An Overview of Preconditioned Iterative Methods for Sparse Matrix Equations

by

Frank W. Letniowski

Research Report CS-89-26
June 1989

An Overview of Preconditioned Iterative Methods for Sparse Matrix Equations

Frank W. Letniowski[1]

June 19, 1989

[1]Department of Applied Mathematics, University of Waterloo.

Abstract

A basic overview is provided toward the solution of large, sparse matrix equations, $Ax = b$ using preconditioned, accelerated iterative methods. The conjugate gradient method of acceleration for symmetric matrices is discussed in some detail, followed by a description of algorithms for nonsymmetric matrices—ORTHOMIN, GMRES, bi-conjugate gradients, and conjugate gradients squared. Methods of preconditioning are discussed with emphasis on DKR factorisations, (modified) incomplete LU factorisations, and reduced systems (red-black ordering), as well as block preconditioning and other re-ordering methods. Some applications of these methods are given at the end.

1 Introduction

The purpose of this paper is to review fast and efficient methods for solving a large, sparse matrix equation

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

where A is an $N \times N$ matrix most of whose entries are 0, and x and b are N–dimensional vectors. The matrix A and vector b are given. The vector x is the unknown solution. Such equations commonly arise when a problem involving differential equations is discretised in order to find an approximate solution.

Some important factors must be considered in the solution of (1). One consideration is the amount of storage needed; in particular, how well a method takes advantage of the sparsity of the matrix. Another consideration is the accuracy of the method. Certain methods are not designed to solve equations involving some types of matrix and may lead to inaccurate results. The final consideration is the efficiency of the method, measured by the time needed for solution. In selecting a method, one must determine a balance between these considerations which achieves the desired results for a particular problem.

(*Throughout the paper, a* footnote *is indicated by a superscripted numeral in the text, a* literary reference *is indicated by a numeral in square brackets, and an* equation reference *is indicated by a numeral in round brackets.*)

2 General Methods

There are two approaches which are used for solution of equation (1). A direct method requires a fixed number of operations depending on the size of the matrix and gives results which are accurate up to round-off error. An iterative method requires a number of repeated steps each of which gives a better approximation to the solution.

2.1 Direct Methods

Direct methods are well-understood and reliable for a wide class of matrices. Many problems however cannot be solved or are expensive to solve directly because of constraints on computational work and storage. How the matrix is structured and how well-conditioned it is are two factors which help determine the suitability of a direct solution technique.

Direct solvers for equation (1) combine a re-ordering of the matrix with gaussian elimination (ref. [17]) or a variation. *Pivoting* (exchanging) rows or columns of a matrix may be done in a manner that improves the results. The steps of gaussian elimination on A have the effect of changing some zero entries of A into nonzero entries. This is called *fill-in*. Re-ordering of the elements of A is designed to reduce the amount of fill-in of A during gaussian elimination thereby reducing both the amount of storage needed and the number of multiplications taken. Of course, the sparsity of the matrix is important in determining how well this can be accomplished. A popular re-ordering method for discretised differential equations is *nested dissection*.

A particular case where direct methods are appropriate is the solution of matrix equations arising from one dimensional discretised problems. These often result in tri-diagonal[1] matrices in which no fill-in occurs.

There are a number of problems in which direct methods are not competitive with good iterative methods. In problems arising in three dimensions, the amount of fill-in becomes large for even re-ordered direct methods. In time-dependent problems, a reasonable approximation of the solution at each time step exists, ie. the solution at the previous time step, which is utilised in an iterative method but not in a direct method. It is very important to consider the type of problem being

[1]a matrix $A = [a_{ij}]$ is called *tri-diagonal* if $a_{ij} = 0$ when $|i - j| > 1$.

solved when deciding on a direct method or an iterative method.

Axelsson and Barker [4] give the work and storage requirements for solving symmetric matrix equations resulting from model finite element problems in two and three dimensions. Here, N is the number of unknowns in the matrix equation. In 2–D, a good direct method[2] requires $O(N^{\frac{3}{2}})$ computational work[3] and $O(N \ln N)$ storage, and a good iterative method[4] requires $O(N^{\frac{5}{4}})$ computational work and $O(N)$ storage. In 3–D problems, the direct method requires $O(N^2)$ computational work and $O(N^{\frac{4}{3}})$ storage, and the iterative method requires $O(N^{\frac{7}{6}})$ computational work and $O(N)$ storage. For 2–D and 3–D problems, iterative methods are superior for large numbers of unknowns.

In general, care must be taken in determining the appropriateness of a direct solution method for sparse matrix equations.

2.2 Iterative Methods

An iterative method consists of repeatedly performing steps to achieve better and better approximations to the solution of (1). The following is a basic summary of the concept of iterative methods. Varga [30] gives more details.

Often when considering a problem, a very rough approximation of the solution is known. An example of this is an initial condition for a time-dependent system. An iterative method uses this result to generate a better approximation which is used in turn to generate an even better approximation. The steps are iterated until the result is within some required tolerance. Of course, for an iterative method to be useful, each step must be relatively cheap compared to a direct method.

The basic iterative approach is described here. Suppose x^k is a given approximate solution at step k. To find a better approximation x^{k+1} at step $k+1$, some approximation, K, of A is selected where $A = K - R$. The choice of K must be nonsingular and such that the matrix equation $Kv = r$ is "easy" to solve. Now,

$$Ax = b$$

[2]nested dissection re-ordering combined with gaussian elimination.

[3]$f(N) = O(g(N))$ if for sufficiently large $\bar{N}$, $f(N) < cg(N)$ for $N > \bar{N}$ and some constant c.

[4]preconditioned conjugate gradient method.

or re-writing

$$Kx = (K - A)x + b.$$

The following approximate procedure results if the successive iterates x^k and x^{k+1} are introduced

$$Kv^{k+1} = r^k$$

where

$$v^{k+1} = x^{k+1} - x^k$$

$$r^k = b - Ax^k \ .$$

Basic Iterative Algorithm

The previous results are used to give the following basic iterative algorithm:

```
$x^0$ given
Calculate $r^0 = b - Ax^0$
Loop over $k = 0, 1, 2, \ldots$ until convergence:
        Solve $Kv^{k+1} = r^k$
        $x^{k+1} = x^k + v^{k+1}$
        Calculate $r^{k+1} = b - Ax^{k+1}$
```

In this method, convergence is achieved when a suitable *norm* of the residual r^k is within the tolerance set for the problem. A good choice of norms for many applications is the *max norm* which equals the maximum absolute value of the components of a vector.

Questions to consider with this basic method are:

- How good is the matrix splitting $K - R$?
  - How well does K approximate A?
  - How easy is it to solve $Kv = r$ and how accurate is the solution?
  - What is the cost in determining K given A?
- Does the algorithm converge?
- How quickly can convergence to desired tolerance be expected?

The question of efficiently splitting the matrix is discussed in a later section.

For convergence, it is required that the *spectral radius*[5] of the matrix $K^{-1}R$ is less than 1 [30]. This is achieved if the splitting of matrix A into $K - R$ is *regular*, ie. K is nonsingular and $K^{-1} \geq 0$ and $R \geq 0$ [6]. An important result is that if A is an M–matrix[7], there exists a splitting of A which is regular, hence the basic iterative routine will converge. In many cases, the numerical solution of differential equations produces matrices which are M–matrices.

The basic iterative method is not very practical. It tends to require many iterations before it converges, hence it is unreliable and expensive. The following sections deal with improving the method to decrease the required number of iterations thereby increasing the efficiency of solution.

[5]The *spectral radius* of a matrix is the largest absolute value of the eigenvalues.

[6]$A = [a_{ij}] \geq 0$ if $a_{ij} \geq 0 \ \ \forall i, j$.

[7]$A = [a_{ij}]$ is an M–matrix if $a_{ij} \leq 0, i \neq j$, A is nonsingular, and $A^{-1} \geq 0$. Sufficient and practical conditions for A to be an M–matrix are A is *irreducible* [30], $a_{ij} \leq 0, i \neq j$; $a_{ii} \geq \sum_{j \neq i, j=1}^{N} |a_{ij}| \ \forall i$ and $a_{ii} > \sum_{j \neq i, j=1}^{N} |a_{ij}|$ for some i.

3 Acceleration

The goal of an acceleration method is to decrease the number of iterations required in the *basic iterative method*.

The main problem with the basic algorithm is the large number of iterations needed, each of which requires a solution of $Kv = r$. This is time-consuming.

The idea behind an acceleration method is to use information gained from previous work to find a better *search direction* for the next approximation. That is, instead of taking $x^{k+1} = x^k + v^{k+1}$, the new choice is $x^{k+1} = x^k + \omega^k q^k$ where q^k is a vector that is the new chosen search direction, and ω^k is a scalar selected to minimise the error in some respect.

There are a number of acceleration methods. They vary according to the strategy taken to choose a new search direction. The focus in this paper is on a class of methods called *conjugate gradient-like* methods. They are popular in practical use.

3.1 The Conjugate Gradient Method

The *conjugate gradient* method (ref. [4]) is applicable to matrices which are *symmetric* and *positive definite*[8]. The strategy of choosing a search direction is to require that all search directions are *conjugate*, ie. if q^i and q^j are search directions at any two different iterations of the method, then the inner product[9] $(Aq^i, q^j) = 0$.

Algorithm:

The k^{th} step of the conjugate gradient algorithm is:

$q^k = r^k + a^k q^{k-1}$ where $a^k = \frac{(r^k, r^k)}{(r^{k-1}, r^{k-1})}$.

$x^{k+1} = x^k + \omega^k q^k$ where $\omega^k = \frac{(r^k, r^k)}{(q^k, Aq^k)}$.

Also $r^{k+1} = b - Ax^{k+1} = (b - Ax^k) - \omega^k Aq^k$

Hence $r^{k+1} = r^k - \omega^k Aq^k$.

[8] a symmetric matrix A is said to be *positive definite* if all eigenvalues of the matrix are positive; equivalently, if for all nonzero vectors x, $x^T Ax > 0$.

[9] Here, $(v, w) = \sum_{i=1}^{N} v_i w_i$, for two vectors v and w.

Derivation:

The matrix equation $Ax = b$ with symmetric matrix A has solution x which minimises the scalar quantity

$$\phi = \frac{1}{2}x^T Ax - b^T x \tag{2}$$

This can be shown by writing (2) in component form, differentiating with respect to x_j (the j^{th} component of x) and setting this to 0. The result is that the vector $[\frac{\partial \phi}{\partial x_j}] = Ax - b$. At $x = x^k$, this quantity is $-r^k$, hence to minimise ϕ, this quantity must tend to 0.

An approach suggested by the above analysis is to select $q^k = r^k$. This method is called *the method of steepest descent*. It turns out to be quite poor in practice.

The *conjugate gradient* method builds on this with the selection of $q^k = r^k + a^k q^{k-1}$. This introduces new parameters a^k. These parameters are determined by requiring that the search directions be *conjugate* which is defined by $(Aq^j, q^k) = 0$ for $j \neq k$. In other words, the search directions are chosen to be independent of one another. The requirement leads to the result $a^k = (r^k, r^k)/(r^{k-1}, r^{k-1})$. See [4] for further information.

The next step is to minimise ϕ for $x^{k+1} = x^k + \omega^k q^k$ with respect to ω^k. This is done by substituting x^{k+1} into ϕ and setting $\frac{\partial \phi}{\partial \omega^k} = 0$ which gives $\omega^k = (r^k, r^k)/(q^k, Aq^k)$. The independence of this quantity from the other search directions is a result of the conjugacy requirement of the search directions.

Properties

A property of the *conjugate gradient* algorithm is that convergence is guaranteed for symmetric, positive definite matrices. In this case, the method converges to the correct solution within N iterations, where N is the number of unknowns in the solution vector. This suggests that larger problems require significantly more iterations to achieve convergence than is practical.

The rate of convergence of the conjugate gradient method depends on the *spectral condition number*[10], $\kappa(A)$. If $p(\epsilon)$ is the number of steps needed to decrease the error by a factor of ϵ, then $p(\epsilon) \leq \frac{1}{2}\sqrt{\kappa(A)}\ln(2/\epsilon) + 1$. In other words, the number of iterations needed to reach convergence is $O(\sqrt{\kappa(A)})$. What this means is that to significantly improve the rate of convergence, it is necessary

[10] $\kappa(A) = |\lambda|_{\max}/|\lambda|_{\min}$, ie. the ratio of the maximum and minimum eigenvalues.

to significantly decrease $\kappa(A)$.

The Preconditioned Conjugate Gradient Algorithm

Preconditioning the matrix equation before applying the conjugate gradient method is a good way to improve the effectiveness of the algorithm [22,19]. The idea of preconditioning is to change the matrix problem to one which the conjugate gradient method handles more effectively.

To precondition, A is split into $A = K - R$ as in the basic iterative method. K is called the *preconditioning matrix*. It must be symmetric for the conjugate gradient method. K is factored into a *Cholesky decomposition* $K = LL^T$ where L is lower triangular and the L^T is the transpose of L. If $AK^{-1} \approx I$, then $L^{-1}A(L^T)^{-1} \approx I$ which means $\kappa(AK^{-1}) = \kappa(L^{-1}A(L^T)^{-1})$ is reduced. This can be significantly better than the original $\kappa(A)$. This translates directly into a significant reduction in the number of iterations.

The effect of preconditioning is to re-write $Ax = b$ as

$$[L^{-1}A(L^T)^{-1}](L^Tx) = (L^{-1}b) \tag{3}$$

or

$$A'x' = b'. \tag{4}$$

A' is symmetric. The preconditioned algorithm is equivalent to the conjugate gradient method applied to equation (4). When A is an M–matrix, the preconditioned method will converge at least as quickly as the unaltered method [22]. How much better the method is depends on the selection of K which is discussed later.

The complete *preconditioned conjugate gradient* algorithm is given below:

```
$x^0$ given
Calculate $r^0 = b - Ax^0$
Loop over $k = 0, 1, 2, \ldots$ until convergence:
        Solve $Kv^k = r^k$.
        $q^k = v^k + a^k q^{k-1}$ where $a^k = \frac{(r^k, v^k)}{(r^{k-1}, v^{k-1})}$ .
        $x^{k+1} = x^k + \omega^k q^k$ where $\omega^k = \frac{(r^k, v^k)}{(q^k, Aq^k)}$ .
        Then $r^{k+1} = r^k - \omega^k Aq^k$.
```

Meijerink and van der Vorst [22] consider two model problems which demonstrate the effectiveness of preconditioning with the conjugate gradient method. With preconditioning, the order of the residual error decreases much more rapidly with each iteration of conjugate gradients than without preconditioning.

In practice, the preconditioned conjugate gradient method has been very successful in the solution of large, sparse, symmetric matrix equations.

Extension to Nonsymmetric Equations

The *conjugate gradient* method can be extended to solve (1) for nonsymmetric matrices by working with the *normal* equation

$$A^T Ax = A^T b \tag{5}$$

$A^T A$ is symmetric positive definite. This permits the use of the conjugate gradient method [13] on equation (5).

The convergence of the method is slowed considerably when $A^T A$ is used. As described previously, convergence depends on the *spectral condition number* of the matrix. The condition number for $A^T A$ is essentially the square of the condition number for A, hence the convergence rate is adversely affected.

For nonsymmetric matrices, it is more appropriate to discuss acceleration techniques which apply to nonsymmetric matrices directly and not to the normal equation (5).

3.2 Methods for Nonsymmetric Matrices

The techniques presented here are designed to solve (1) for nonsymmetric matrices A. These techniques involve ideas similar to the *conjugate gradient* method for symmetric matrices and hence are often given the title *conjugate gradient-like* methods.

In a manner similar to the conjugate gradient method, preconditioning is used to improve convergence rates for these methods. Further details are given later.

Most algorithms for nonsymmetric matrices are designed to work with *positive real* matrices, ie. matrices whose *symmetric parts*[11] are positive definite. Normally, this fact is not known *a priori*,

[11]The symmetric part of A is defined as $\frac{1}{2}(A + A^T)$.

nonetheless these methods tend to give good results for a wide range of problems.

For an approximation K to A, a general form for many iterative algorithms for nonsymmetric matrices is [13]:

```
$x^0$ is given
$r^0 = b - Ax^0$
Iterate for $k = 0, 1, 2, \ldots$ until convergence
        Solve $Kv^k = r^k$
        Determine a new search direction $q^k$
        $x^{k+1} = x^k + \omega^k q^k$
        $r^{k+1} = r^k - \omega^k A q^k$
        where $\omega^k = \frac{(r^k, Aq^k)}{(Aq^k, Aq^k)}$
```

Here, ω^k has been chosen to minimise the sum of the squares of the residuals r^k.

Methods of this form vary in the choice of strategies used to determine the search direction.

In the following sections, four popularly used acceleration methods for nonsymmetric matrices are discussed. They are ORTHOMIN, GMRES, the preconditioned bi-conjugate gradient method, and the conjugate gradient squared method.

3.2.1 ORTHOMIN

ORTHOMIN (ref. [7]) is a popular acceleration method for nonsymmetric matrix equations. It has the general form given in the previous section. Eisenstat, et. al., [13] refer to this method as the *generalised conjugate residual* (GCR) method. The strategy for the choice of the search directions in ORTHOMIN is the requirement of the *orthogonality* of $\{Aq^i\}$; that is $(Aq^i, Aq^j) = 0$ for $i \neq j$. This is accomplished by the following selection for the search direction at step k:

$$q^k = v^k - \sum_{i=1}^{k-1} a_i^k q^i$$

where $a_i^k = \frac{(Av^k, Aq^i)}{(Aq^i, Aq^i)}$.

The orthogonalisation requirement guarantees a convergence result. For matrices A which are *positive real*, the ORTHOMIN procedure will converge to the exact solution within N iterations (in the absence of round-off). That means, however, that for this full ORTHOMIN procedure, it is necessary to store up to N search directions, q^i, and N products Aq^i. Also, a large summation will be required and round-off could accumulate significantly in the search direction calculation.

Improvements

In practice to avoid some of the problems with the full ORTHOMIN procedure, modifications are made to the orthogonalisation requirements.

In ORTHOMIN(m), orthogonality is demanded for the previous m search directions only. In effect, this sets $a_i^k = 0$ for $i = 1$ to $(k - m - 1)$. In doing this, the guarantee of convergence in N iterations is lost; however, in practice, this method works quite well and converges to tolerance with relatively few iterations.

An alternative method is to restart the orthogonalisations every m iterations. Essentially, this means running ORTHOMIN for m steps to get an approximation x^m, then setting $x^0 = x^m$ and running ORTHOMIN for m steps again. This is done until convergence is reached at any of the steps. This method is referred to as restarted ORTHOMIN or GCR(m). The number of iterations required for convergence of this method in practice seems to be approximately the same as for ORTHOMIN(m) for many problems. As well, GCR(m) is less likely to break down than ORTHOMIN(m). The restarted method is, hence, preferable since it requires less work per iteration on average than ORTHOMIN(m).

Convergence

The following analysis is valid for matrices A which are *positive real*; that is M, the symmetric part of A, is positive definite.

Full ORTHOMIN, ORTHOMIN(m) and GCR(m) are all guaranteed to converge for *positive real* matrices A. The rate of convergence is proportional to $\sqrt{1 - \alpha/\beta}$ where $\alpha = [\lambda_{\min}(M)]^2$ and $\beta = \lambda_{\max}(A^T A)$. This result gives the same goals in selection of preconditioners for these methods as for the conjugate gradient method.

What happens if the matrix A is not *positive real*? Convergence of these methods is not guaranteed. To avoid this problem, normally one can find a preconditioning matrix K which creates a *positive real* matrix AK^{-1} or $K^{-1}A$.

3.2.2 GMRES—Generalised Minimal Residual Method

Another possible acceleration method for nonsymmetric matrices is the *generalised minimal residual method*(GMRES) which is described by Saad and Schultz [26].

GMRES is a generalisation of MINRES which is a solution method for symmetric matrices. GMRES is mathematically equivalent to GCR and ORTHODIR [33]. It can be shown that in a computer implementation, GMRES requires approximately half the storage, and approximately one-third fewer operations than GCR. It also has the advantage that it cannot break down whereas GCR may break down with certain types of matrices.

The idea behind these methods is to find, at each step, an optimal search direction to the solution of (1) in a *subspace* of $\Re^N$.

At step k of the method, the subspace over which the search direction is optimised is called the Krylov subspace, $\mathcal{K}_k$. The Krylov subspace is constructed from the vectors $\{A^i v_1\}$, $i = 0, \ldots, (k-1)$; that is, $\mathcal{K}_k = \text{span}\{v_1, Av_1, \ldots, A^{k-1}v_1\}$ [12]. In these methods, $v_1 = r^0/\|r^0\|$ where $\|r^0\|^2 = (r^0, r^0)$. For $k < N$, $\mathcal{K}_k$ defines a k-dimensional subspace of $\Re^N$.

The next step in the procedure is to generate an *orthonormal basis*[13] for $\mathcal{K}_k$ using Arnoldi's method which is based on the Gram-Schmidt orthogonalisation process.

Algorithm: Arnoldi's Method

```
1. Choose $v_1$ with $\|v_1\| = 1$.

2. Iterate for $j = 1, \ldots, k$

        $h_{i,j} = (Av_j, v_i)$ for $i = 1, \ldots, j$
        $u_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i$
        $h_{j+1,j} = \|u_{j+1}\|$
        $v_{j+1} = u_{j+1}/h_{j+1,j}$
```

From this procedure, we can define the $N \times k$ matrix whose columns are the orthonormal vectors

[12] $\text{span}\{x^1, x^2, \ldots, x^k\} = \{x = a_1 x^1 + a_2 x^2 + \ldots + a_k x^k \quad \forall a_i \in \Re\}$

[13] a set of k vectors, $\{v_i\}$, where $\text{span}\{v_i\} = \mathcal{K}_k$ and $(v_i, v_j) = 0$ if $i \neq j$ or 1 if $i = j$.

generated: $V_k = [\ v_1\ v_2\ \ldots\ v_k\ ]$. Then $H_k := V_k^T A V_k$ is a $k \times k$ upper Hessenberg[14] matrix and $H_k = [h_{i,j}]$ where $h_{i,j}$ are generated by the Arnoldi method. For GMRES, define $\bar{H}_k$ which is the $(k+1) \times k$ matrix extended from H_k by adding a $(k+1)^{\text{th}}$ row whose only nonzero element is $h_{k+1,k}$ in the k^{th} position of the row.

Algorithm—GMRES

```
Choose $x^0$
Compute $r^0 = b - Ax^0$ and $v_1 = r^0/||r^0||$
Perform Arnoldi's method until convergence.
Minimise $J(y) = ||\beta e_1 - \bar{H}_k y||$ for $y \in \Re^k$ giving $y^k$
where $\beta = ||r^0||$ and $e_1 = (1, 0, 0, \ldots, 0)^T$.
Then $x^k = x^0 + V_k y^k$.
```

There is a practical way to do the minimisation as each v_k is being calculated that gives the *norm* of the residual at that step, hence determining convergence when desired. This is achieved by factoring $\bar{H}_k$ into $Q_k R_k$ using plane rotations. Refer to Saad and Schultz [26] for details.

Derivation

GMRES is based on using the orthonormal vectors generated by Arnoldi's method to find the "best" approximation to the solution of (1) in the subspace spanned by these vectors. The derivation of GMRES is briefly explained below.

With the definitions given previously, the following is an important result:

$$AV_k = V_{k+1} \bar{H}_k \ . \tag{6}$$

The choice $x^k = x^0 + z$ where $z \in \mathcal{K}_k$ is made such that the norm of the residual is minimised, ie. $||r^k|| = ||b - A(x^0 + z)|| = ||r^0 - Az||$ is minimised for $z \in \mathcal{K}_k$. Now, since $\{v_i\}$ is a basis for $\mathcal{K}_k$, z can be set to $z = V_k y$ where $y \in \Re^k$. The problem has been changed to that of minimising $J(y) := ||\beta v_1 - AV_k y||$ over $y \in \Re^k$ where $\beta := ||r^0||$. Using (6), $J(y) = ||V_{k+1}[\beta e_1 - \bar{H}_k y]||$ and since V_{k+1} is orthonormal in terms of the chosen norm, $J(y) = ||\beta e_1 - \bar{H}_k y||$. This leads to the GMRES algorithm.

[14] $A = [a_{ij}]$ is a $k \times k$ upper Hessenberg matrix if $a_{ij} = 0$ for $j = 1, \ldots, i-2$.

Preconditioning for GMRES

To improve convergence of the method, the matrix equation can be *preconditioned*. For example, GMRES can be used to solve the *right* preconditioned equation

$$A'y = b$$

where $A' = AK^{-1}$ and $y = Kx$. When y is determined, the last step is the calculation of x by

$$x = K^{-1}y \ .$$

For further details, refer to the later sections dedicated to preconditioning methods.

GMRES(m)

In computer applications, a restarted GMRES algorithm is a preferable alternative to the full GMRES algorithm.

The full GMRES method can become unwieldy after relatively few iterations. With the GMRES algorithm, as the number of iterations increases, the storage requirements increase linearly with k and the number of multiplications increase with $\frac{1}{2}k^2N$ where N is the number of unknowns. This is computationally impractical for large matrices.

An alternative to the full GMRES method is to restart the computations after a fixed number of iterations, similar to what is done for the restarted ORTHOMIN procedure. In GMRES(m), the GMRES method is restarted after every m steps until convergence:

```
    Choose $x^0$
    Compute $r^0 = b - Ax^0$ and $v_1 = r^0/||r^0||$
    Perform Arnoldi's method for $m$ steps.
    Minimise $J(y) = ||\beta e_1 - \bar{H}_m y||$ for $y \in \Re^m$ giving $y^m$
    where $\beta = ||r^0||$ and $e_1 = (1, 0, 0, \ldots, 0)^T$.
    Then $x^m = x^0 + V_m y^m$.
    If not converged, choose $x^0 = x^m$ and start over.
```

GMRES(m) is mathematically equivalent to GCR($m - 1$). If the matrix A is positive real, then GMRES(m) is guaranteed to converge. The convergence is dependent on $\sqrt{1 - \alpha/\beta}$ where $\alpha = [\lambda_{\min}(M)]^2$; $\beta = \lambda_{\max}(A^TA)$; and $M = (A^T + A)/2$.

Saad and Schultz compare GMRES(5) with GCR(1) in a particular problem. The results show that the two methods have similar rates of convergence for the problem, in terms of computational work. In a more difficult problem, the comparison between the mathematically equivalent methods GMRES(20) and GCR(19) gives results where GMRES(20) is nearly 25% better in the number of multiplications while requiring about half the storage. The preconditioning method used in these problems resulted in matrices which were not *positive real* and hence the results may not be representative.

3.2.3 The Bi-conjugate Gradient Method

Another popularly-used acceleration method for nonsymmetric matrices is the *bi-conjugate gradient* method (BCG) (ref. [16]).

The *bi-conjugate gradient* method is another extension of the conjugate gradient method for nonsymmetric matrices. It introduces new vectors and enforces *bi-conjugacy* and *bi-orthogonality* conditions on the search direction.

The Preconditioned Bi-conjugate Gradient Method

The following method is based on the assumption that the preconditioning matrix has the form $K = LU$ where L is a *lower triangular* matrix and U is an *upper triangular* matrix.

```
$x^0$ is given.
$r^0 = b - Ax^0$
$\bar{r}^0 = U^T L^{-1} r^0$
Iterate for $k = 0, 1, 2, \ldots$ until convergence
        Solve $Kv^k = r^k$
        Solve $K^T \bar{v}^k = \bar{r}^k$
        $p^k = v^k + a^k p^{k-1}$ where $a^k = \frac{(\bar{r}^k, v^k)}{(\bar{r}^{k-1}, v^{k-1})}$
        $\bar{p}^k = \bar{v}^k + a^k \bar{p}^{k-1}$
        $x^{k+1} = x^k + \omega^k p^k$ where $\omega^k = \frac{(\bar{r}^k, v^k)}{(\bar{p}^k, Ap^k)}$
        $r^{k+1} = r^k - \omega^k Ap^k$
        $\bar{r}^{k+1} = \bar{r}^k - \omega^k A^T \bar{p}^k$
```

Derivation

In this method, ω^k is chosen to satisfy the *bi-orthogonality* condition $(\bar{r}^i, r^j) = 0,\ i \neq j$ and a^k is chosen to satisfy the *bi-conjugacy* condition $(Ap^i, \bar{p}^j) = (A\bar{p}^i, p^j) = 0,\ i \neq j$.

Like the other two methods, BCG converges in at most N iterations for *positive real* matrices. In general, there is no guarantee that the algorithm does not break down nor that it is stable. Also, BCG does not satisfy a strict minimisation property, hence an estimate of the actual error at each step is unknown.

Results given by Mikić and Morse [23] show some interesting behaviour of BCG. They use a complex version of BCG with preconditioning to solve a system of complex differential equations. For many of the examples given, it appears that the error of the approximate solution does not strictly decrease with the number of iterations. Overall, the error decreases quickly when the preconditioning is good but it may increase between consecutive iterations.

Unlike the two previous methods, BCG does not need to be restarted since it does not have the same memory constraint problems. Each iteration, however, BCG requires an extra matrix solve with K^T and an extra matrix–vector multiply with A^T. Assuming that BCG converges in roughly the same number of iterations as ORTHOMIN or GMRES then the computational work of BCG is approximately 2 times greater.

3.2.4 The Conjugate Gradient Squared Method

Another method related to the bi-conjugate gradient method has been proposed by Sonneveld [28]. It is designed to avoid both the extra matrix–vector multiply and the extra matrix solve in the bi-conjugate gradient method. The *conjugate gradient squared* (CGS) method is derived from a polynomial form of the conjugate gradient method.

The algorithm is presented here without derivation. There is also a version of the method with preconditioning.

```
Given $x_0$, and $\tilde{r}_0$
$r_0 = b - Ax_0$
$q_0 = 0 = p_{-1}$
$\rho_{-1} = 1$
Loop over $k = 0, 1, \ldots$ until convergence
        $\rho_k = (\tilde{r}_0, r_k)$
        $\beta_k = p_k / p_{k-1}$
        $u_k = r_k + \beta_k q_k$
        $p_k = u_k + \beta_k (q_k + \beta_k p_{k-1})$
        $v_k = A p_k$
        $\sigma_k = (\tilde{r}_k, v_k)$
        $\alpha_k = p_k / \sigma_k$
        $q_{k+1} = u_k - \alpha_k v_k$
        $r_{k+1} = r_k - \alpha_k A(u_k + q_{k+1})$
        $x_{k+1} = x_k + \alpha_k (u_k + q_{k+1})$
```

Examples given by Sonneveld [28] and by Brussino & Sonnad [8] show the superiority of CGS over BCG; however it is not clear how well CGS performs over other methods. As with BCG, the residual error is not strictly decreasing with the number of iterations. It should be noted that the examples presented by Brussino & Sonnad do not use the same initial vector x_0, hence no conclusion should be made in comparing the results between CGS and GMRES. Further investigation is necessary.

4 Preconditioning

This section is concerned with finding a good preconditioning matrix K for the matrix equation (1). K is selected to improve the iterative method. It is factored into a right and a left preconditioner, $K = K_L K_R$.

The idea behind preconditioning is to solve the equation

$$(K_L^{-1} A K_R^{-1})(K_R x) = (K_L^{-1} b) . \tag{7}$$

Note that $\kappa(K_L^{-1} A K_R^{-1}) = \kappa(AK^{-1}) = \kappa(K^{-1}A)$. Also, if $K_L = I$, ie. the identity matrix, then the residual of an iterative method for (7) is the same as for the original matrix equation.

The effectiveness of a preconditioning matrix is judged by the following criteria:

- How easy is it to compute K in relation to the savings in the iterative method?
- How easy is it to solve $Kv = r$?
- How appropriate is the new matrix $K_L^{-1} A K_R^{-1}$? Does it significantly improve the convergence rate of the iterative routine?
- How much additional storage does K require?

Obviously, A is the best choice for preconditioning in terms of the convergence of an iterative routine but this is completely impractical in terms of the second question above.

For the purposes of an iterative routine, a preconditioning matrix is gauged by how well the rate of convergence is improved. That means the eigenvalues of the preconditioned matrix must be clustered better than the eigenvalues of A. This will be true if $AK^{-1} \approx I$ where I is the N-dimensional identity matrix.

One choice for K is the diagonal of A. Another choice is the lower triangular part of A. These are the Jacobi and the Gauss-Seidel methods respectively. They are very poor choices for preconditioners in comparison to the other methods described here.

The types of preconditioners discussed are DKR factorisations, incomplete LU factorisations (ILU), and modified incomplete LU factorisations (MILU). They are the most popular types of preconditioners in use.

4.1 DKR Factorisation

A DKR (Dupont, Kendall, Rachford) [12] factorisation is similar to *generalised SSOR* [1,2]. The preconditioning matrix K is given the form $K = \tilde{L}\tilde{D}^{-1}\tilde{U}$ where $\tilde{D}$ is a diagonal matrix and $\tilde{L}$, $\tilde{U}$ are lower, upper triangular matrices respectively. If $A = L + D + U$ where D is diagonal and L, U are strictly lower, upper triangular respectively then $\tilde{L}$ and $\tilde{U}$ are given the form $\tilde{L} = L + \tilde{D}$ and $\tilde{U} = U + \tilde{D}$. Classical SSOR makes the choice $\tilde{D} = \omega^{-1}D$ where $0 < \omega < 2$. An obvious benefit of this form for K is that the additional storage requirement is restricted to $\tilde{D}$.

Under the DKR method, $\tilde{D}$ is chosen in a manner such that $K = (L+\tilde{D})\tilde{D}^{-1}(U+\tilde{D})$ agrees with A along the diagonal. $\tilde{D}$ is uniquely determined from the resulting N equations in N unknowns.

From the definition, $K = L + \tilde{D} + U + L\tilde{D}^{-1}U$, thus $\tilde{D} + diag(L\tilde{D}^{-1}U) = D$. Hence, in a recursive manner,

$$\tilde{d}_{ii} = d_{ii} - \sum_{j=1}^{i-1} l_{ij}\tilde{d}_{jj}^{-1}u_{ji} \tag{8}$$

for $i = 1 \ldots N$. The error in the DKR method is given by *offdiag*$(L\tilde{D}^{-1}U)$. To take advantage of the sparsity of A, only the nonzero multiplications in (8) are completed, thus the work required to calculate $\tilde{D}$ is the order of the number of nonzero entries in A.

In actual use, DKR preconditioning works extremely well compared to methods such as Jacobi, Gauss-Seidel and classical SSOR.

4.2 Incomplete LU Preconditioners

An incomplete LU (ILU) factorisation [4,27,22,19] employs the idea of gaussian elimination to provide a reasonable approximation to the original sparse matrix A. This approximation is used as the preconditioning matrix K.

Gaussian elimination results in a factorisation of the original matrix A into lower and upper triangular factors L and U (ref. [10]). If $A = LU$ then the solution of (1) is greatly simplified: one forward solve for y from $Ly = b$ and one backward solve for x from $Ux = y$. The difficulty in the solution of $Ax = b$ comes from finding L and U.

The idea for an incomplete LU factorisation is to find an approximation to A which has form of a lower triangular matrix, L, multiplied by an upper triangular matrix, U. The calculation of

L and U should be *cheap*, ie. much less than the cost for a complete factorisation. Also, L and U should be almost as sparse as A to avoid unmanageable storage requirements.

The method for choosing L and U is conceptually simple—the selection of a *sparsity pattern*[15] for L and U, then the requirement that LU agree with A at the matrix entries given by the chosen sparsity pattern.

The effectiveness of this method for M–matrices was studied by Meijerink and van der Vorst [22]. They prove a theorem guaranteeing the existence of a *regular splitting*[16] $A = LU - R$ for any sparsity pattern of L and U which includes the matrix diagonal. Also, there are unique factors L and U when the splitting is chosen so that R has zero entries where L or U has nonzero entries.

An ILU factorisation is equivalent to performing gaussian elimination on A where updates on entries not in the pre-chosen sparsity pattern are ignored.

For symmetric matrices, an *incomplete Cholesky* decomposition may be used. That is, K takes the form $K = LL^T$ where L is lower triangular. In this manner, the calculation of K results in the use of *square roots*. The square root calculation is avoided by using $K = (L + D)D^{-1}(L^T + D)$ where D is diagonal and L is strictly lower triangular [19].

For a nonsymmetric matrix, it is useful to consider the incomplete factorisation in the form $K = (L + D)D^{-1}(U + D)$ where D is diagonal and L, U are strictly lower, upper triangular respectively. This results in the following general recursive form for ILU factorisation of $A = [a_{ij}]$.

$$\begin{aligned} l_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{kj} && \text{for } j = 1\ldots(i-1) \\ d_{ii} &= a_{ii} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{ki} \\ u_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{kj} && \text{for } j = (i+1)\ldots N \end{aligned}$$

for $i = 1\ldots N$, and where ij is in the chosen sparsity pattern.

As in the DKR method, only the nonzero multiplications in the sums are performed, thus the complexity of calculating K is the order of the number of nonzero entries in A.

[15] *sparsity pattern* refers to the set of matrix entries which are nonzero. That is, ij is in the sparsity pattern of L and U if $L_{ij} \neq 0$ for $i \geq j$, or $U_{ij} \neq 0$ for $i \leq j$.

[16] Remember that a regular splitting is desirable to guarantee convergence of an iterative solution method.

4.3 Modified Incomplete LU Preconditioners

Modified incomplete LU factorisations (MILU) [18,2,4] make use of the *fill-in* generated during the steps of an ILU factorisation. The fill-in terms are added to the diagonal of the matrix in hopes of improving the rate of convergence of the iterative routine.

The form of the MILU matrix is as before, $K = (L + D)D^{-1}(U + D)$ where D is diagonal and L, U are strictly lower, upper triangular respectively. The factorisation is calculated row-by-row with the fill-in error of each row being added to the diagonal entry for that row. This results in the following MILU factorisation routine for $A = [a_{ij}]$.

```
$d_{ii} = a_{ii}$
for $j = 1 \ldots (i-1)$
    if $ij$ in sparsity pattern then
        $l_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{kj}$
    else
        $d_{ii} = d_{ii} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{kj}$
$d_{ii} = d_{ii} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{ki}$
for $j = (i+1) \ldots N$
    if $ij$ in sparsity pattern then
        $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{kj}$
    else
        $d_{ii} = d_{ii} - \sum_{k=1}^{i-1} l_{ik} d_{kk}^{-1} u_{kj}$
```

The modification of the ILU factorisation produces an error matrix $R = A - K$ which has zero rowsums. For a diagonally dominant M–matrix A, MILU preconditioning has an improved rate of convergence over ILU [4] for model problems; however, MILU factorisation of an M-matrix may not result in a preconditioned system that is *positive real*. This may mean that convergence of an iterative routine is lost.

As a summary, MILU preconditioning may result in a faster rate of convergence than ILU for matrices which are diagonally dominant M–matrices. In many cases where these properties are not known, MILU is less likely to produce a convergent iteration routine suggesting that ILU would be more appropriate.

4.4 Selection of Sparsity Pattern

ILU and MILU demand *a priori* selection of a sparsity pattern. The question that is important to ask in this selection is "What is the error in my choice of sparsity pattern?"

As previously stated, ILU and MILU involve incomplete gaussian elimination where updates on certain entries are ignored at each step. Gaussian elimination results in the change of zero entries of the sparse matrix to nonzero entries, entries which are called *fill-in*. The size of the ignored fill-in at each step is the error of an incomplete factorisation. This means that judging the effectiveness of a sparsity pattern involves consideration of the resulting fill-in or lack thereof. To do this, the idea of *level of fill* [27] is introduced.

In the mth step of gaussian elimination, row m, which has elements on and to the right of the diagonal only, is used to eliminate column m below the diagonal. That is,

$$\text{row}(i) = \text{row}(i) - \frac{a_{im}}{a_{mm}}\text{row}(m)$$

for $i = (m+1)\ldots N$. Any fill-in in row i is created from a term involving division by the diagonal entry of row m. For a matrix that has large diagonal entries (diagonal dominance), the fill-in terms created become smaller each time they are divided by a diagonal entry. Thus, one way to anticipate the size of fill-in is to determine how many times a nonzero entry has been divided by a diagonal term. Hence, *level of fill* can be described as the number of times a nonzero term has been divided by a diagonal entry. The result is that a higher level of fill indicates a smaller magnitude for a fill-in term for an M–matrix.

To determine positions of fill-in and levels of fill, a *symbolic* gaussian elimination may be performed where actual values of terms are not calculated. Initially the level of fill $\ell(a_{ij})$, for a matrix entry a_{ij} is defined as

$$\ell(a_{ij}) = \begin{cases} 0 & \text{if } a_{ij} \neq 0 \\ \infty & \text{otherwise.} \end{cases}$$

In the mth step of a complete gaussian elimination, a term a_{ij} would be updated,

$$a_{ij} = a_{ij} - \frac{a_{im}}{a_{mm}} a_{mj} \ .$$

Hence, in the symbolic procedure, the updated level of fill is given by

$$\ell(a_{ij}) = \min[\ell(a_{ij}), \ell(a_{im}) + \ell(a_{mj}) + 1] \ .$$

That is, the minimum of the previous level of a_{ij} and of the level of $a_{im}a_{mj}/a_{mm}$.

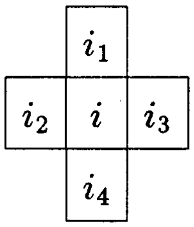
From this analysis, a sparsity pattern for ILU or MILU factorisation can be defined by selecting matrix positions whose level of fill are at most a certain value. For example, the sparsity pattern of a level(2) factorisation includes the entries whose levels of fill are 0, 1, or 2. Note that a level(0) factorisation involves choosing a sparsity pattern that is identical to the sparse structure of the original matrix.

The level of a preconditioning matrix affects an iterative solution routine in important ways. With a high level, a preconditioner is a good approximation to the matrix so the number of iterations will be low; however, the computational work in calculating the preconditioner and in using it each iteration will be comparatively high. Also, the amount of storage required increases with the level of the factorisation. Overall, level(0) or level(1) factorisations work well and are widely used. Simon [27] gives a good comparison of methods for 5 test problems arising from 3-D reservoir simulation and Brussino & Sonnad [8] give a comparison of level(0) and level(1) factorisations over a variety of acceleration techniques.

4.5 Reduced Systems

The size of a matrix problem may be reduced by a suitable ordering of the unknowns. That is, as many as half of the unknowns are implicitly removed from the system leaving a smaller, decoupled matrix equation. This *reduced system* is solved and the remaining unknowns are calculated directly from the result. The new ordering is referred to as *red–black* or *checkerboard* ordering [5,21].

The red–black ordering is illustrated best by considering a nearest neighbour finite difference problem in two dimensions. This results in a five point discretisation.



For an element i with nearest neighbours i_1, i_2, i_3, and i_4, the nonzeros in row i are in positions i, i_1, i_2, i_3, and i_4. The red–black ordering for a rectangular grid is pictured by considering the grid as a checkerboard, then numbering all the red squares first followed by the black squares. This is illustrated below:

15	7	**16**	8
5	**13**	6	**14**
11	3	**12**	4
1	**9**	2	**10**

With a red–black ordering, the matrix equation $Ax = b$ has the form

$$\begin{bmatrix} D_r & A_r \\ A_b & D_b \end{bmatrix} \begin{bmatrix} x_r \\ x_b \end{bmatrix} = \begin{bmatrix} b_r \\ b_b \end{bmatrix} \tag{9}$$

where the subscripts r or b correspond to the elements which are labeled red or black respectively and D indicates a diagonal matrix block.

Performing a step of block elimination gives

$$\begin{bmatrix} I & A'_r \\ 0 & D_b - A_b A'_r \end{bmatrix} \begin{bmatrix} x_r \\ x_b \end{bmatrix} = \begin{bmatrix} b'_r \\ b_b - A_b\, b'_r \end{bmatrix} \tag{10}$$

where $A'_r = D_r^{-1} A_r$ and $b'_r = D_r^{-1} b_r$. This means that the x_b equations have been decoupled from x_r resulting in the reduced system

$$R\, x_b = c \tag{11}$$

where $R = D_b - A_b A'_r$ and $c = b_b - A_b\, b'_r$.

Equation (11) is solved using any of the previously discussed methods to give x_b, then x_r is calculated directly from

$$x_r = b'_r - A'_r x_b \tag{12}$$

Note that the size of the reduced system (11) is half the size of the original system $Ax = b$ for the 5-point discretisation on a rectangular grid.

As shown below, a similar procedure can be carried out for any sparse matrix problem:

1. Select an order for the unknowns which results in the original matrix A_0.
2. Initially label all unknowns black.
3. Pick the first unknown in the ordering and label it red.
4. Repeat over the remaining unknowns, i

   if j is black for all j such that either $a_{ij} \neq 0$ or $a_{ji} \neq 0$ then label i red
   else leave i black.

5. Re-order the unknowns by labeling all the red unknowns first followed by the black unknowns.
6. Carry out the procedure that starts with equation (9).

In general, the size of x_r is less than the size of x_b; however, the same procedure is applicable. For example, a 9-point discretisation arising from finite elements in two dimensions on a rectangular grid results in approximately 1 red element in every 9 elements. In such a case, red–black ordering would not be much of an improvement.

Eisenstat, et al, [14] give results for 5 test problems which use finite differencing in petroleum reservoir simulation. For the test problems which use ORTHOMIN with MILU preconditioning, the red–black ordering consistently produces better results than natural ordering. Calculations by Axelsson and Gustafsson [5] also demonstrate this result. Northrup and Woo [24] show a dramatic improvement of red–black ordering over natural ordering for 2 test problems in 3-D reservoir simulation; however, they do not give results for 3 other problems which have diagonal connections, ie. whose matrices have structures similar to finite element matrices.

4.6 Other Techniques

In this section, two additional techniques for hopefully improving the convergence rate of sparse matrix routines are discussed briefly. Both methods are based on solution techniques which have proven effective for direct methods.

4.6.1 Block Preconditioning

A block method for preconditioning involves dividing the original sparse matrix into blocks (submatrices) which are themselves sparse [11,3,14]. An incomplete factorisation is performed using the blocks instead of individual matrix entries.

When a matrix is divided into sparse blocks, a direct solution can be given recursively. This direct solution involves inversion of the diagonal blocks. A sparse block which is inverted generally becomes a full matrix, but for certain matrices which sometimes arise in two dimensional modeling, these sparse block matrices can be inverted efficiently.

In cases where a direct solution is not practical, a similar technique can be applied. In this case, an *incomplete* inversion is performed. That is, a sparse approximation to the inversion of the sparse blocks is calculated. Details are found in the references [11,3,14]. Common examples are *nested factorisation*, INV and MINV (modified INV). Test results given by Eisenstat, et al, [14] favour a reduced system MILU over the block methods in terms of speed and robustness. This is achieved at the expense of more storage; however, all methods retain $O(N)$ storage, where N is the number of unknowns.

4.6.2 Re-ordering Methods

In a previous section, a re-ordering method (red–black ordering) was introduced as a means to reduce a sparse matrix system to a smaller system. This section introduces re-ordering techniques with the goal of improving the rate of convergence of an iterative method without changing the size of a system.

Re-ordering methods have been highly successful in improving direct methods. Two such methods, *quotient minimum degree* and *nested dissection* [17], are useful in reducing the amount of fill-in generated by gaussian elimination on sparse matrices. This reduction in fill-in improves the storage requirements and the computational effort of gaussian elimination.

These methods can be applied to iterative methods as well. The hope is that applying one of these re-ordering techniques on a matrix before incomplete factorisation is performed will result in a better preconditioning matrix. Simon [27] compared the two re-ordering mentioned above to methods with natural ordering for 5 problems. He concluded that the re-ordering methods did not produce any gains. This should not be unexpected since only the *amount* of fill-in is affected and not the *size*. This indicates that re-ordering methods applicable to direct methods do not necessarily produce properties useful to iterative methods.

Re-orderings exist which are shown to improve convergence rates of iterative solution methods. In fact, for 2–D symmetric problems with grids similar to that given in the red–black ordering discussion, labelling the unknowns *diagonally* on the grid has been shown by Behie and Forsyth [6] to be superior to the natural ordering (labelling the unknowns row by row). Diagonal ordering is fully explained by Watts [31] for 2–D and 3–D problems.

These improvements suggest that general techniques may exist to find re-orderings which improve iterative routines based on the type of incomplete factorisation and on the structure of a sparse matrix. This is an area of current research.

5 Applications

Preconditioned accelerated iterative methods are applicable whenever large, sparse matrix equations arise. By far, the largest current utilisation of these techniques is with the numerical solution of differential equations particularly for problems in 2 or 3 space dimensions. These problems are *discretised* by some method such as *finite elements* or *finite differencing* resulting in matrices which are used to solve for thousands or millions of unknowns.

In some fields, these iterative methods have established themselves as the preferred approach. In fact, for example, some of the techniques presented were originally proposed in the context of numerical solution in petroleum reservoir simulation [31,7,21]. Another well-established field is semiconductor device modelling [8,25]. In other fields these approaches are just beginning to be utilised. Applications occur in fields as diverse as astrophysics [20], nuclear engineering [23], tidal calculations [32] and fluid mechanics [9].

One particularly important application is the numerical solution of systems of partial differential equations which result in nonlinear systems of equations. The nonlinear system is usually solved with a Newton–Raphson technique which generates a large, sparse Jacobian system to solve. With a system of partial differential equations, each solution node will have more than one unknown. An efficient way to solve this is to allow the sparse matrix to consist of dense blocks corresponding to the multiple unknowns at each node. That is, an entry a_{ij} of A is actually an $Neq \times Neq$ matrix where Neq is the number of partial differential equations. All the methods presented carry through in an analogous way for these dense blocks [7].

Many other fields could benefit from the use of preconditioned iterative methods. Large problems which once were impractical to solve can now be tackled effectively.

6 Conclusions

The preconditioned accelerated iterative methods presented in this paper are powerful tools for the numerical solution of large, sparse matrix equations. These methods compare very favourably with classical iterative methods such as Jacobi, Gauss-Seidel, alternating direction implicit (ADI), block successive over-relaxation (BSSOR) [19], successive line over-relaxation (SLOR), and the strongly implicit procedure (SIP) [22].

These favourable results support the use of these methods in a variety of fields ranging from petroleum reservoir simulation to semiconductor device modelling to nuclear engineering. As they become better known, these iterative techniques will be applied in more and more fields.

Although much has already been accomplished, promising new work is still being carried out with the improvement of preconditioning and re-ordering techniques, in the use of vector processing [29] and parallel processing [15] with the algorithms, and with the improvement of the stability and convergence of acceleration routines.

Acknowledgements

The author would like to thank Dr. R. B. Simpson for his comments and Dr. P. A. Forsyth for his helpful suggestions and guidance in preparing this document. This work was supported by the Information Technology Research Centre which is funded by the Province of Ontario.

References

[1] O. Axelsson. A generalized SSOR method. *BIT*, 13:443–467, 1972.

[2] O. Axelsson. On conjugate gradient methods for large sparse systems of linear equations. In J. Hinze, editor, *Numerical Integration of Differential Equations and Large Linear Systems*, number 968 in Lecture Notes in Mathematics. Springer-Verlag, 1982.

[3] O. Axelsson. A general incomplete block-matrix factorization method. *Lin. Alg. Appl.*, 74:179–190, 1986.

[4] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems: Theory and Computation.* Academic Press, Inc., Orlando, 1984.

[5] O. Axelsson and I. Gustafsson. Note: On the use of preconditioned conjugate gradient methods for red–black ordered five-point difference schemes. *J. Comp. Phys.*, 35:284–289, 1980.

[6] A. Behie and P. Forsyth Jr. Comparison of fast iterative methods for symmetric systems. *IMA J. Num. Anal.*, 3:41–63, 1983.

[7] A. Behie and P. K. W. Vinsome. Block iterative methods for fully implicit reservoir simulation. *Soc. Pet. Eng. J.*, pages 658–668, October 1982.

[8] G. Brussino and V. Sonnad. A comparison of direct and preconditioned iterative techniques for sparse, unsymmetric systems of linear equations. *Int. J. Num. Meth. Eng.*, 28:801–815, 1989.

[9] G. F. Carey, K. C. Wang, and W. D. Joubert. Performance of iterative methods for newtonian and generalized newtonian flows. *Int. J. Num. Meth. Fluids*, 9:127–150, 1989.

[10] W. Cheney and D. Kincaid. *Numerical Mathematics and Computing.* Brooks/Cole Publishing Company, Monterey, second edition, 1985.

[11] P. Concus, G. H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Stat. Comp.*, 6:220–252, 1985.

[12] T. Dupont, R. P. Kendall, and H. H. Rachford, Jr. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal.*, 5:559–573, 1968.

[13] S. C. Eisenstatt, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 20:345–357, 1983.

[14] S. C. Eisenstatt, H. C. Elman, and M. H. Schultz. Block-preconditioned conjugate-gradient-like methods for numerical reservoir simulation. *Soc. Pet. Eng. Res. Eng.*, pages 307–312, February 1988.

[15] H. C. Elman. Approximate schur complement preconditioners on serial and parallel computers. *SIAM J. Sci. Stat. Comp.*, 10:581–605, 1989.

[16] R. Fletcher. Conjugate gradient methods for indefinite systems. In A. Dold and B. Eckmann, editors, *Numerical Analysis Dundee 1975*, number 506 in Lecture Notes in Mathematics. Springer-Verlag, 1976.

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

[18] I. Gustafsson. On modified incomplete factorization methods. In J. Hinze, editor, *Numerical Integration of Differential Equations and Large Linear Systems*, number 968 in Lecture Notes in Mathematics. Springer-Verlag, 1982.

[19] D. S. Kershaw. The incomplete Cholesky–conjugate gradient method for the iterative solution of systems of linear equations. *J. Comp. Phys.*, 26:43–65, 1978.

[20] W. Kley. Radiation hydrodynamics of the boundary layer in accretion disks. 1. Numerical methods. *Astron. Astr.*, 208:98–110, 1989.

[21] A. E. McDonald and R. H. Trimble. Efficient use of mass storage during elimination for sparse sets of simulation equations. *Soc. Pet. Eng. J.*, pages 300–316, August 1977.

[22] 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. Comp.*, 31:148–162, 1977.

[23] Z. Mikić and E. C. Morse. The use of a preconditioned bi-conjugate gradient method for hybrid plasma stability analysis. *J. Comp. Phys.*, 65:154–185, 1985.

[24] E. J. Northrup and P. T. Woo. Application of preconditioned conjugate-gradient-like methods in reservoir engineering. *Soc. Pet. Eng. Res. Eng.*, pages 295–301, February 1988.

[25] S. J. Polack, C. den Heijer, H. A. Schilder, and P. Markowich. Semiconductor device modelling from the numerical point of view. *Int. J. Num. Meth. Eng.*, 24:763–838, 1987.

[26] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.

[27] H. D. Simon. Incomplete LU preconditioners for conjugate-gradient-type iterative methods. *Soc. Pet. Eng. Res. Eng.*, pages 302–306, February 1988.

[28] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 10:36–52, 1989.

[29] H. A. van der Vorst. The performance of FORTRAN implementations for preconditioned conjugate gradients on vector computers. *Parallel Comp.*, 3:49–58, 1986.

[30] R. S. Varga. *Matrix Iterative Analysis.* Prentice-Hall, Inc., Englewood Cliffs, 1962.

[31] J. W. Watts III. A conjugate gradient–truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Soc. Pet. Eng. J.*, pages 345–353, June 1981.

[32] P. Wilders, Th. J. van Stijn, G. S. Stelling, and G. A. Fokkema. A fully implicit splitting method for accurate tidal calculations. *Int. J. Num. Meth. Eng.*, 26:2707–2721, 1988.

[33] D. M. Young and K. C. Jea. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Lin. Alg. Appl.*, 34:159–194, 1980.