellon University

ABSTRACT

This paper surveys iterative methods for arbitrary mesh discretizations of elliptic partial differential equations. The methods discussed are preconditioned conjugate gradients, algebraic multigrid, deflated conjugate gradients, an element-by-element technique, and domain decomposition. Computational results are included.

This work was supported by the Air Force Office of Scientific Research under Grant AFOSR-84-0137. In addition, the work of the first author was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-18107 while in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23665.

Iterative methods for elliptic finite element equations on general meshes

R.A. Nicolaides & Shenaz Choudhury
Department of Mathematics
Carnegie-Mellon University

1. Introduction

It is fair to say that the development of iterative solution techniques for all kinds of discretized partial differential equations remains a vigorous branch of numerical analysis. Perhaps the greater part of the effort has gone into multigrid algorithms, the next most common topic being preconditioning methods. Traditionally applied to elliptic problems, multigrid methods have also recently been successfully applied to solving the hyperbolic equations of gas dynamics (see [Jam1] for a survey).

Iterative methods in general have yet to penetrate mainstream (elliptic) finite element methodology, where direct solvers are the rule. This situation is partly historical, but is also due to the difficulty of using multigrid methods in situations of great geometrical complexity such as those routinely encountered in structural mechanics. It is not easy to automate the construction of several increasingly coarse embedded meshes which conform to the geometry of an arbitrary domain. Even when this is possible, problems with smoothing or other multigrid components may remain. In spite of these difficulties progress has been made towards a general implementation in [Loh1]. Where three dimensional problems must be solved, there does seem to be considerable interest in iterative methods, even for routine structural mechanics. The storage and time requirements of direct methods for such problems are sufficiently large that serious consideration of competing methods is a virtual necessity.

Aside from classical multigrid methods - which we will not consider further here - what choices remain? Basically there are two: general first or second order recursions with some form of preconditioner, and "algebraic multigrid". Both groups have their origins in the classical iterative methods. In the first class a special role is assigned to the conjugate gradient method, and in the second the Gauss-Seidel method is employed. The purpose of this paper is to survey some recent developments in this field. Section 2 introduces notation, conventions used in the analysis of iterative methods, indicates criteria for selecting good methods, and defines the conjugate gradient method on which many later results depend. Section 4 introduces the idea of preconditioning of conjugate gradients and defines an important class of preconditioners based on approximate Gauss elimination. Section 3 is about "algebraic multigrid" and its applicability to finite element problems. Section 5 introduces three newer methods. These are a "deflation" method for improving the convergence of conjugate gradients, an "element by element" iterative method, and some recent ideas based on classical substructuring, which may have applications to parallel computing. Section 6 gives a few computational results illustrating the properties of the various methods and also contains some further general remarks.

2. Preliminaries

The equation to be solved is

$$Ku = f \tag{2.1}$$

where K is an N × N (symmetric) positive definite matrix. ([Elm1] contains a survey of iterative methods for nonsymmetric problems.) In keeping with the aims of the paper, (2.1) is assumed to come from the finite element discretization of an elliptic equation or system. In practice, most such problems concern the equation

$$\text{div } (A \text{ grad } v) = g \tag{2.2}$$

in a bounded domain of $\mathbb{R}^2$ or $\mathbb{R}^3$ with appropriate boundary conditions. A denotes a positive definite second or fourth order tensor, frequently piecewise constant and/or highly anisotropic. The displacement equations of linear elasticity have the form (2.2) where v now denotes the displacement vector and A denotes the elasticity tensor. We will usually stay with the scalar case of (2.2) using it as a model for the vector case (but see Bercovier's example in section 6.2). Concerning the discretization of (2.2) we shall once and for all assume that the finite element space chosen uses only function values as nodal parameters on the mesh. This is an essential restriction for most of the more efficient methods we consider below; efforts to circumvent it tend to involve extra constraints. The important point is that efficient methods are not usually designed to deal with the general case, and can fail if applied indiscriminately.

Iterative methods (i.m.) produce a sequence of approximations $u^{(k)}$ to the solution u of (2.1). We define the *error* $\epsilon^{(k)} = u - u^{(k)}$ and the *residual* $r^{(k)} = f - Ku^{(k)}$. An important relation is the *residual equation*

$$K\epsilon^{(k)} = r^{(k)}. \tag{2.3}$$

A basic idea of i.m. is that if $u^{(k)}$ is a given approximation to u then we can write

$$u = u^{(k)} + \epsilon^{(k)}. \tag{2.4}$$

Although $\epsilon^{(k)}$ is unknown, we can use (2.3) to compute an approximation $e^{(k)}$ and then define

$$u^{(k+1)} = u^{(k)} + e^{(k)}. \tag{2.5}$$

Surprisingly crude choices of $e^{(k)}$ lead to convergent iterations. For example, even approximating (2.3) by

$$(1/\alpha)Ie^{(k)} = r^{(k)} \tag{2.6}$$

where α is a carefully chosen scalar and I is the identity matrix gives a (slowly) convergent algorithm: $u^{(k)} \to u$ as $k \to \infty$. This method, the simplest of all iterative methods was apparently first suggested in 1910 [Ric1] and is known as the (1st order) Richardson method. Its convergence can be improved by allowing α to vary with k. Using (2.6) in (2.5) we get the *iteration formula*

$$u^{(k+1)} = u^{(k)} + \alpha r^{(k)} \tag{2.7}$$

and from (2.3), (2.4), and (2.7) the *error equation*

$$\epsilon^{(k+1)} = (I - \alpha K)\epsilon^{(k)}. \tag{2.8}$$

The Jacobi method defines $e^{(k)}$ by

$$De^{(k)} = r^{(k)} \tag{2.9}$$

where D denotes the diagonal of K. For this, the error equation is

$$\epsilon^{(k+1)} = (I - D^{-1}K)\epsilon^{(k)}. \tag{2.10}$$

In the method of steepest descents introduced by Cauchy in 1847, (2.7) is replaced by

$$u^{(k+1)} = u^{(k)} + \alpha_k r^{(k)} \tag{2.11}$$

for which

$$\epsilon^{(k+1)} = (I - \alpha_k K)\epsilon^{(k)} \tag{2.12}$$

where $\alpha_k = (r^{(k)},r^{(k)})/(r^{(k)},Kr^{(k)})$, the parentheses denoting ordinary inner products of vectors. α_k chosen in this way has the property of minimizing $(u,Ku)-2(u,f)$, the energy functional associated with (2.1), down the gradient at the point $u^{(k)}$.

The classical Gauss-Seidel method defines $e^{(k)}$ as

$$e^{(k)} = (1/a_{kk})i_k i_k^T r^{(k)} \tag{2.13}$$

where the subscripts on the right are evaluated mod N, and i_k denotes the vector with 1 in position k, and 0 in all other positions. Letting k = 1,2, . . . ,N in turn in (2.13) and (2.5) gives one iteration of the Gauss-Seidel method. To generalize this to SOR, the second term in (2.13) is multiplied by ω, the relaxation factor.

Each of the methods defined above is a *first order* i.m. This refers to the fact that as in e.g. (2.12) successive errors (and iterates) satisfy a first order difference equation. For these and other methods it is often possible to determine a number $0<\rho<1$ such that

$$|\epsilon^{(k)}| \le C\rho^k|\epsilon^{(0)}| \tag{2.14}$$

where C is a constant, and $|.|$ denotes some norm on $\mathbb{R}^N$. Provided (2.14) is a sharp inequality, ρ can serve as a performance measure for a method; to be useful it must be complemented by a measure of the work needed to compute each iterate.

In the literature on i.m. it is customary to at least evaluate ρ for the test problem consisting of the standard 5 point approximation to the Laplacian on a uniform mesh of spacing $h = 1/(n+1)$ in a square of side 1 in the plane with Dirichlet boundary conditions. Each i.m. has its own analysis, usually involving substantial mathematics. Here, we shall merely list some results valid for $h \to 0$:

1st order Richardson	:	$\rho = 1 - Ch^2$
same: variable α_k	:	$\rho = 1 - Ch$
Jacobi	:	$\rho = 1 - Ch^2$
Steepest descents	:	$\rho = 1 - Ch^2$
Gauss-Seidel	:	$\rho = 1 - Ch^2$
SOR	:	$\rho = 1 - Ch$.

Proof of these results may be found in the standard references [For1], [Var1], [You1].

The constant C which appears is not necessarily the same for each method, but in every case is independent of h.

In general, if $\rho = 1 - Ch^\beta$ then $O(h^{-\beta})$ iterations are required to compute each new decimal digit of the solution. Thus, we see that e.g. SOR requires only about h times the number of iterations of Gauss-Seidel for a given accuracy for the simple test problem. But SOR is much more difficult to use because the parameter ω must be determined somehow, and the number of iterations can increase substantially if it deviates even slightly from its exact theoretical value.

The preceding observation suggests that we should list some criteria for an i.m. to be acceptable. Basic ones are the following:

1. High rate of convergence (i.e. small ρ).
2. Performance should be essentially independent of A in (2.2).
3. Users should not have to supply parameters or interact with the program.
4. A method should achieve an order of magnitude saving in storage and computation time over Gaussian elimination.

1 is required in order that the iteration can be terminated with a reasonable guarantee that the current approximation is a good one. It is very difficult to decide whether a slowly convergent iteration has in fact "converged". Concerning 2, the methods described above may fail for highly anisotropic or highly inhomogeneous materials. For example, if the square is bisected parallel to the y axis and the material tensor A is diag(1,1) in the left side and diag(κ,κ) where $0<\kappa<1$ on the right, then with certain boundary conditions, κ appears in the above expressions for ρ as $\rho = 1 - C\kappa h^{\beta}$. This gives an unacceptably poor rate of convergence if $\kappa \ll 1$. Incidentally, related difficulties can arise even with isotropic and homogeneous problems if there are high mesh aspect ratios or sudden changes in the mesh spacing. Then κ measures the mesh ratios etc. Point 3 reflects the fact that users are just that, and presumably do not want to become experts in i.m. Point 4 is the main rationale for considering i.m. in the first place: symmetric banded elimination for the model 2d problem has an operation count of $N^2/2$ and requires $N^{3/2}$ storage locations. Other demands could be made; for example, it could be required that i.m. impose absolutely no more constraints on the user than Gauss elimination, but this is not usually feasible.

We now generalize the previous ideas to *second order* i.m. The basic formula for these is a generalization of (2.7),

$$u^{(k+1)} = \omega_k u^{(k)} + \omega_k' u^{(k-1)} + \alpha_k \omega_k r^{(k)} \qquad (2.15)$$

where ω_k and α_k are scalar parameters, and $\omega_k + \omega_k' = 1$. The problem here is to choose the parameters to have a good convergence rate. As with the first order case, they may be chosen as constants (the second order Richardson method, due to Frankel [Fra1]), as dependent on k (the semi-iterative method [Var1]) or by a variational approach (conjugate gradients). We shall not consider the first two cases any further here, because the correct choice of the parameters requires a knowledge of the smallest and largest eigenvalues of K - information which is rarely available. For the details see [Hag1], [Var1], [You1]. Instead, consider the third case, the conjugate gradient method, where $\omega_0 = 1$ and

$$\alpha_k = (r^{(k)}, r^{(k)})/(r^{(k)}, Kr^{(k)}) \tag{2.16}$$

and

$$\frac{1}{\omega_k} = 1 - \frac{1}{\omega_{k-1}} \frac{\alpha_k}{\alpha_{k-1}} \frac{(r^{(k)}, r^{(k)})}{(r^{(k-1)}, r^{(k-1)})} \tag{2.17}$$

This choice for the 2 parameters of the basic formula (2.15) is obtained by imposing the 2 orthogonality conditions

$$(r^{(k+1)}, r^{(j)}) = 0 \qquad j = k,\ k-1 \tag{2.18}$$

on the new residual $r^{(k+1)}$. This is not the standard way of writing the conjugate gradient algorithm, but it is mathematically equivalent to it (see Appendix) and is, perhaps, conceptually simpler. The basic result which follows rather surprisingly from (2.15) - (2.17) is that $r^{(k+1)}$ is orthogonal to *all* the previous residuals, not just to the two previous ones by construction. This can be easily proved by induction, making essential use of the symmetry of K. Thus, after at most N steps the residual will be zero, and the current iterate will be u itself. In practice N is usually too large for the method to be used in this way. What occurs frequently is that the residual decreases sufficiently rapidly that an acceptable approximation is produced after considerably fewer steps. In fact, for second order homogeneous and isotropic equations, the required number of iterations is usually closer to $\sqrt{N}$ in two dimensions ($\sqrt[3]{N}$ in 3d). As with the first order i.m. the speed of convergence may slow significantly if the material properties deviate much from isotropy or homogeneity.

Applied to the standard test problem it can be proved [Hag1] that for conjugate gradients, the factor $\rho = 1 - Ch$. This implies the $O(\sqrt{N})$ iterations per digit of accuracy mentioned above. Conjugate gradients is the easiest way to get this 1 - Ch factor without supplying special problem dependent iteration parameters. Each of the methods so far considered either converge slower than conjugate gradients or need such parameters. This is perhaps the main reason for the popularity of the method. On the other hand, this is not very fast convergence. It would be more satisfactory to have $\rho = 1 - C\sqrt{h}$ say, giving $O(N^{1/4})$ iterations per digit, or even faster rates, provided they could be obtained at small cost.

It should be mentioned that making the constant C small is another way to improve convergence. This seems to be much more difficult than adjusting the exponent of h, to judge from the very small number of methods which achieve it. In the conjugate gradient setting, such a situation occurs sometimes when there are relatively few distinct eigenvalues. These distinct eigenvalues may range from, say $O(1)$ to $O(h^{-2})$, giving $\rho = 1 - Ch$, but if there are m of them it can be proved that at most m iterations will produce the exact solution of the linear system, apart from roundoff. In general, there are a large number of distinct eigenvalues so that the estimate in terms of h is more realistic, but transformations could be sought which reduce the number of distinct eigenvalues and make this estimate irrelevant.

Each iteration of conjugate gradients consists of forming matrix-vector products and dot products of vectors which are $O(N)$ operations. Since the number of digits significant in approximating the solution to the model differential equation is $O(\log_{10}N)$, the total work count is $O(N^{3/2}\log_{10}N)$. The storage is $O(N)$ because only the nonzeros of K are needed. This is to be compared with $N^2/2$ work and $N^{3/2}$ storage for the solution of the model problem by direct methods.

3. Algebraic multigrid (amg)

This technique attempts to extend the basic ideas of regular multigrid methods to a more general class of problems. No continuous problem underlies the given algebraic system of equations which is to be solved and, in particular, no grids are involved. The outline given here is based on [Rugl] which contains more details and references.

3.1 Multigrid principles

Standard multigrid algorithms are based on the following principles: consider the residual equation (2.3)

$$K\epsilon = r. \tag{3.1}$$

If K is a discrete partial differential operator, and r a corresponding source or load vector representing a suitably smooth function, then ϵ can probably be well approximated by defining a coarser problem

$$\hat{K}\hat{\epsilon} = \hat{r} \tag{3.2}$$

where ^ denotes a coarsening of the objects involved. For K the simplest example is the discrete 5 point Laplacian operator on a uniform mesh of side $h = 1/2^m$ in a unit side square. If $K \overset{\text{def}}{=} K(h)$ denotes this approximation, then $\hat{K} \overset{\text{def}}{=} K(2h)$. For $\hat{r}$ numerous approximations can be used, of which the simplest is $\hat{r} \overset{\text{def}}{=} r$ at points of the coarse (2h) mesh. If (3.2) can be solved, then $\hat{\epsilon}$ should approximate ϵ on the coarse mesh points. Then it can be interpolated (extended) to the fine mesh by a rectangular matrix E taking coarse vectors to fine ones, $\epsilon \simeq E\hat{\epsilon}$, so that (2.4) gives

$$u^{(new)} = u^{(old)} + E\hat{\epsilon}. \tag{3.3}$$

This algorithm requires specifying (1) the coarse grid operator $\hat{K}$ (2) the extension operator E and (3) a smooth residual r.

For standard finite elements there is a self evident definition of E, coming from using the coarse grid trial (shape) functions to interpolate to the fine grid. To find $\hat{K}$ we substitute the coarse trial functions into the energy functional and minimize in the usual way. It then turns out that

$$\hat{K} = E^T K E \tag{3.4}$$

so that $\hat{K}$ can be expressed in terms of K and E, and does not have to be chosen independently. In a very similar way, the correct choice for $\hat{r}$ is

$$\hat{r} = E^T r. \tag{3.5}$$

Thus, the components of (3.2) are fully defined once E is specified.

The construction of a smooth r is achieved by the application of a few iterations of a classical iterative method to an arbitrary starting approximation. On (i,j) meshes relaxation methods, such as Gauss-Seidel in its point or line versions, are frequently successful.

In practice, (3.2) is itself reduced by smoothing and coarsening, and so on recursively until an easily solved coarse equation, usually containing just a handful of unknowns, is reached. This recursion is rather irrelevant to the main mathematical properties of multigrid methods. Its significance is practical: without it, (3.2) cannot be efficiently solved.

The finite element algorithm just described was first defined and its ("W-cycle") convergence analyzed in [Nic1], [Nic2] following earlier finite difference work. Many improvements and additions to this analysis have since been made. [Hac1] is a recent reference on the theoretical aspects of multigrid methods for elliptic problems. Here, we wish to write out the error formula for the two grid method in order to motivate some of the amg concepts below. Substituting (3.2) into (3.3) and subtracting both sides from the exact solution u gives

$$\epsilon^{(new)} = (I - E\hat{K}^{-1}E^T K)\epsilon^{(old)}. \tag{3.6}$$

The key observation is that if

$$\epsilon^{(old)} = E\eta \tag{3.7}$$

for some η then since $\hat{K} = E^T KE$,

$$\epsilon^{(new)} = 0 \tag{3.8}$$

i.e., we will have the exact solution on completing the coarse grid operation. This conclusion is independent of any particular choice of E. Of course, there is no reason in general to suppose that (3.7) will hold, but we can try to relate the smoothing to the choice of coarse grid so that it is nearly true. Then we can expect to get a rapidly convergent method. amg attempts explicitly to achieve this correspondence.

3.2 amg components

In standard multigrid, the coarse meshes are assumed given and a smoothing algorithm must be found which enables smoothed residuals and errors to be adequately represented on these meshes. In the amg context where no meshes exist, the reverse idea is adopted: the smoothing algorithm is chosen first, and based on what it achieves - strongly dependent on the properties of K - the coarse operators are constructed. This is done by using the method of section 3.1, i.e., by choosing a coarse to fine extension operator E and defining the other operations in terms of it as in (3.4) - (3.5). This must be done recursively for several coarsenings.

In practice, this idea is not easy to implement, and attention has so far been restricted to symmetric positive definite matrices K satisfying the additional conditions

$$k_{ij} \leq 0 \qquad i \neq j, \qquad \sum_{j=1}^{N} k_{ij} \geq 0 \qquad i,j = 1,2, \ldots ,N. \tag{3.9}$$

Numerical results show that the method continues to converge when these conditions are slightly violated.

Only the Gauss-Seidel method is used as an amg smoother in [Rug1]. Relaxation at a single point is given by

$$u_i = -(1/k_{ii})\Big(\sum_{j\neq i}^{(1)} k_{ij}u_j + \sum_{j\neq i}^{(2)} k_{ij}u_j - f_i\Big) \tag{3.10}$$

where $\Sigma^{(1)}$ denotes summation over points already updated in this pass and $\Sigma^{(2)}$ over those not yet modified. Smoothing occurs for *strongly connected* subsets of the unknowns, where the *strength* is measured by the relative magnitude of the off-diagonals $|k_{ij}|$.

Next, note that for the errors ϵ_i, (3.10) gives

$$\epsilon_i = -(1/k_{ii})\Big(\sum_{j\neq i}^{(1)} k_{ij}\epsilon_j + \sum_{j\neq i}^{(2)} k_{ij}\epsilon_j\Big). \tag{3.11}$$

(3.11) can be further simplified by ignoring the notational distinction between $\Sigma^{(1)}$ and $\Sigma^{(2)}$, and by setting to zero the terms with only weak (i.e. $|k_{ij}|$ is relatively small) connections. Let S_i denote the set of strong connections of point i. Then

$$\epsilon_i \cong -(1/k_{ii}) \sum_{\substack{j \in S_i \\ j \neq i}} k_{ij}\epsilon_j. \tag{3.12}$$

Let C denote the set of points designated as coarse, and F the remainder. Interpolation to an F point is by linear combinations of values at C points. Let $S_i(E) \subseteq S_i \cap C$ denote the set to be used for interpolation to point $i \in F$. (3.12) now becomes

$$\epsilon_i \cong -(1/k_{ii})\Big(\sum_{j \in S_i(E)} k_{ij}\epsilon_j + \sum_{\substack{j \notin S_i(E) \\ j \neq i}} k_{ij}\epsilon_j\Big). \tag{3.13}$$

(3.13) is a good approximation to the actual smoothing formula we are using. Recalling (3.7) we want to use (3.13) as the interpolation formula too. But the points in the second sum are not interpolation points for $i \in F$. Clearly, we must choose C so that they can be expressed in terms of $S_i(E) \cup \{i\}$ values. Several ideas have been proposed. Good results [Rug1] have been obtained by choosing C so that points in the second sum in (3.13) are strongly connected to the set $S_i(E)$. For these points the following approximations are now made:

$$\epsilon_j \cong \Big(k_{ji}\epsilon_i + \sum_{\ell \in S_i(E)} k_{j\ell}\epsilon_\ell\Big)\Big/\Big(k_{ji} + \sum_{\ell \in S_i(E)} k_{j\ell}\Big). \tag{3.14}$$

Substituting (3.14) into (3.13) gives $\epsilon_i \cong \sum_{j \in S_i(E)} v_{ij}\epsilon_j$ for $i \in F$, where

$$v_{ij} = -(k_{ij} + c_{ij})/(k_{ii} + c_{ii}) \tag{3.15}$$

and

$$c_{ij} = \sum_{\substack{\ell \in S_i(E) \\ \ell \neq i}} k_{i\ell} k_{\ell j} / (k_{\ell i} + \sum_{m \in S_i(E)} k_{\ell m}). \qquad (3.16)$$

For i ∈ C the interpolated value is just the coarse value existing at i. This defines E, and the rest of the problem setup follows automatically, as in section 3.1.

It remains to give explicit rules for the construction of C and F. It is very important that C contains a relatively small number of points, because $|C|$ is the size of the coarse problem. A low rate of coarsening would produce a large number of successively coarser levels, causing problems with storage as well as efficiency. Finding suitable sets C is thus crucial to the operation of the algorithm. The algorithm given in [Rug1] for finding C is, interestingly, considerably more complicated than the amg algorithm itself, involving much graph theoretical manipulation and empirical testing. We will not describe it here. Full details including flow charts and rules for picking parameters are in [Rug1].

3.3 Applications

Some figures from [Rug1] give an idea of the behavior of the amg algorithm described above. Several examples are reported including severely anisotropic and nonhomogeneous scalar second order elliptic problems. The meshes were structured although of course this fact was not explicitly used. The performance is uniformly good on all the test problems, once the setup phase is complete. The latter appears to cost about 1 - 2 times as much as the solution itself, so that presumably a number of solutions would have to be done to neutralize this cost. Convergence factors are quite remarkable, being well below 0.1 for the Poisson equation on an h = 1/64 mesh, giving an expectation of at least one new correct digit each iteration, and less than 0.12 for all the test problems. For the work to actually solve a model problem with a given precision, see section 6.4.

4. Preconditioning and preconditioners

In this section, we will survey a class of techniques for improving the convergence of i.m. Although these techniques can be used quite generally, we will consider only their application to conjugate gradients (cg). It has long been known that cg works best for matrices which have small condition numbers $\lambda_{max}/\lambda_{min}$, the number of iterations required per digit of accuracy being proportional to the square root of the condition number. Convergence is also improved if there are many nearly identical eigenvalues. "Preconditioning" refers to the general strategy of transforming K to reduce its condition number or cluster or otherwise redistribute its eigenvalues.

4.1 Using preconditioners

Suppose we have constructed a representation of K in the form

$$K = LL^T + R \tag{4.1}$$

where L is lower triangular and R can be interpreted as the error of the approximate factorization of $K \cong LL^T \equiv M$. Then we can solve

$$M^{-1}Ku = M^{-1}f \tag{4.2}$$

instead of (2.1) with the expectation of a smaller condition number or of better clustered eigenvalues for $M^{-1}K$ than for K. A minor point is that $M^{-1}K$ is not symmetric, so that cg may not converge for (4.2) as it stands. But using the definition of M it can be transformed into

$$L^{-1}KL^{-T}v = L^{-1}f \qquad v = L^Tu \tag{4.3}$$

where the coefficient matrix is symmetric positive definite. This appears slightly inconvenient in that it does not refer to the original variable u, but it is easy to express the iteration in terms of approximations $u^{(k)}$ convergent directly to u. Indeed, (2.15) gives for (4.3)

$$v^{(k+1)} = \omega_k v^{(k)} + \omega_k' v^{(k-1)} + \alpha_k \omega_k s^{(k)} \tag{4.4}$$

where $s^{(k)}$ denotes the residual of (4.3),

$$\alpha_k = (s^{(k)}, s^{(k)})/(s^{(k)}, L^{-1}KL^{-T}s^{(k)}) \tag{4.5}$$

and ω_k is defined by the recursion formula corresponding to (2.17). Defining $u^{(k)} = L^{-T}v^{(k)}$, (4.4) gives

$$u^{(k+1)} = \omega_k u^{(k)} + \omega_k' u^{(k-1)} + \alpha_k \omega_k L^{-T}s^{(k)} \tag{4.6}$$

while (4.3) gives

$$s^{(k)} = L^{-1}(f - KL^{-T}L^{T}u^{(k)}) = L^{-1}r^{(k)} \tag{4.7}$$

so that (4.6) can be expressed as

$$u^{(k+1)} = \omega_k u^{(k)} + \omega_k' u^{(k-1)} + \alpha_k \omega_k M^{-1}r^{(k)}. \tag{4.8}$$

Similarly, by (4.7)

$$\alpha_k = (r^{(k)}, M^{-1}r^{(k)})/(M^{-1}r^{(k)}, KM^{-1}r^{(k)}) \tag{4.9}$$

and ω_k continues to be given by the formula (2.17),

$$\frac{1}{\omega_k} = 1 - \frac{1}{\omega_{k-1}} \frac{\alpha_k}{\alpha_{k-1}} \frac{(r^{(k)}, M^{-1}r^{(k)})}{(r^{(k-1)}, M^{-1}r^{(k-1)})}. \tag{4.10}$$

To summarize, (4.8) - (4.10) are the formulas for preconditioned cg with preconditioner M. It is clear that the only extra work consists of forming $M^{-1}r^{(k)}$ at each iteration.

This formulation shows that any convenient M can be used as preconditioner and that knowing its factored form (LL^T) is theoretically superfluous. Some very simple preconditioners can exploit this, for example diagonal preconditioning in which the preconditioner is D, the diagonal of K. More complicated choices of M, required for faster convergence, should also be cheap to form, store and invert. This motivates the definition of M as the product of incomplete factors in the next subsection. These factors preserve the sparsity pattern of K, so that only O(N) operations are needed for the forward and back substitutions to compute $M^{-1}r^{(k)}$. The factors themselves are computed, once and for all, in O(N) operations, and require only O(N) storage.

4.2 Incomplete factorizations

The most important class of preconditioners is based on an idea known as *incomplete factorization*. This idea was first suggested by Varga [Var2], although a Russian paper [Bul1] the same year - according to some remarks in [Sto1] - contains the algorithm of [Dup1] which itself contains a detailed implementation of Varga's idea. These authors consider the application of preconditioning to first order i.m. for 5 or 9 point difference formulas in rectangles. The application to cg was suggested much later by Meijerink and Van der Vorst [Mei1], who also give other extensions.

We will quickly review the standard matrix formulation of Gaussian elimination without pivoting. Let $K^{(i)}$ be the ith stage matrix with pivot row i, and columns 1,2, . . . ,i-1 already eliminated. Then with $K^{(1)} = K$,

$$K^{(i+1)} = L^{(i)}K^{(i)} \qquad i = 1,2,\ . \ . \ . \ ,N-1 \tag{4.11}$$

where $L^{(i)}$ is unit lower triangular, having multipliers

$$-k^{(i)}_{ji}/k^{(i)}_{ii} \qquad j = i+1,\ . \ . \ . \ ,N \tag{4.12}$$

in column i, and zeros in the other off diagonal positions. Then

$$K^{(N)} = L^{(N-1)}.\ .\ .L^{(1)}K \equiv U \tag{4.13}$$

where U is upper triangular. It follows by a simple direct calculation that

$$K = LU \tag{4.14}$$

where L is unit lower triangular with the negatives of (4.12) in the same places below the diagonal.

Instead of (4.11), consider the more general recursion with $\kappa^{(1)} = K$,

$$\kappa^{(i+1)} = \ell^{(i)}\kappa^{(i)} - R^{(i)} \qquad i = 1,2,\ . \ . \ . \ ,N-1 \tag{4.15}$$

where $\mathcal{L}^{(i)}$ eliminates column i from $\kappa^{(i)}$, and $R^{(i)}$, called the local error matrix, is any N × N matrix not introducing fill and such that the (i+1)th step is well defined. Denoting by $\Pi_{j,k}$ the product $\mathcal{L}^{(j)}\mathcal{L}^{(j-1)} \ldots \mathcal{L}^{(k)}$ and by $\Pi^{k,j}$ its inverse, it follows from (4.15) that

$$\Pi_{N-1,1}K = \kappa^{(N)} + \Pi_{N-1,2}R^{(1)} + \Pi_{N-1,3}R^{(2)} + \ldots + R^{(N-1)} \qquad (4.16)$$

where $\kappa^{(N)}$ is upper triangular. Hence, defining $\mathcal{L} = \Pi^{1,N-1}$ and $\mathcal{U} = \kappa^{(N)}$, (4.16) gives

$$\begin{aligned} K &= \mathcal{L}\mathcal{U} + \Pi^{1,1}R^{(1)} + \Pi^{1,2}R^{(2)} + \ldots + \Pi^{1,N-1}R^{(N-1)} && (4.17) \\ &= \mathcal{L}\mathcal{U} + R && (4.18) \end{aligned}$$

where R denotes the combination of the local error matrices. (4.18) shows that $\mathcal{L}\mathcal{U}$ is the Gaussian decomposition of K - R. If the $R^{(i)}$ are well chosen, then $\mathcal{L}\mathcal{U}$ can approximate LU of (4.14) and be used as a preconditioner.

A significant specialization of (4.17) occurs when $R^{(i)}$ is zero in and above its ith row, so that $R^{(i)}$ modifies the exact elimination result only below the pivot row. In this case, (4.17) becomes

$$K = \mathcal{L}\mathcal{U} + R^{(1)} + R^{(2)} + \ldots + R^{(N-1)} \qquad (4.19)$$

because

$$\Pi^{1,i}R^{(i)} = (\mathcal{L}^{(1)})^{-1}(\mathcal{L}^{(2)})^{-1} \ldots (\mathcal{L}^{(i)})^{-1}R^{(i)}, \qquad (4.20)$$

and the successive inverse elimination matrices use only pivot rows 1 through i of $R^{(i)}$. Thus R is just the sum of the local error matrices.

There are two more or less standard ways to choose R, both controlling the fill in $\mathcal{L}$ and $\mathcal{U}$. In the first approach, $R^{(i)}$ is simply defined to contain the fill from the ith step of the elimination. This is the method of [Meil]. It follows that wherever K has nonzero entries, R has zero entries and thus, since

$$\mathcal{L}\mathcal{U} = K - R \qquad (4.21)$$

K and $\mathcal{L}\mathcal{U}$ agree on the nonzeros of K. In the second approach [Dup1], [Dup2] a modification is made to the R just defined, consisting of adding to the diagonal of $R^{(i)}$ at each stage the negative of its row sums, so that the resulting matrix has zero row sums. Thus, we keep agreement of the off diagonal nonzero terms of K with those of the approximate factorization, while giving up agreement of the diagonal terms in favor of having the row sums of K and $\mathcal{L}\mathcal{U}$ equal. The second approach is usually considerably better than the first in terms of convergence of preconditioned cg (pcg).

In both of these methods, the zero patterns of $\mathcal{L}$ and $\mathcal{U}$ match those of the lower and upper triangular parts of K. This suggests seeking $\mathcal{L}$ and $\mathcal{U}$ directly in this form, by multiplying them together and equating to K. The extra nonzero terms (analogous to fill) arising in this product are put into R. In a similar way, a Cholesky type decomposition can be obtained. This approach is usual in the finite difference literature and is especially helpful when there is an (i,j) mesh available. However, it is possible to show that the different approaches lead to the same approximate factors.

The above approximate factorizations are called *incomplete* factorizations, and abbreviated as "ILU" (for incomplete LU) or "IC" (incomplete Cholesky). If the zero row sum feature is incorporated, the methods are usually called the modified incomplete LU (MILU) or Cholesky factorizations (MIC).

4.3 Theoretical results

There are two groups of results available for ILU methods. The first concerns the existence and stability of the approximate factorizations, and the second gives estimates for the condition numbers of the preconditioned matrices, from which the rate of convergence of pcg can be found. In this section we will state representative theorems of this kind.

It follows from (4.21) that $\mathcal{L}\mathcal{U}$ is the decomposition of K - R. In the ILU setting, if K is positive definite then R will be symmetric with 0 diagonal and so indefinite. If K also has nonpositive off diagonal entries, then in the MILU case R will have a positive diagonal, negative off diagonal entries and zero row sums. Hence, it is positive semidefinite. In both cases therefore, it follows that the matrix actually factored is less positive definite than K. It is conceivable - and can actually happen for merely positive definite matrices - that the construction of ℓ and $\mathcal{U}$ can break down. On the other hand, if K - R is positive definite not only does the factorization exist, it is also stable.

In [Meil] existence and relative stability are proved for M-matrices, i.e., matrices K such that $k_{ij} \leq 0$ for $i \neq j$ and $K^{-1} \geq 0$. For such matrices, it is known that Gauss factorization is well defined.

Theorem 1 [Meil]

Let K be an M-matrix. Then the ILU factorization algorithm is well defined for K. Moreover the factorization is relatively stable in the sense that the ILU pivots are at least as large in magnitude as those of the exact LU process, and $|\ell| \leq |L|$ elementwise where L is the exact lower triangular factor of K.

For MILU we have another theorem.

Theorem 2

Let K satisfy the conditions

$$k_{ij} \leq 0 \quad i \neq j, \quad k_{ii} > 0 \quad i,j = 1,2, \ldots, N \tag{4.22}$$

and

$$k_{ii} > -\sum_{j \neq i} k_{ij} \tag{4.23}$$

Then for MILU, K - R is positive definite.

Condition (4.23) can be replaced with the weaker $\geq$ condition with at least one strict inequality, provided each row of K contains a nonzero element after the diagonal [Axel]. A class of matrices satisfying this and also the conditions of Theorem 1 is given by linear finite element discretizations of the scalar problem (2.2) with Dirichlet boundary conditions provided all of the triangles are acute. Other than this, there does not seem to be any large class of symmetric finite element problems satisfying the conditions of either theorem.

But in practice both methods have been successful in cases where these conditions are not too strongly violated. Also, convection-diffusion equations discretized by upwind schemes often satisfy the conditions of Theorem 1 [Mei2].

The condition number of the preconditioned matrix $M^{-1}K$ can be computed for model problems. Chandra proved the following result.

Theorem 3 [Cha2]

For the Laplacian operator $-\Delta$ with Dirichlet boundary conditions in a unit side square in $\mathbb{R}^2$, discretized by linear elements on a standard triangulation of side h, and preconditioned by ILU there exist constants C_1 and C_2 such that

$$C_1 h^{-2} \leq \text{cond}(M^{-1}K) \leq C_2 h^{-2}. \qquad (4.24)$$

From (4.24) it follows that ILU pcg does not have an improved rate of convergence over regular cg in terms of powers of h. Nevertheless, it is observed in applications that ILU pcg does give a more efficient algorithm than cg alone in the sense that a smaller number of iterations and a smaller amount of work are needed to solve with given accuracy. But as h decreases, the required number of iterations to compute a new digit increases as h^{-1}. If we have a method where this figure increases like some smaller power of h, then as h reduces, at some point the second method will become cheaper. Where this point actually occurs is practically unpredictable and has to be found experimentally. For the model problem of Theorem 3 it is proved (for finite differences) in [Gus1] that a slight variation of MILU has $\text{cond}(M^{-1}K) = O(h^{-1})$ giving $O(h^{-1/2})$ iterations per digit, potentially a large saving over ILU. The variation consists of adding a quantity of $O(h^2)$ to the diagonal during the factorization. We will call this MILU+ for brevity. In practice, the same performance is seen for MILU.

Theorem 4 [Gus1]

For the problem in Theorem 3, for MILU+, constants D_1, D_2 exist such that

$$D_1 h^{-1} \leq \text{cond}(M^{-1}K) \leq D_2 h^{-1}. \qquad (4.25)$$

In the case of an arbitrary mesh and for other generalizations the $O(h^{-1})$ condition number can be obtained, although once again only by introducing one or more problem dependent parameters. [Axel] contains an extensive account of these developments.

4.4 Further developments

This subsection gives several refinements and extensions of the basic algorithms. First, we will consider procedures which increase the density of nonzero elements in $\mathcal{L}$ and $\mathcal{U}$. [Mei2] contains a suggestion for doing this with finite differences. The ILU procedure, suitable for (i,j) difference schemes, is to define a prior set P of entries in $\mathcal{L}$ where nonzeros are permitted to occur at least including the nonzero positions of K. In all other positions, $\ell_{ij} = 0$ by definition. $\mathcal{U}$ has the transposed nonzero positions. Then we generate the permitted entries in $\mathcal{L}$ and $\mathcal{U}$ by

$$\begin{aligned} \mathcal{U}_{ii} &= k_{ii} - \sum_{k=1}^{i-1} \ell_{ik}\mathcal{U}_{ki} \\ \ell_{ij} &= (1/\mathcal{U}_{jj})(k_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\mathcal{U}_{kj}) \\ \mathcal{U}_{ij} &= k_{ij} - \sum_{k=1}^{i-1} \ell_{ik}\mathcal{U}_{kj}. \end{aligned} \tag{4.26}$$

These are the usual recurrences, but with the entries not in P omitted. For mesh calculations with e.g. 5 point formulas, a convenient choice for P consists of K's nonzero bands together with some nearby ones [Mei2].

Clearly, this technique is not very useful for general meshes. A more suitable method given by [Gus2] is to recursively define $\mathcal{L}^{(s)}\mathcal{U}^{(s)}$ by defining $P^{(s)}$ to be the nonzero set associated with the product $\mathcal{L}^{(s-1)}\mathcal{U}^{(s-1)}$. $P^{(0)}$ is defined to be P, so s = 0 corresponds to the original approximate factorization. In general, as s increases $P^{(s)}$ contains more and more nonzero positions, and eventually all of them, so that an exact factoring would be required. [Gus2] reports that small values of s give the best overall results.

Another problem which arises in solving general positive definite systems is that of negative pivots in ILU and MILU. Conjugate gradients is proved to converge only for symmetric positive definite matrices. If $\mathcal{U}$ doesn't have a positive diagonal a problem may arise. Two papers [Ker1], [Man1] deal with the avoidance of these negative or small pivots. In [Ker1] the proposed remedy is to add a sufficiently large number to the diagonal culprit in $\mathcal{U}$. This just corresponds to a further addition into the $R^{(i)}$ for this step. The argument usually advanced is that if not too many of these corrections have to be made, their effect on the rate of convergence of pcg will be small. Perhaps because of its ad hoc nature no rigorous estimate of the effect of this modification seems to be known. In [Man1] the prior addition of a matrix αI to K is advocated. However, [Mei2] points out that this is a global change being made to correct what can be conceived as a local problem. Moreover, a problem dependent parameter has once again crept in. In spite of these objections, some good results have been reported in both of the above references.

5. Other methods

In this section, we will very briefly mention three methods which have been recently proposed. Much less is known about them than the methods considered above but all have some new feature which may be of interest. They are "deflated cg", the "element by element method", and "domain decomposition". In each case we can do little more than describe the algorithm and mention whatever else seems relevant.

5.1 Deflated cg

This method, described more fully in [Nic3], gives another way to improve the convergence of cg or pcg. It is closer in spirit to multigrid methods than to incomplete factorizations, but is suitable for general meshes.

Recalling (2.15) we can generalize it to

$$u^{(k+1)} = \omega_k u^{(k)} + \omega_k' u^{(k-1)} + \alpha_k \omega_k (r^{(k)} - Ec^{(k)}) \qquad (5.1)$$

where E is $N \times m$ ($m < N$) and has a meaning similar to the E in section 3. The idea behind (5.1) is that $Ec^{(k)}$ "deflates" certain constituents of the residual, particularly those for which the regular algorithm is ineffective. $c^{(k)}$ is chosen to minimize

$$(K(r^{(k)} - Ec^{(k)}),\ r^{(k)} - Ec^{(k)})$$

leading to

$$E^TKEc^{(k)} = E^TKr^{(k)} \tag{5.2}$$

which must be solved for $c^{(k)}$ at each iteration. Since $m \ll N$ this is feasible. From (5.1) and (5.2) we find

$$r^{(k+1)} = \omega_k r^{(k)} + \omega_k' r^{(k-1)} - \alpha_k\omega_k K(I - E\mathbf{K}^{-1}E^TK)r^{(k)} \tag{5.3}$$

where $\mathbf{K} \equiv E^TKE$ (c.f. (3.4)). Let $P = E\mathbf{K}^{-1}E^TK$. In (5.3) $K(I - P)$ is positive semidefinite since it is symmetric and equals

$$K^{1/2}(I - K^{1/2}PK^{-1/2})K^{1/2}$$

where the bracketed matrix is an orthogonal projection matrix, and $K^{1/2}$ is positive definite. If $E^Tr^{(0)} = 0$ then it follows by induction from (5.3) that

$$E^Tr^{(k)} = 0 \qquad k = 1,2, \ldots ,N. \tag{5.4}$$

This shows that the residual is always orthogonal to the column space of E and so we can try to set up a cg iteration in the subspace null(E^T), which presumably will converge faster than in the whole space for a good choice of E. To set up the iteration, all we have to do is pick the two coefficients in (5.3) - which are arbitrary up until now - to force the usual orthogonalities

$$(r^{(k+1)}, r^{(j)}) = 0 \qquad j = k,\ k-1$$

from which it follows as before that

$$(r^{(k+1)}, r^{(j)}) = 0 \qquad j = 0,1, \ldots ,k.$$

To make $E^Tr^{(0)} = 0$, pick v arbitrarily, let $s = f - Kv$ and solve

$$Kd = E^T s$$

for d. Then define

$$u^{(0)} = v + Ed.$$

$r^{(0)}$ now has the required property.

Deflation is used in a very similar way to improve a preconditioned cg algorithm [Nic3].

5.2 Convergence

Success with the deflation technique is dependent on a good choice of E. A basic strategy for problems with smooth coefficients and not too much anisotropy is to divide the domain into m disjoint subdomains of approximately equal areas and to associate one column of E with each subdomain. The jth column will be zero in every position except those corresponding to the unknowns in the jth subdomain, where it is equal to 1. (5.4) then implies that the residuals always have zero mean in each subdomain. In the general case, the choice of E depends on the properties of K, and will have to be made using the ideas required to pick the corresponding operator for algebraic multigrid.

If the maximum area of the subdomains is small, then K will be of large order, although convergence will be rapid. If the minimum area is too large, then K will be a small matrix, but will not much help the convergence relative to cg. For the above "piecewise constant" choice of E, it is proved in [Nic3] that for second order equations of the type (2.2) with smooth coefficients and meshes, the error multiplier $\rho = 1 - Ch/d$ where d^2 is the area of the largest subdomain. If there are severe anisotropies or discontinuities, their effect will show up in C and slow the convergence. E has to be chosen somewhat differently for these cases. For the Poisson equation with Dirichlet conditions and a uniform mesh [Nic3] proves that choosing $d = O(h^{3/5})$ gives a cost of $O(N^{6/5})$ flops per digit. This cost is a slight theoretical improvement over say MILU+, for which the corresponding figure is $O(N^{5/4})$.

The deflation algorithm has the advantage of more general applicability than ILU type methods because it is based on pde theory rather than on Gauss elimination. It would apply immediately to linear elasticity for example. On the other hand, good choices for E are problem dependent for degenerate cases. General software for generating E in such cases could be developed, however.

See section 6.3 for some numerical results.

5.3 Element by element method

This section contains a brief description of an algorithm recently proposed by Hughes et al [Hug1]. Unlike the methods of the previous sections, this one is specifically for finite element equations because it uses the fact that the stiffness matrix is a sum of element stiffness matrices. In [Hug1] the steady state solution of the ode system

$$\begin{aligned} W du/dt &= Ku - f \\ u(0) &= 0 \end{aligned} \tag{5.5}$$

is approximated by the iteration

$$\begin{aligned} (W - \delta t K)u^{(k+1)} &= Wu^{(k)} - \delta t f \\ u^{(0)} &= 0. \end{aligned} \tag{5.6}$$

In this section only it is assumed that K is *negative* definite. W is a positive definite matrix chosen to enhance the convergence to steady state, e.g. $W = -\mathrm{diag}(K)$, and δt is the time step. It follows from (5.6) after some manipulation that

$$u^{(k+1)} = u^{(k)} - \delta t W^{-1/2} V W^{-1/2} r^{(k)}$$

where as usual, $r^{(k)} = f - Ku^{(k)}$ and

$$V = (I - \delta t W^{-1/2} K W^{-1/2})^{-1}.$$

Noting for finite element systems that

$$K = \Sigma K_i$$

where K_i are element matrices it follows that

$$V = (I - \delta t \sum_i W^{-1/2} K_i W^{-1/2})^{-1}.$$

[Hug1] suggests the approximation

$$(I - \delta t \sum_i W^{-1/2} K_i W^{-1/2})^{-1} \approx \prod_{i=1}^{\mu} (I - \delta t W^{-1/2} K_i W^{-1/2})^{-1} \qquad (5.7)$$
$$\equiv V_1.$$

Each of the inverses on the right is easy to compute, in essence reducing to the inversion of a matrix of order equal to that of K_i. It is also suggested that the approximations (5.7) and

$$V_2 \equiv \prod_{i=\mu}^{1} (I - \delta t W^{-1/2} K_i W^{-1/2})^{-1}$$

be used alternately. The product $V_3 \equiv V_1 V_2$ is symmetric which may be a desirable property in certain cases. For this, the algorithm becomes

$$u^{(k+1)} = u^{(k)} - \delta t W^{-1/2} V_3 W^{-1/2} r^{(k)}. \qquad (5.8)$$

Of course, the convergence will be improved if partial assemblies are carried out, although the cost of the inversions will increase.

Some further refinements are to use the well known BFGS update to improve the "search direction" $W^{-1/2} V_3 W^{-1/2}$ in (5.8) and to use an accurate line search in the improved direction. Numerical results for a plane strain problem are reported in [Hug1].

The line search/BFGS combination actually brings the iteration closer to a pcg technique. In fact V_3 can be used as a cg preconditioner. This is investigated in [Nou1].

5.4 Domain decomposition / substructuring

This section describes some work which is the subject of much current attention [Bra1], [Cha1]. We can give only a general impression of the underlying ideas.

"Substructuring" refers to the technique of ordering the variables of a finite element system in such a way that the coefficient matrix takes the form

$$\begin{bmatrix} K_{11} & \cdot & & K_{1s} \\ & K_{22} & & K_{2s} \\ \cdot & \cdot & & \cdot \\ & & K_{s-1s-1} & K_{s-1s} \\ K_{1s}^T & K_{2s}^T \quad \cdot & K_{s-1s}^T & K_{ss} \end{bmatrix}. \qquad (5.9)$$

The diagonal blocks are usually square matrices, although this is not necessary [Gun1]. Usually, the variables belonging to each diagonal block are associated with some disjoint physical subdomains, and those belonging to the last block column are variables associated with the interfaces between (and disjoint from) the subdomains. Such an ordering is frequently convenient on physical grounds, where the subdomains may represent different parts of a physical structure. These orderings also seem attractive for parallel processing. The recent interest is mostly motivated by this last factor. Specifically, the (block) last row of (5.9) can be *simultaneously* eliminated giving a block upper triangular system with coefficient matrix

$$\begin{bmatrix} K_{11} & \cdot & & K_{1s} \\ & K_{22} & & K_{2s} \\ & \cdot & & \cdot \\ & & K_{s-1s-1} & K_{s-1s} \\ & & & C \end{bmatrix} \qquad (5.10)$$

where

$$C = K_{ss} - K_{1s}^T K_{11}^{-1} K_{1s} - \cdot \cdot \cdot - K_{s-1s}^T K_{s-1s-1}^{-1} K_{s-1s}. \qquad (5.11)$$

Back substitution with this matrix can be achieved with another simultaneous operation once the last (vector) unknown representing the interface variables is found. It is the solution of the latter equation, say

$$Cu_B = g \tag{5.12}$$

which is the bottleneck for the parallel implementation of this method. We are confronted with a familiar situation: if s is large, each K_{ii} can be of small order and easy to invert. But C will be of large order and expensive to form and invert. If s is small, K_{ii} can be large and more difficult to invert, while C will be small, fairly easily formed and easily inverted.

For parallel implementation it is mostly the first case which is important. However, the gains from parallelism can be lost because of the problems of solving (5.12) for large C. To circumvent this, several suggestions have recently been made for solving (5.12) iteratively. It can be proved without difficulty that for the type of problem we are considering, C is positive definite so that cg is naturally suggested. The residual can be simultaneously computed using (5.11) and the L_iU_i factors of K_{ii}. The main problem is then to speed the convergence of the cg iteration. The important new point is that the elements of C are not explicitly given, so that preconditioners of the earlier sections cannot be easily used.

Some recent work has dealt with the case of the Poisson equation on a uniform mesh in a rectangular domain divided horizontally by one or more mesh lines into subdomains. For this case, Fourier techniques can be used to find the eigenvalues and eigenvectors of C explicitly. Based on this, [Dry1] proposes the choice of preconditioner M as

$$M = \sqrt{L}$$

where $L = (1/h^2)\{-1\ 2\ -1\}$ with suitable boundary conditions, and proves that $M^{-1}C$ has eigenvalues independent of h. [Gol1] generalizes this to

$$M = \sqrt{L^2 + 4L}.$$

It is unclear whether these have any use for arbitrary substructures of a given domain, but it seems unlikely. On the other hand, both generalize formally to the case of several horizontal strips.

Another preconditioner which has been suggested is, for two subdomains,

$$M = K_{33} - 2K_{13}^{T}K_{33}^{-1}K_{13}.$$

[Bjo1] contains a full account of the motivation for this and its use.

6. Numerical examples

This section reports some numerical results for cg, ILU and MILU pcg, deflated cg and amg. It is not intended that any method be selected as "best" from these results. Each method has its strengths and weaknesses which these results do not fully reveal. The idea is just to give some feeling for what efficiency can be expected in a few special cases. Also, only methods for which a fair amount of published data exists, or for which the authors have personal experience are reported.

6.1 cg and pcg

The first set of numerical examples are from [Cha2] and deal with the Poisson equation for 2 and 3 dimensional problems with Dirichlet boundary conditions. The domains are the square (0,1) × (0,1) (2d) and the unit cube (3d), with $h = 1/64$ (2d) and $h = 1/16$ (3d). The initial distribution is "random" (distribution unknown) and the termination criterion is $\|\varepsilon^{(k)}\|_2 \leq 10^{-6}\|\varepsilon^{(0)}\|_2$. Computations were all done in single precision on a PDP10. Results are shown in Tables 1 and 2. Flop counts are approximate and setup times are not included.

	cg	ILU	MILU
#itns	180	47	27
#flops/10^{-6}	3.8	1.5	.86

Table 1 (2d)

	cg	ILU	MILU
#itns	47	18	21

#flops/10^{-6}	.95	.49	.57

Table 2 (3d)

For comparison, solution of the 2d problem by a banded Cholesky algorithm would require about 9.3×10^6 flops. Table 1 shows that for the given accuracy, MILU achieves the goal of an order of magnitude speed improvement over the direct method. Relative to cg, Table 1 shows that ILU and MILU respectively require about .39 and .23 of the computer time of the unpreconditioned algorithm. For smaller h, the MILU preconditioned cg algorithm would probably show larger gains relative to the ILU case.

Table 2 shows a rather worse performance for MILU than ILU. Presumably, this can be attributed to the coarseness of the mesh. More evidence is required on this point.

[Cha2] contains comparisons with other preconditioners which we have not discussed, either because they are sensitive to problem dependent parameters or depend on a special mesh structure being available.

A more recent set of calculations was performed by [Con1]. We will give some results from this paper. The square $(0,1) \times (0,1)$ with $h = 1/51$ is used for Table 3, which reports results for the 2d Poisson equation with Dirichlet boundary conditions. The initial approximation was "random" with entries in $[-1,1]$ (presumably uniformly distributed). Iterations were terminated when $\|r^{(k)}\|_\infty \leq 10^{-6}\|r^{(0)}\|_\infty$. Double precision arithmetic on an IBM 3081 was used.

	cg	ILU	MILU
#itns	109	33	23
#flops/10^{-6}	2.7	1.3*	.92
α	-2.0	-1.97	-1.08

Table 3

*This particular run took 1.37 cpu secs.

In the last row of Table 3, α gives the numerically computed exponent of h in the condition number of $M^{-1}K$. The improvement in the condition number of MILU over ILU is essentially the theoretical one, namely a factor of h. On the other hand, Tables 1 and 3 show that the condition number is not the whole story - the performance of ILU is too good to be explained this way. It is the clustering of the eigenvalues of $M^{-1}K$ which is responsible for the good behavior of ILU. [Cha2] and [Con1] contain direct computations of the spectrum of $M^{-1}K$ which support this.

Table 3 shows that ILU and MILU need respectively about .48 and .34 of the standard cg computation time. These are a little worse than the corresponding figures for Table 1. Presumably this is accounted for by the different termination rules, and initial approximations.

[Con1] contains two more difficult computations, one with a piecewise constant material tensor and another with pure Neumann boundary conditions. In the former case, the square with lower left corner at (1/4, 1/4) and upper right corner at (3/4), 3/4) is given the material constant 1000 and the remainder of the original square is given its previous value 1. The rest of the details are as above for the Poisson equation. The latter example also has piecewise constant coefficients although not with wide variations, and a term σu, which ensures unique solvability. σ is relatively small, and piecewise constant. For this case only, $h = 1/43$. This example is due to [Var1], and is used in [Gus2] as well. Tables 4 and 5 show the results.

	ILU	MILU
#itns	47	32
#flops/10^{-6}	1.9	1.3
est. cond. no.	46770	40

Table 4

	ILU	MILU
#itns	74	53
#flops/10^{-6}	2.2	1.6

Table 5

The strong effect of the eigenvalue distribution is evident from Table 4, in which the ratio of the estimated condition numbers is greater than 10^3, and yet the ratio of the number of iterations is less than 1.5. On the other hand MILU is the more efficient algorithm in both cases. Table 5 shows that the third example is the most difficult. Probably the Neumann condition is the major reason for this. A banded

Cholesky algorithm would require about 1.8×10^6 flops for solving the third problem, and Table 5 shows that MILU is nothing like the desired order of magnitude better than this. h is rather large for this example, and greater relative savings would be seen if it was decreased.

[Con1] deals specifically with preconditioning of block tridiagonal matrices such as those arising from 5 point difference formulas with mesh lines parallel to the coordinate axes. Over 30 preconditioners are compared for such problems, some of them quite sophisticated. It is very interesting to note that for problem 3 only one preconditioner was more than twice as good as MILU, for problem 2 none were twice as good, and for the Poisson problem none were more than 3.3 times as good. Moreover, none of them required less storage, most of them needing quite a bit more, as well as more complex programming. This is in addition to the fact that for the majority of the algorithms it is not clear how to correctly apply them to general mesh problems.

6.2 Bercovier's example

In this section, we shall show some results from [Ber1] of a calculation for which negative pivots are encountered in the ILU factorization but for which formal application of pcg gives good results. According to [Ber1] the results obtained this way are "far better" than those using either of the remedies in [Ker1] and [Man1]. The example is that of an orthotropic cantilevered beam, 10 units long by 1 unit deep, in plane strain. The load f is applied at the free end. Discretization is by bilinear elements on a uniform 10 x 3 mesh. Letting 1 and 2 denote the principal directions of the orthotropic material, and ϵ_1, ϵ_2, ϵ_{12} and σ_1, σ_2, σ_{12} the corresponding strains and stresses, the elasticity matrix is defined by

$$D = \begin{bmatrix} 10^3 & 30 & 0 \\ 30 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix}.$$

Three cases are considered, in which the angle β between the x axis and direction 1 is 0°, 45°, and 90°. The negative pivots occur in the latter two cases. The results are given in Tables 6, 7, and 8.

	cg	ILU
#itns	133	7
$\|r\|_2^2$	.00080	.00014

Table 6 (β = 0°)

	cg	ILU
#itns	>300	14
$\|r\|_2^2$	-	.00003

Table 7 (β = 45°)

	cg	ILU
#itns	>300	19
$\|r\|_2^2$	-	.0025

Table 8 (β = 90°)

Unfortunately, [Ber1] does not give much more detail than that reproduced above, so it is difficult to draw specific conclusions. Clearly though, further investigation is warranted.

6.3 Deflation

Relatively few computations have so far been carried out with deflation; here we record its performance against cg on the model Poisson problem with Dirichlet bc. The discrete equations were solved using n square subdomains each containing n nodes of the triangulation for n=9,16,25,36,49,64,81, where n is the number of interior nodes along cross-sections and $N = n^2$. For Table 9, the initial error was a smooth function, and iterations were terminated when the rms error was reduced below 10^{-6} of its initial value.

n =	9	16	25	36	49	64	81
cg itns	25	43	67	96	130	171	216
dcg itns	17	24	29	36	41	45	52
cg time	.004	.02	.06	.18	.45	1	2.0
dcg time	.003	.01	.03	.08	.16	.30	.56

Table 9

The last two lines show times relative to the cg time for n = 64. The first two lines are well fitted by the formulas 2.7n and $6\sqrt{n}$ respectively. The exponent in the second of these is accounted for by the theory in [Nic3]. Thus, the number of iterations is rising much more slowly for the deflated case, as is the overall time. The times given include factoring times for the deflation matrix and other overheads.

Comparison with MILU is not easy, but there seems to be a small time advantage (≈20%) with deflation applied with the above choice of E for the model problem. The E chosen above is not the optimum one for this problem - slightly smaller subdomains are needed for that - but the difference is small. It needs to be mentioned that deflation requires quite a bit less work per iteration than MILU, because most of the operations carried out at the full mesh level are additions. Also, there is no problem with deflation "breaking down" on more general problems. But deflation requires choosing E properly and presently good choices are only known for a restricted range of problems.

6.4 amg

An estimate of the work required to solve the model Poisson problem can be inferred from [Rug1]. There, for $h = 1/64$ a convergence factor $\rho = .054$ is reported, so that to reduce an initial error by 10^{-6} requires

$$6/|\log_{10} .054| \approx 5 \text{ iterations ("cycles").}$$

According to [Rug1] 85 flops/mesh point/iteration are used by the amg algorithm, giving a work count of about

$$1.7 \times 10^6 \text{ flops},$$

to reduce the error below $10^{-6}|\epsilon^{(0)}|$.

Setup costs consist of

1. Computing the interpolation weights
2. Forming the coarse grid operators
3. Construction of the coarse mesh sets C.

The third of these requires more work than the first two combined. [Rug1] gives a figure of about 5 - 9 amg iterations for 1-3. Total storage is about 3 times that required for storage of the problem itself. Thus, for the model problem at least, the setup costs are less than twice the solution cost.

6.5 Comments

As already stated, no attempt should be made to order the methods on the basis of these results. Nevertheless, it is worth pointing out where each method can be expected to do well and not so well. First, it seems probable from the amg and MILU results for the model problem that neither has any clear advantage over the other, since the factor of about 2 in favor of MILU could easily change with the method of accounting. Moreover, if h becomes smaller, there should be an advantage with amg. The setup time for MILU is an order of magnitude less than for amg and, of course, the programming is far less complex. Similar comments apply to deflation vis-à-vis amg, both for speed and storage for the model problem.

In more difficult examples, it seems that amg deteriorates less rapidly than MILU, although the latter does what can be considered a reasonable job in most cases. Again, there may be an advantage in convergence speed for amg when h becomes relatively small, although the setup costs remain to be neutralized. For problems with more general geometry, and for three dimensional problems MILU seems not to have been adequately tested. For three dimensional problems, amg also has not been tested so far.

Appendix

In this appendix, we summarize the properties of the classical conjugate gradient algorithm, and obtain the second order form used in the paper. For solving

$$Ku = f$$

where K is symmetric positive definite and $N \times N$, the algorithm is this:

$$\begin{aligned} &u^{(0)} \text{ is arbitrary}, \quad r^{(0)} = f - Ku^{(0)}, \quad p^{(0)} = r^{(0)} \\ &u^{(k+1)} = u^{(k)} + \gamma_k p^{(k)} \qquad k=0,1,\ldots \\ &p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)} \qquad k=0,1,\ldots \end{aligned} \tag{A1}$$

The coefficients are given by

$$\gamma_k = (r^{(k)},p^{(k)})/\pi_k \qquad \beta_k = -(r^{(k+1)},Kp^{(k)})/\pi_k$$

where

$$\pi_k = (p^{(k)},Kp^{(k)}).$$

The *direction vectors* $p^{(k)}$ and the residuals $r^{(k)}$ satisfy the following orthogonality relations:

$$\begin{aligned} &(1)\ (p^{(k)},Kp^{(j)}) = 0 \qquad j = 0,1,\ldots,k-1 \\ &(2)\ (r^{(k)},r^{(j)}) = 0 \qquad j = 0,1,\ldots,k-1. \end{aligned} \tag{A2}$$

From (A2) it follows that $r^{(N)} = 0$. Some earlier residual can be zero in certain cases, but we do not need to worry about this here. From [Lue1] we have the following bound for the "energy" $E(\epsilon^{(k)}) = (\epsilon^{(k)},K\epsilon^{(k)})$ of the kth error:

<u>Theorem</u>

$$E(\epsilon^{(k)}) \le 4\rho^{2k}E(\epsilon^{(0)})$$

where $\rho = (1 - \sqrt{\kappa})/(1 + \sqrt{\kappa})$ and $\kappa = \lambda_{min}/\lambda_{max}$.

For the model Poisson problem, the eigenvalue ratio $\kappa = O(h^2)$, giving $\rho = 1 - Ch$ as mentioned in section 2.

To obtain the second order form of the algorithm, apparently used first in [Rut1] (see also [Con2]), eliminate $p^{(k)}$ from (A1) using

$$p^{(k)} = \beta_{k-1}p^{(k-1)} + r^{(k)}$$

and

$$\gamma_{k-1}p^{(k-1)} = u^{(k)} - u^{(k-1)}.$$

The result is

$$u^{(k+1)} = \omega_k u^{(k)} + \omega_k' u^{(k-1)} + \alpha_k\omega_k r^{(k)} \qquad \text{(A3)}$$

where $\omega_k + \omega_k' = 1$, $\omega_k' = -\gamma_k\beta_{k-1}/\gamma_{k-1}$ and $\alpha_k\omega_k = \gamma_k$. We define $\beta_{-1} = 0$ for consistency with (A1). From (A3) it follows that

$$r^{(k+1)} = \omega_k r^{(k)} + \omega_k' r^{(k-1)} - \alpha_k\omega_k K r^{(k)}. \qquad \text{(A4)}$$

Since the coefficients in (A4) involve the vectors $p^{(k)}$, we shall determine them afresh, directly from (A4). To do this, note that the property (A2) implies in particular that

$$(r^{(k+1)},r^{(k)}) = (r^{(k+1)},r^{(k-1)}) = 0.$$

In conjunction with (A4) the first implies

$$\alpha_k = (r^{(k)},r^{(k)})/(r^{(k)},Kr^{(k)}). \qquad \text{(A5)}$$

Similarly, the second implies

$$0 = \omega_k'(r^{(k-1)},r^{(k-1)}) - \alpha_k\omega_k(r^{(k-1)},Kr^{(k)})$$
$$0 = \omega_k'(r^{(k-1)},r^{(k-1)}) - \alpha_k\omega_k(Kr^{(k-1)},r^{(k)})$$

by the symmetry of K. This last term can be rewritten using (A4) with k:= k - 1 and taking its inner product with $r^{(k)}$. Then

$$0 = \omega_k'(r^{(k-1)},r^{(k-1)}) + (\alpha_k\omega_k/\alpha_{k-1}\omega_{k-1})(r^{(k)},r^{(k)})$$

from which it follows that

$$\frac{1}{\omega_k} = 1 - \frac{1}{\omega_{k-1}}\,\frac{\alpha_k}{\alpha_{k-1}}\,\frac{(r^{(k)},r^{(k)})}{(r^{(k-1)},r^{(k-1)})}. \qquad \text{(A6)}$$

(A3) with (A5) - (A6) and the initial contition $\omega_0 = 1$ now define the second order form of the algorithm.

It can be proved by induction that the property (A2) is preserved for (A3), (A5) - (A6) with $\omega_0 = 1$. There remains the question of whether the $u^{(k)}$ of the standard form and $u^{(k)}$ of the second order form are indeed equal. In fact, all of the iterates of the two methods are identical (for exact arithmetic). This can also be proved by induction, although the calculations are longer than for the proof of (A2).

References

[Axe1] Axelsson, O. and Barker, V.A., *Finite Element Solution of Boundary Value Problems*, Academic Press, Orlando, Florida, 1984.

[Ber1] Bercovier, M. and Rosenthal, A., Using the conjugate gradient method with preconditioning for solving FEM approximations of elasticity problems, Engineering Computations 3, p77, 1986.

[Bjo1] Bjorstad, P.E. and Widlund, O.B., Iterative methods for the solution of elliptic problems on regions partitioned into substructures, Technical Report 136, Courant Institute of Mathematical Sciences, New York University, 1984.

[Bra1] Bramble, J.H., Pasciak, J.E. and Schatz, A.H., The construction of preconditioners for elliptic problems by substructuring, Mathematics of Computation 47, p103, 1986.

[Bul1] Buleev, N.I., A numerical method for the solution of two-dimensional and three-dimensional equations of diffusion, Mat. Sb. 51, p227, 1960.

[Cha1] Chan, T.F. and Resasco, D.C., A survey of preconditioners for domain decomposition, Research Report YALEU/DCS/RR-414, Department of Computer Science, Yale University, New Haven, Connecticut, 1985.

[Cha2] Chandra, R., Conjugate gradient methods for partial differential equations, PhD dissertation, Department of Computer Science, Yale University, New Haven, Connecticut, 1978.

[Con1] Concus, P., Golub, G.H. and Meurant, G., Block preconditioning for the conjugate gradient method, SIAM J. Sci. Stat. Comput. 6, p220, 1985.

[Con2] Concus, P., Golub, G.H. and O'Leary, D.P., 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 (editors), p309, Academic Press, New York, 1976.

[Dry1] Dryja, M., A capacitance matrix method for Drichlet problem on polygonal region, Numer. Math. 39, p51, 1982.

[Dup1] Dupont, T., Kendall, R.P. and Rachford, H.H., Jr., An approximate factorization procedure for solving self-adjoint elliptic difference equations, SIAM J. Numer. Anal. 5, p559, 1968.

[Dup2] Dupont, T., A factorization procedure for the solution of elliptic difference equations, SIAM J. Numer. Anal. 5, p753, 1968.

[Elm1] Elman, H.C., Iterative methods for large, sparse, nonsymmetric systems of linear equations, PhD dissertation, Department of Computer Science, Yale University, New Haven, Connecticut, 1982.

[For1] Forsythe, G.E. and Wasow, W.R., *Finite-Difference Methods for Partial Differential Equations*, John Wiley, New York, 1960.

[Fra1] Frankel, S.P., Convergence rates of iterative treatments of partial differential equations, Math. Tables Aids Comput 4, p65, 1950.

[Gol1] Golub, G.H. and Mayers, D., The use of pre-conditioning over irregular regions, lecture at 6th Int. Conf. on Computing Methods in Applied Sciences and Engineering, Versailles, December 1983.

[Gun1] Gunzburger, M.D. and Nicolaides, R.A., On substructuring algorithms and solution techniques for the numerical approximation of partial differential equations, to appear in Applied Numerical Mathematics.

[Gus1] Gustafsson, I., A class of first order factorization methods, BIT 18, p142, 1978.

[Gus2] Gustafsson, I., Modified incomplete Cholesky methods, in *Preconditioning Methods: Analysis and Applications*, D.J. Evans (editor), p265, Gordon and Breach, New York, 1983.

[Hac1] Hackbusch, W., *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[Hag1] Hageman, L.A. and Young, D.M., *Applied Iterative Methods*, Academic Press, New York, 1981.

[Hug1] Hughes, T.J.R., Levit, I. and Winget, J., Element-by-element implicit algorithms for heat conduction, Journal of Engineering Mechanics 109, p576, 1983.

[Jam1] Jameson, A., Multigrid algorithms for compressible flow calculations, MAE Report 1743, Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, New Jersey, 1985.

[Ker1] Kershaw, D.S., The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations, Journal of Computational Physics 26, p43, 1978.

[Loh1] Löhner, R. and Morgan, K., An unstructured multigrid method for elliptic problems, to appear in Int. J. Num. Meth. in Eng., 1987.

[Lue1] Luenberger, D.G., *Linear and Nonlinear Programming, 2nd edition*, Addison-Wesley, Reading, Massachusetts, 1984.

[Man1] Manteuffel, T.A., An incomplete factorization technique for positive definite linear systems, Mathematics of Computation 34, p473, 1980.

[Mei1] Meijerink, J.A. and van der Vorst, H.A., An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Mathematics of Computation 31, p148, 1977.

[Mei2] Meijerink, J.A. and van der Vorst, H.A., Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems, Journal of Computational Physics 44, p134, 1981.

[Nic1] Nicolaides, R.A., On the ℓ^2 convergence of an algorithm for solving finite element equations, Mathematics of Computation 31, p892, 1977.

[Nic2] Nicolaides, R.A., On some theoretical and practical aspects of multigrid methods, Mathematics of Computation 33, p933, 1979.

[Nic3] Nicolaides, R.A., Deflation of conjugate gradients with applications to boundary value problems, to appear in SIAM J. Numer. Anal.

[Nou1] Nour-Omid, B. and Parlett, B.N., Element preconditioning using splitting techniques, SIAM J. Sci. Stat. Comput. 6, p761, 1985.

[Ric1] Richardson, L.F., The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam, Philos. Trans. Roy. Soc. London Series A, vol. 210, p307, and Proc. Roy. Soc. London Series A, vol. 83, p335, 1910.

[Rug1] Ruge, J. and Stüben, K., Efficient solution of finite difference and finite element equations by algebraic multigrid, in Proceedings of the Multigrid Conference, Bristol, UK, September 1983.

[Rut1] Rutishauser, H., Theory of gradient methods, in *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems*, M. Engeli, Th. Ginsburg, H. Rutishauser and E. Stiefel, p24, Birkhäuser Verlag, Basel, 1959.

[Sto1] Stone, H.L., Iterative solution of implicit approximations of multidimensional partial differential equations, SIAM J. Numer. Anal. 5, p530, 1968.

[Var1] Varga, R.S., *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

[Var2] Varga, R.S., Factorization and normalized iterative methods, in *Boundary Value Problems in Differential Equations*, R.E. Langer (editor), p121, University of Wisconsin Press, Madison, Wisconsin, 1960.

[You1] Young, D.M., *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

Standard Bibliographic Page

1. Report No. NASA CR-178223 ICASE Report No. 86-78	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle ITERATIVE METHODS FOR ELLIPTIC FINITE ELEMENT EQUATIONS ON GENERAL MESHES		5. Report Date November 1986
		6. Performing Organization Code
7. Author(s) R. A. Nicolaides and Shenaz Choudhury		8. Performing Organization Report No. 86-78
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665-5225		10. Work Unit No.
		11. Contract or Grant No. NAS1-18107
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Contractor Report
		14. Sponsoring Agency Code 505-90-21-01

15. Supplementary Notes

Langley Technical Monitor:
J. C. South

Final Report

16. Abstract

This paper surveys iterative methods for arbitrary mesh discretizations of elliptic partial differential equations. The methods discussed are preconditioned conjugate gradients, algebraic multigrid, deflated conjugate gradients, an element-by-element technique, and domain decomposition. Computational results are included.

17. Key Words (Suggested by Authors(s)) numerical analysis, iterative methods, partial differential equations		18. Distribution Statement 64 - Numerical Analysis Unclassified - unlimited	
19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 43	22. Price A03

For sale by the National Technical Information Service, Springfield, Virginia 22161

NASA-Langley, 1987