PRECONDITIONED KRYLOV SUBSPACE METHODS FOR CFD APPLICATIONS *

Y. SAAD

The numerical solution of fluid flow problems gives rise to linear systems that can be rather challenging for iterative methods. In this paper we compare a number of standard preconditioning approaches to solve these problems. We test two accelerators, GMRES and DQGMRES, combined with a few threshold based preconditioners such as ILUT and approximate inverse techniques, on a number of linear systems arising from various models.

KEY WORDS: Iterative methods, Krylov subspaces, GMRES, DQGMRES, ILUT, ILUS, preconditioning, Approximate inverses

1. INTRODUCTION

In the past it was often customary to solve the linear systems that arise from Computational Fluid Dynamics applications by direct methods. These methods have the advantage of being 'predictably reliable' and for this reason they have been preferred over iterative methods in industrial applications. However, with the increase of three-dimensional models as well as models that incorporate more complex phenomena, iterative methods are gaining ground.

A major stumbling block in the acceptance of iterative methods is the difficulty in getting general purpose preconditioners. Indeed, it is often the preconditioner rather than the accelerator which can make the difference between success and failure. Thus, a common cause of failure of iterative techniques is the 'instability' of the ILU factorization, a term which is sometimes used to mean that the norm of $(LU)^{-1}$ can be extremely large. This is caused by the long recurrences in the forward and backward triangular solutions when the preconditioning operations are applied. In such situations, the accelerator will generally fail on the preconditioned system which may well have a condition number that is much worse than that of the un-preconditioned system.

The standard and inexpensive approaches, such as ILU(0), which work quite well for elliptic problems, have a great rate of failure in the indefinite case. It is often the case that this incomplete factorization does not exist and when it does exist, then it is generally either unstable, or too inaccurate to yield a satisfactory rate of convergence. One alternative is to use a more accurate factorization, such as ILUT [15]. This yields a better ILU factorization and works well for many cases. However, it is still prone to instability. To remedy this, a pivoting variant of ILUT called ILUTP can be used. We discuss one such variant in this paper. Another alternative is to attempt to solve the normal equations by an Incomplete Choleski preconditioned Conjugate Gradient method. The IC(0) factorization of positive definite matrices does not necessarily exist but one can shift the matrix slightly by adding a multiple of the identity matrix [13,10]. However, the condition number of

* This work was supported by NASA Grant NAG2-904 and by NSF grant number CCR-9214116

the coefficient matrix is typically very high and ICCG(0) may not perform well for these cases. Another class of preconditioners is based on using approximate inverse techniques, see, e.g., [8,3,9,12]. These are not prone to instability since their corresponding preconditioning operations do not involve linear recurrences. However, they tend to be more expensive to compute and to require more storage than the ILU counterparts.

The outline of the paper is as follows. In the next section we present two Krylov subspace algorithms which emphasize variable preconditioners. Section 3 is an overview of preconditioners for indefinite problems. In section 4 we present some numerical experiments and a few tentative conclusions are drawn in Section 5.

2. PRECONDITIONED KRYLOV SUBSPACE METHODS

Iterative techniques based on Krylov subspace projection coupled with suitable preconditioners are currently considered to be the best compromise between efficiency and robustness. In addition to their advantage over direct methods, in terms of memory and computational cost, iterative methods are also attractive because of the simplicity with which they can be adapted to high performance computers. There are two ingredients in the use of a preconditioned Krylov subspace approach. First, the original linear system

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

is preconditioned by, for example, transforming it into the "right-preconditioned" equivalent system

$$AM^{-1}(Mx) = b. \tag{2}$$

where the preconditioned matrix M has the property that it is not too expensive to compute $M^{-1}v$ for an arbitrary v. Thus, the system $AM^{-1}y = b$ is solved for the unknown $y \equiv Mx$, and the final x result is obtained through the post-transformation $x = M^{-1}y$. One can also use left-preconditioning,

$$M^{-1}Ax = M^{-1}b$$

which requires a pre-transformation of the right-hand-side, or the initial residual. The main difference between these two approaches is that in the right-preconditioned case, the actual residual norm is available at each step of the iterative process whereas in the second case only the preconditioned residual is available barring any additional computations.

The second ingredient in the method is the "accelerator", typically a conjugate gradient like method, based on projections on Krylov subspaces. Here we will describe two techniques which can be used when the preconditioning operations varies from step to step, a property which can be very useful. The first is a method presented in [16] and known as FGMRES. This variant is derived by observing that in the last step of the standard GMRES algorithm [17], the approximate solution x_m

is formed as

$$x_m = x_0 + \sum_{i=1}^{m} \alpha_i M^{-1} v_i$$

Here, the v_i's are the Arnoldi vectors, M the preconditioner, x_0 the initial guess and m the dimension of the Krylov subspace. This is a linear combination of the preconditioned vectors $z_i = M^{-1}v_i, i = 1, \ldots, m$. Since these vectors are all obtained by applying the same preconditioning matrix M^{-1} to the v's, we need not save them. We only need to apply M^{-1} to the linear combination of the $v's$. If the preconditioner varies at every step, then we need to save the 'preconditioned' vectors $z_i = M_i^{-1}v_i$ and use them instead of $M^{-1}v_i$ when computing the above linear linear combination. The resulting 'flexible' variant of GMRES is described below.

ALGORITHM 1 . **Flexible GMRES (FGMRES)**

1. **Start:**
   Choose x_0 and a dimension m of the Krylov subspaces.
   Define $\bar{H}_m = \{h_{i,j} := 0\}_{i=1,\ldots,m+1;\ j=1,\ldots,m}$.
2. **Arnoldi process:**
   Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ *and* $v_1 = r_0/\beta$.
   For $j = 1, ..., m$ *do*
      Compute $z_j := M_j^{-1} v_j$
      Compute $w := Az_j$
      For $i = 1, \ldots, j$, *do*
         $h_{i,j} := (w, v_i)$
         $w := w - h_{i,j} v_i$
      Enddo
      Compute $h_{j+1,j} = \|w\|_2$ *and* $v_{j+1} = w/h_{j+1,j}$.
   Enddo
   Define $Z_m := [z_1, ..., z_m]$.
3. **Form the approximate solution:**
   Compute $x_m = x_0 + Z_m y_m$ *where*
   $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ *and* $e_1 = [1, 0, \ldots, 0]^T$.
4. **Restart:**
   If satisfied stop, else set $x_0 \leftarrow x_m$ *and goto 2.*

The Arnoldi loop simply constructs an orthogonal basis of the preconditioned subspace $\text{Span}\{v_1, AM_1^{-1}v_1, \ldots, AM_{m-1}^{-1}v_{m-1}\}$ by a modified Gram-Schmidt process, in which the new vector to be orthogonalized is defined from the previous vector in the process.

Note that if $M_j = M$ for $j = 1, \ldots, m$ then the method is equivalent to the standard GMRES algorithm, right-preconditioned with M. The approximate solution x_m obtained from this modified algorithm minimizes the residual norm $\|b - Ax_m\|_2$ over $x_0 + \text{Span}\{Z_m\}$, [16]. In addition, if at a given step k, we have $Az_k = v_k$

(i.e., if the preconditioning is 'exact' at step k) and if the $k \times k$ Hessenberg matrix $H_k = \{h_{ij}\}_{i,j=1,\ldots,k}$ is nonsingular then the approximation x_k is exact.

There are many possible applications of the added flexibility provided by FGMRES. In our context, we would like to be able to use any *secondary* iterative procedure as a preconditioner, a feature which is quite helpful in domain decomposition methods or in any parallel computing implementation. FGMRES even allows the inner preconditioning steps to be completely asynchronous, a feature which can help minimize communication and synchronization costs in a parallel approach.

A second variant of the GMRES algorithm described in [18] is the Direct Quasi Generalized Minimal Residual algorithm, or DQGMRES. This algorithm also has the feature of being flexible. It is based on the following idea. Instead of orthogonalizing the Krylov vectors, we replace the Arnoldi loop by an 'incomplete orthogonalization' process,

ALGORITHM **2** . **Incomplete Arnoldi Process:**

For $j = 1, ..., m$ *do*
  Compute $z_j := M_j^{-1} v_j$
  Compute $w := A z_j$
  For For $i = \max\{1, j-k+1\}, \ldots, j$ *do*
    $h_{i,j} := (w, v_i)$
    $w := w - h_{i,j} v_i$
  Enddo
  Compute $h_{j+1,j} = \|w\|_2$ *and* $v_{j+1} = w/h_{j+1,j}$.
Enddo

Thus, the only difference with the full Arnoldi orthogonalization is that at each step the current vector is orthogonalized only against the k previous ones instead of all of them. The vectors generated by the above algorithm are known to be 'locally' orthogonal to each other, in that $(v_i, v_j) = \delta_{ij}$ for $|i-j| < k$ The matrix $\bar{H}_m$ becomes banded upper Hessenberg. As a result of this it can be shown that the approximate solution at step j can be updated from the approximate solution at step $j-1$ via a recurrence of the form

$$\begin{aligned} p_j &= \frac{1}{r_{jj}} \left[v_j - \sum_{i=j-k+1}^{j-1} r_{ij} p_i \right] \\ x_j &= x_{j-1} + \gamma_j p_j \end{aligned}$$

in which the scalars γ_j and r_{ij} are obtained recursively from the Hessenberg matrix $\bar{H}_j$. In terms of memory usage note that we now have to keep only the k most current v_i directions which are needed in the incomplete orthogonalization procedure and the k most recent 'search directions' p_i. As a result the algorithm does not need to be restarted as the classical GMRES and the Flexible GMRES.

An attractive characteristic of DQGMRES is that it is also *flexible*. The principle

is similar to that of FGMRES. In both cases we must compute the vectors $M_j^{-1}v_j$ and in the case of FGMRES, we need to save these vectors which requires extra storage [16]. In the case of DQGMRES, the preconditioned vectors only affect the update of the vector p_j in the preconditioned version of the above formula,

$$p_j = \frac{1}{r_{jj}} \left[M_j^{-1} v_j - \sum_{i=j-k+1}^{j-1} r_{ij} p_i \right].$$

As a result, $M_j^{-1}v_j$ can be discarded immediately after it is used. We can simply overwrite it onto the space used for p_j. We should point out that DQGMRES is similar in nature to the GMRESR family of flexible preconditioning algorithms introduced by Van der Vorst and Vuik [19]. We omit the full description of the algorithm; for details see [18]. It is our experience that DQGMRES is more robust than the standard restarted GMRES algorithm, in that it is less prone to the stagnation phenomenon which occurs in the indefinite case.

3. PRECONDITIONING TECHNIQUES

As was stated above a critical component in the success of iterative methods is the preconditioner. For a system that is poorly preconditioned the iterative process may require so many steps to converge that a direct solver may actually perform better. Preconditioning a linear system is typically a difficult task, except in the traditional elliptic-dominated, one variable-per mesh point cases. Unfortunately, the best known preconditioning techniques have been developed mainly with these problems in mind. In this section we give an overview of standard preconditioners and present some alternatives that are more suitable for indefinite systems.

3.1. Standard preconditioners

We would like to give a quick overview of commonly used preconditioners. In the simplest case, M is simply the diagonal or block diagonal of A. This is referred to as Jacobi (or diagonal), or block Jacobi (block diagonal) preconditioning and is effective only in special cases, e.g., for transient solutions. The SSOR preconditioner is defined by

$$M_\omega = (D - \omega E) D^{-1} (D - \omega F)$$

in which D is the diagonal of A, $-E$ its strict lower part, and $-F$ its strict upper part. In the late 60's and early 70's the idea came about to use an M which has the same form as above with $\omega = 1$ but with a D that is defined recursively to ensure that the diagonal elements of M and A are the same. This lead to the ILU(0) preconditioner in the special case of 5-point matrices. More generally, if we denote by $NZ(A)$ the nonzero structure of A, i.e., the set of all pairs (i, j) such that $a_{ij} \neq 0$ then ILU(0) can be described as follows.

ALGORITHM **3** . *ILU(0)*

For $i = 1, \ldots, N$ *Do:*
  For $k = 1, \ldots, i-1$ *and if* $(i,k) \in NZ(A)$ *Do:*
    Compute $a_{ik} := a_{ik}/a_{kj}$
    For $j = k+1, \ldots$ *and if* $(i,j) \in NZ(A)$, *Do:*
      compute $a_{ij} := a_{ij} - a_{ik}a_{k,j}$.
    EndDo
  EndDo
EndDo

Notice that this is nothing but an (i,k,j) version of Gaussian elimination [7] which is essentially restricted to the $NZ(A)$ part of the matrix. This algorithm can be generalized to any preset nonzero pattern. In particular, one can classify the fill-ins by assigning them a level which is defined from the parents which generated the element in the elimination [20]. For diagonally dominant matrices, the higher the level-of-fill the smaller the element. Once the level of fill of each element is defined we can execute an algorithm similar to the one above, in which $NZ(A)$ is replaced by $NZ_p(A)$ which is the set of all elements whose level-of-fill does not exceed p. This defines the ILU(p) factorization.

3.2. ILUT and ILUTP

The elements that are dropped in the ILU(p) factorizations depend only on the pattern of A and not on the values. The property exploited here is that the larger the level of fill, the smaller the elements, which results in the dropping of smaller elements in the ILU(p) factorization. However, this property is no longer true for non-diagonally dominant matrices. Another strategy altogether is to use the same general structure of the ILU factorization, namely the i,k,j variant of Gaussian Elimination, and to drop elements according to their magnitude. One such strategy defined in [15], was referred to as ILUT (ILU with threshold).

ALGORITHM **4** . *ILUT*(p, ϵ)

For $i = 1, \ldots, N$ *Do:*
  Compute $\epsilon_i := \epsilon \|a_{i,:}\|_2$
  For $k = 1, \ldots, i-1$ *and if* $a_{i,k} \neq 0$ *Do:*
    Compute $a_{ik} := a_{ik}/a_{kj}$
    If $|a_{ik}| \geq \epsilon_i$ *Then*
      For $j = k+1, \ldots$ *Do:*
        compute $a_{ij} := a_{ij} - a_{ik}a_{k,j}$.
        If $|a_{ij}| \leq \epsilon_i$ *then* $a_{ij} := 0$
      EndDo
    EndIf
    Keep p *largest elements in* L*-part of* $a_{i,:}$
      and p *largest elements in* U*-part of* $a_{i,:}$;
  EndDo
EndDo

An advantage of this algorithm is that the amount of fill-in is controlled. When $\epsilon = 0$, then the higher the parameter p, the more accurate the factorization. Reordering for reducing fill-in can help improve the quality of the factorization, and we refer to [4] and [5] for similar experiments performed with the ILU(0) factorization.

The ILUT factorization can be used to solve indefinite problems, and does work for a much broader set of matrices than ILU(0). However, there may be problems computing the factorization itself, because a zero pivot can be encountered. An obvious solution for cases where ILUT fails to yield a good incomplete factorization is to perform some form of pivoting. We can perform a form of pivoting that leads to an algorithm similar, in simplicity and cost, to ILUT. Because of the structure of ILUT, it is not very practical to perform row pivoting. However, a column pivoting variant is not too difficult to develop. The algorithm uses a permutation array *perm* as well as its reverse permutation array to hold the new orderings of the variables. These arrays are updated at each step. Once the most significant element in a row is selected, we define the new i-th variable and update these two permutation arrays. The matrix elements of L and U are kept in their original labeling. At the end of the process, we apply the permutation to all elements of A as well as L/U.

The algorithm corresponding to the above modification will be termed ILUTP (ILUT with Pivoting). The complexity of the ILUTP procedure is virtually identical with that of ILUT. In addition, our implementation provides for a few possible options. A tolerance parameter called *permtol* is included to help determine whether or not to permute variables. Furthermore, the user may elect to perform the pivoting only within diagonal blocks of a fixed size. The size *mbloc* of the blocks within which to perform the permutations must therefore be provided. If the user does not wish to restrict permutations to take place within diagonal blocks, then the value of *mbloc* to be entered should be any number $\geq n$.

For the more difficult matrices, we found that it was usually a good strategy to always apply a scaling to all the rows (or columns), e.g., so that their 2-norms are all equal to 1; to use a small drop tolerance (e.g., $\epsilon = 10^{-4}$); as well as a large fill -in parameter (e.g., $lfil = 20$). As an illustration we show in Table 1 a few results with both ILUT and ILUTP on some of the harder problems in the Harwell-Boeing collection. In these tests, the matrices are read and an artificial right-hand-side is computed so that the solution is known, then GMRES(20) is used to solve the system with the preconditioner shown. Recall that the parameters in ILUT are the drop tolerance ϵ and the amount of fill-in allowed in L and in U.

A failure sign in the table means that the method was not able to reduce the residual norm by a factor of 10^{-7} in 300 steps. The above matrices are all from the Harwell-Boeing collection, except the matrix MAT1 which is a matrix obtained from a finite volume code applied to the Navier Stokes. In spite of its nice numerical properties, which characterize matrices arising from finite volume approximations, the system is not too easy to solve because of the inadequate initial ordering. A plot of the pattern of the matrix confirms this. In this last case, the Reverse Cuthill Mc-Kee ordering (RCMK) does an excellent job at reducing the number of iterations, compared with the original matrix. In fact, column pivoting in this situation makes the preconditioner poorer not better. There are two conclusions

	ILUT$(10,\epsilon)$	ILUT$(20,\epsilon)$	ILUTP$(10,\epsilon)$	ILUTP$(20,\epsilon)$
BP1000	Fail	Fail	25	14
FS7603	Fail	70	fail	32
SHERMAN5	17	10	17	10
WEST0989	Fail	Fail	179	20
MAT1	57	34	48	35
MAT1+RCMK	13	10	140	70

TABLE 1: Iterations to converge with GMRES(20) and ILUT and ILUTP for a few test matrices

from this experiment and other similar ones. First, a poor initial ordering of the matrix can cause the preconditioner to be very ineffective and it may be beneficial to apply some reordering to the original matrix. Second, pivoting in ILU, does not always help. It does seem to help tremendously for those matrices that have very poor diagonal dominance – and not too much for other cases. The reasons for this are still unclear and remain to be investigated.

3.3. ILUS

A different type of incomplete factorization can be derived from another form of Gaussian elimination. Consider the sequence of matrices

$$A_{k+1} = \begin{pmatrix} A_k & v_k \\ w_k & \alpha_k \end{pmatrix}$$

If we already have the LDU factorization of A_k

$$A_k = L_k D_k U_k$$

then we get the LDU factorization of A_{k+1} as

$$A_{k+1} = \begin{pmatrix} L_k & 0 \\ y_k & 1 \end{pmatrix} \begin{pmatrix} D_k & 0 \\ 0 & d_{k+1} \end{pmatrix} \begin{pmatrix} U_k & z_k \\ 0 & 1 \end{pmatrix}$$

in which

$$z_k = D_k^{-1} L_k^{-1} v_k \tag{3}$$

$$y_k = w_k U_k^{-1} D_k^{-1} \tag{4}$$

$$d_{k+1} = \alpha_{k+1} - y_k D_k z_k \tag{5}$$

Thus, we can obtain the last row/column pairs of the factorization by solving two unit lower triangular systems and computing a scaled dot product. This can be exploited for sparse matrices provided we use an appropriate data structure to take advantage of the sparsity of the matrices L_k, U_k *as well as the vectors* v_k, w_k, y_k, and z_k. A good data structure to use consists of storing the rows / columns pairs w_k, v_k^T as a single row in sparse mode. All these pairs are stored in sequence.

The diagonal elements are stored separately. We refer to this as the Unsymmetric Sparse Skyline format (USS in the SPARSKIT notation [14]). Each step of the ILU factorization based on this approach will consist of two approximate sparse linear system solutions and a sparse dot product. The question that arises here is: how can we solve a sparse triangular system inexpensively? It would seem natural to solve the triangular systems (3) and (4) exactly and then drop small terms at the end, using a numerical dropping strategy. However, the cost of this strategy would be $O(n^2)$ operations at least, which is not acceptable. We note that we only need to obtain approximate solutions since we are seeking an approximate LU factorization. The first idea that comes to mind is the truncated Neumann series,

$$z_k = D_k^{-1} L_k^{-1} v_k = D_k^{-1}(I + E_k + E_k^2 + \ldots + E_k^p)v_k \tag{6}$$

in which $E_k \equiv I - L_k$. In fact, by analogy with ILU(p), it is interesting to note that the powers of E_k will also tend to become smaller as p increases. A close look at the structure of $E_k^p v_k$ shows that there is indeed a strong relation between this approach and ILU(p) in the symmetric case. We now make another important observation, namely that the vector $E_k^j v_k$ can be computed in *sparse-sparse mode*, i.e., in terms of operations involving products of *sparse matrices by sparse vectors*. Without exploiting this the total cost would still be $O(n^2)$. When multiplying a sparse matrix A by a sparse vector v the operation can best be done by accumulating the linear combinations of the columns of A. If there are only i nonzero components in the vector v and an average of ν nonzero elements per column, then the total cost will be $2 \times i \times \nu$ on the average. A sketch of the resulting ILUS algorithm is as follows.

ALGORITHM **5** . *ILUS(ϵ, p)*

Set $A_1 = D_1 = a_{11}$, $L_1 = U_1 = 1$;
For $i = 1, \ldots, n-1$ *Do:*
  Compute z_k *by (6) in sparse mode.*
  Compute y_k *in a similar way;*
  Apply numerical dropping to y_k *and* z_k
  Compute d_{k+1} *via (5)*
EndDo

We note that the computation of d_k via (5) involves the inner product of two sparse vectors which is often implemented by expanding one of the vectors into a full vector and computing the inner product of a sparse vector by this full vector.

There are mainly two advantages to the ILUS approach over the usual i, k, j implementation, used in ILUT for example. These are

- In the symmetric case, ILUS becomes an incomplete Choleski and the generation of y_k is not necessary
- It is easier to control instability of the factorization.

Regarding the control of instability, in the standard ILU factorizations, we do not have access to a partial solution with the U factor until this factor is entirely

computed. This is because the LU factorization is obtained row-wise so at step k only the first k rows of U are available, and these cannot be used to analyze instability. In contrast, the partial factors obtained at step k of ILUS are used in obtaining the solution for step $k+1$ when solving a triangular system. This allows us to estimate instability and curtail the process if a problem arises. A block-diagonal factorization can be obtained instead. The only apparent disadvantage of ILUS is that its data structure is not suitable for implementing partial pivoting.

We also point out that the one can use a simple iterative procedure such as MR or GMRES(m) to solve the triangular systems in sparse mode. Our implementation use this approach. It is our experience that these alternatives are not much better than the Neumann series approach.

3.4. ICNE

One of the first ideas that was suggested for handling symmetric positive definite sparse linear systems $Ax = b$, that are not diagonally dominant, was to use an incomplete Choleski factorization on the 'shifted' matrix $A + \alpha I$, see, e.g., [10,13]. The shift is necessary since the IC(0) factorization does not necessarily exist for positive definite matrices. The Preconditioned Conjugate Gradient (PCG) can then be used to solve the preconditioned system. This idea can be applied to the normal equations. Thus, we can use the special forms of PCG to solve the system $A^T Ax = A^T b$ (CGNR variant), preconditioned with a matrix $M = LL^T$ which is the IC(0) factorization of the shifted matrix $A^T A + \alpha I$. Similarly, for the CGNE variant we can use the CG algorithm for solving the system $AA^T y = b$, preconditioned with an IC(0) factorization for the matrix $AA^T + \alpha I$. One issue which has often been debated in the past is to find good values for the shift α. There is no easy and well-founded solution to this problem for irregularly structured symmetric sparse matrices. One idea is to select the smallest possible α which makes the shifted matrix diagonally dominant. However, this shift tends to be too large in general since, as was observed in [13], the IC(0) may exist for much smaller values of α. We can also try to determine the smallest α for which the IC(0) factorization exists. This is unfortunately not the best strategy. As it turns out, we found that in many examples, the number of steps required for convergence, starts decreasing as α increases, and then it increases again. This is borne out in Figure 1 for an example arising from the incompressible Navier Stokes equations.

What seems to be apparent from this plot, is that there is an optimal value for α which is far from the smallest admissible one. For small α, the diagonal dominance of $AA^T + \alpha I$ is weak and as a result the computed IC factorization is a poor approximation to the matrix $B(\alpha) \equiv AA^T + \alpha$. In other words, $B(\alpha)$ is close to the original matrix AA^T, but the IC(0) factorization is far from $B(\alpha)$. For large α the contrary is true. The matrix $B(\alpha)$ has a large deviation from $B(0)$ but its IC(0) factorization may be quite good. Therefore, the general shape of the curve shown in the figure is not too surprising.

A heuristic that worked reasonably well is to select α based on an approximation

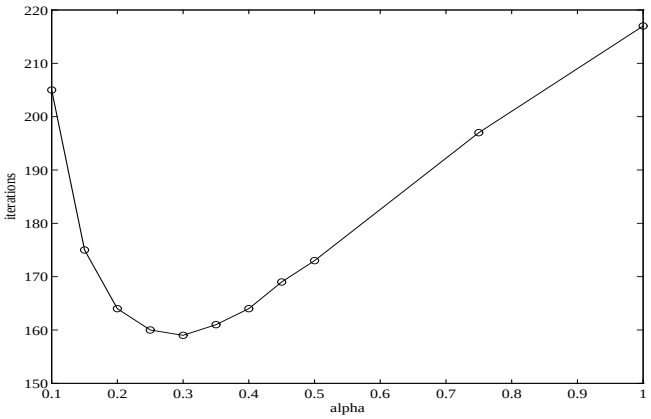

FIGURE 1: Iteration count versus shift parameter α

of the equation

$$d_{k+1} = \alpha_{k+1} - v_k^T L_k^{-T} \; D_k^{-1} L_k^{-1} v_k$$

which is derived from (5), by exploiting symmetry of the original matrix. We assume that all columns of A are scaled such that their 2-norms are equal to one. Thus, $\alpha_{k+1} = 1$ and all diagonal elements of A_k are equal to one. If all elements are shifted by α, and if the *exact factorization* were to be performed instead of IC(0) then the term $v_k^T L_k^{-T} \; D_k^{-1} L_k^{-1} v_k$ would become $v_k^T (A_k + \alpha I)^{-1} v_k$ and we would obtain the ideal relation,

$$d_{k+1} = \alpha_{k+1} + \alpha - v_k^T (A_k + \alpha I)^{-1} v_k \; .$$

Since the diagonal elements of the matrix A_k are all equal to one, a rough approximation of $v_k^T (A_k + \alpha I)^{-1} v_k$ is given by

$$\frac{\|v_k\|_2^2}{1+\alpha}.$$

Note that the larger α is the better this approximation. The idea is to choose α so that the resulting d_k's obtained from this approximation would be larger than a certain number which we take in the form 2ϵ for convenience. Let

$$\eta = \max_{k=1,\ldots,n} \|v_k\|_2^2$$

Then, we would like to have,

$$d_{k+1} = 1 + \alpha - \frac{\eta}{1+\alpha} \geq 2\epsilon$$

which, leads to the condition

$$\alpha \geq \sqrt{\eta + \epsilon^2} + \epsilon - 1 \tag{7}$$

Because of the various approximations made, this strategy is not guaranteed to work. The larger ϵ the safer the resulting α, i.e., the better the chances that the IC(0) factorization will exist for the shifted matrix. In our test cases we typically take $\epsilon = 0.2$. In case of breakdown the easiest solution is to increase ϵ, recompute a new α and attempt a new IC(0) factorization with this new shift.

3.5. APINV

There is currently a growing interest in techniques which seek to precondition a linear system by exploiting a sparse approximation to the inverse of A [1,8,3,9,12,11,2] motivated in large part by parallel processing.

One approach of this type consists of finding a matrix M such that AM is close to the identity matrix. Then the preconditioned system is of the form,

$$AMy = b, \quad x = My \ .$$

Each column m_j of M can be obtained by approximately solving the linear system $Am_j = e_j$. In [2] we exploited the observation that the iterates produced by algorithms such as GMRES or MR, stay sparse if the initial guess is sparse. Initially, the approximate inverse M is taken to be the identity matrix. Rough approximations can be obtained inexpensively by performing a very small number of steps starting with the columns of the current M. Once a first approximation of M is obtained we can improve it by incorporating it in an outer loop which takes as initial guess the columns of the most current M and improves it with a few steps of sparse-sparse mode iterative technique. A potential improvement can be obtained by using the most recent approximate inverse to precondition the system solved when approximating a column. We refer to this as *self-preconditioning*. The inner-outer algorithm looks as follows.

ALGORITHM 6 . **Self-preconditioned Minimal Residual iteration**

1. *Start:* $M = M_0$
2. *For outer* $= 1, 2, \ldots, n_o$ *do*
3. *For each column* $j = 1, \ldots, n$ *do*
4. *Define* $s := Me_j$
5. *For* $inner = 1, \ldots, n_i$ *do*
6. $r := e_j - As$, $z = Mr$
7. $q := Az$
8. $\alpha := \frac{(r,q)}{(q,q)}$
9. $s := s + \alpha z$
10. *Apply numerical dropping to* s
11. *End do*
12. *Update* j*-th column of* M*:* $m_j := s$
13. *End do*
14. *End do*

Approximate inverse preconditioners have the advantage of not requiring any forward/backward triangular solution sweeps. Instead, the preconditioning operation requires only a matrix-vector product. This means that there are no potential instabilities in the preconditioning operations. An additional benefit is that parallelization is trivial.

One of the disadvantages of approximate inverse preconditioners is that it is not easy to guarantee that the preconditioning matrix M is nonsingular. More precisely, we can guarantee that M is nonsingular only if the approximation is accurate enough, which most often leads to unrealistic conditions. Furthermore, in the non-diagonal dominant case, we do not know in advance whether or not there exists indeed an approximate inverse which is sparse enough to be practically useful. This means that the sparse approximate inverse may have to be quite dense in order to be accurate enough [2]. In spite of this, Approximate Inverse preconditioners seem to constitute a promising alternative to incomplete factorizations for very indefinite matrices.

4. NUMERICAL TESTS

We tested a few preconditioned Krylov subspace methods based on a set of preconditioners chosen from the ones described above and the accelerators DQGMRES or FGMRES. We would like to point out that all the test matrices referred to in this section are available through anonymous FTP; for more information contact the author.

We first consider a lid-driven cavity problem as modeled by the Navier-Stokes and continuity equations,

$$\begin{aligned} \mathrm{Re}(u.\nabla u) &= -\nabla p + \nabla.\nabla u \\ \nabla u &= 0 \end{aligned}$$

on a square. The boundary conditions $u = (1, 0)$ on the top edge of the square and $u = (0, 0)$ on the other sides. Rectangular elements were used, with biquadratic basis functions for velocities, based on nine edge nodes, and linear (discontinuous) basis functions for pressure based on three internal (Gauss-Legendre) nodes for each element.

We generated 11 linear systems corresponding to Reynolds number varying from $Re = 0$ to $Re = 1000$. All matrices arise from a regular mesh using 20×20 rectangular elements, leading to matrices of size $n = 4562$ and having $nnz = 138,187$ nonzero entries. The ordering of the unknowns was velocity components followed by pressure. Because of the incompressibility condition, there are many zero elements within the main diagonal. All matrices were scaled so that the rows have 2-norm one and then the resulting columns were scaled similarly. Then we generated a right-hand side by taking the product of A with the vector consisting of all ones. The initial guess to the solvers are zero vectors. Table 2 shows the number of matrix-vector products that were needed to achieve convergence. Note

Re.	ILUTP(30)	ILUT(30)	ILUTP(15)	ILUS(30)	ILUSNR(30)
0	20	20	38	27	556
100	20	20	41	36	518
200	20	20	42	40	460
300	23	23	54	42	396
400	22	22	63	47	412
500	23	23	61	49	364
600	23	23	74	59	420
700	25	25	51	63	370
800	33	33	60	79	460
900	40	40	126	95	396
1000	79	80	117	132	518

TABLE 2: Number of MatVec's required for convergence for the driven cavity test problems

that the number of nonzero elements per row required to store the LU factors for ILUT(k) and ILUTP(k) are $2k$ in addition to the original number of the original nonzero elements in each row. For ILUS(k) this number is only k. So ILUS(30) and ILUT(15) require about the same storage. In view of this, ILUS and ILUT yield very close performance for roughly the same memory usage. ILUSNR consisted of solving the normal equations $A^TAx = A^Tb$ using the preconditioned produced by ILUS on the shifted matrix $A^TA + \alpha I$. In this case, alpha was selected to be a 0.01 times the shift that would ensure diagonal dominance. This is somewhat arbitrary but the more rigorous choice of the shift dictated by diagonal dominance leads to slow convergence or non-convergence. A lower alpha was sometimes possible in these examples, but lead to non-existent factorizations in many other cases. For the ILUT, ILUTP, and ILUS preconditioners, we used GMRES acceleration with a Krylov subspace of dimension 20. The algorithms were stopped when the residual norm has been decreased by a factor of 10^7.

The ICNE strategy using IC(0) with the shift computed as suggested in Section 3.4. did not perform too well for these examples. Generally speaking most of the approaches related to the normal equations have not been competitive with the other ones for the types of problems that are discussed in this paper.

We also worked with a second set of test matrices which originated from the Package FIDAP [6]. We have collected 40 test matrices from different applications which were provided in the package. They all solve the fully coupled, incompressible Navier Stokes equation, either in 2-dimensional or three dimensional space. We show the results with only 29 of the matrices. Some of the problems which we do not show are too easy to solve. The matrices arise from modeling various phenomena ranging from turbulent flow to Stokes flow and chemical convection-diffusion equations. We list below a few characteristics of some of the problems. For more information, see [6].

- Three-dimensional flows: 2, 40. All others are two-dimensional.

				APINV(60)				
Prob	N	ILUTP	ILUT	n_o =1	n_o =2	n_o =3	n_o =4	n_o = 5
02	441	300	2	300	300	300	300	300
03	1821	300	300	300	220	60	57	53
04	1601	9	9	300	300	300	300	300
06	1651	93	300	300	300	300	300	300
07	1633	300	300	55	43	30	25	20
08	3096	300	300	300	300	300	300	300
09	3363	err	300	300	151	119	136	99
10	2410	300	300	300	300	209	129	93
12	3973	58	300	300	300	300	300	300
13	2568	err	300	71	87	57	54	51
14	3251	300	err	300	300	300	300	300
15	6867	300	300	300	138	118	300	300
18	5773	300	300	300	300	300	300	300
19	12005	300	300	300	300	300	300	300
20	2203	9	13	234	182	300	131	298
21	656	9	21	300	300	300	300	300
22	839	4	8	300	300	300	300	221
23	1409	11	16	300	300	300	300	300
24	2283	18	300	300	300	300	300	300
25	848	7	300	300	300	300	300	247
26	2163	300	78	300	300	300	300	300
27	974	4	280	300	300	300	300	300
28	2603	9	10	300	300	300	300	300
31	3909	8	9	300	300	300	300	300
32	1159	23	3	300	300	300	300	300
33	1733	300	300	214	112	144	300	55
35	19716	300	300	300	300	300	300	300
36	3079	8	300	300	300	300	300	300
40	7740	18	18	300	300	300	300	300

TABLE 3: Results for the FIDAP test problems

- Flows involving heat transfer: 9, 13, 14, 32, 33, 35.
- Turbulent flow models: 14, 18, 19, 35,
- Problems leading to symmetric linear systems: 2, 3, 4, 9, 10, 12, 13, 14, 15, 32, 33.
- Other: 12 (Stokes), 2 (Couette flow), 4 (Hamel flow), 7 (Convection).

The accelerator used for all the above problems was DQGMRES(15). Notice that for the symmetric case, we are not taking advantage of symmetry. We could have used ILUS with a CG algorithm for these problems instead. The algorithms were stopped when the residual norm has been decreased by a factor of 10^5 or the number of iterations exceeded 300. Thus, a performance showing 300 steps means that the method did not converge in a satisfactory number of steps.

There are a few problems which could not be solved with either the ILU-type approach or the APINV approach. One interesting observation from the table is that the ILUT-type preconditioners and the APINV preconditioners are in some sense complimentary, in that in many cases where the ILUT approach fails, the APINV approach succeeds, and vice-versa.

5. CONCLUDING REMARKS

We would like to make the following tentative concluding remarks. We are currently investigating a whole range of preconditioning techniques and accelerators for solving indefinite problems arising from various application areas. The solution of highly indefinite problems that arise in many of these areas, including fluid flow problems, can be very hard to solve by general purpose iterative techniques. We observed that the main source of difficulties lies in the preconditioner which, in such cases, may give rise to instabilities or inaccurate approximate factorizations. It is often possible to still solve the linear systems by using more accurate factorizations, or approximate inverse preconditioners. The only difficulty with these is that they are very demanding in terms of memory. In fact, it seems that in order to gain robustness one must require more memory. It is conceivable that more can be done from the start, e.g., by formulating a model and a discretization scheme that will lead to problems that are easier to handle by iterative methods. For example, one can think of time stepping techniques for steady state problems. Although the transient solutions may not be needed, it is clear that the intermediate problems are easier to solve because of the underlying 'continuation' approach. This approach, or any other continuation procedure, may be very slow since we may now have to solve a very large set of easier problems. The ultimate issue may well be one of trading large memory usage versus large CPU usage.

Acknowledgements. The Minnesota Supercomputer Institute provided the computer resources and an excellent research environment to conduct this research. The author would like to thank Edmond Chow for conducting the numerical experiments and Abdelkader Baggag, Andrew Chapman, and Barry Rackner for providing the test matrices.

REFERENCES

1. M. W. Benson and P. O. Frederickson. Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems. *Utilitas Math.*, 22:127–140, 1982.
2. E. Chow and Y. Saad. Approximate inverse preconditioners for general sparse matrices. Technical Report UMSI 94-101, University of Minnesota Supercomputer Institute, Minneapolis, MN 55415, May 1994.
3. J. D. F. Cosgrove, J. C. Díaz, and A. Griewank. Approximate inverse preconditioning for sparse linear systems. *Intl. J. Comp. Math.*, 44:91–110, 1992.
4. I. S. Duff and G. A. Meurant. The effect of reordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.
5. L. C. Dutto. The effect of reordering on the preconditioned GMRES algorithm for solving the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Engineering*, 36:36, 1993.
6. M. Engleman. FIDAP manuals. Technical Report Vol. 1, 2, and 3, Fluid Dynamics International, Evanston, Illinois, 1986.
7. G. H. Golub and C. Van Loan. *Matrix Computations*. Academic Press, New York, 1981.
8. M. Grote and H. D. Simon. Parallel preconditioning and approximate inverses on the connection machine. In R. F. Sincovec, D. E. Keyes, L. R. Petzold, and D. A. Reed, editors, *Parallel Processing for Scientific Computing – vol. 2*, pages 519–523. SIAM, 1992.
9. T. Huckle and M. Grote. A new approach to parallel preconditioning with sparse approximate inverses. Technical Report SCCM-94-03, Stanford University, Scientific Computing and Computational Mathematics Program, Stanford, California, 1994.
10. D. S. Kershaw. The incomplete Choleski conjugate gradient method for the iterative solution os systems of linear equations. *J. of Comput. Physics*, 26:43–65, 1978.
11. L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings ii. solution of 3d fe systems on massively parallel computers. Technical Report EM-RR 3/92, Elegant Mathematics, Inc., Bothell, Washington, 1992.
12. L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings i. theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.
13. T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–497, 1980.
14. Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.
15. Y. Saad. ILUT: a dual threshold incomplete ILU factorization. Technical Report 92-38, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1992. to appear.
16. Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Stat. Comput.*, 14:461–469, 1993.
17. Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.
18. Y. Saad and K. Wu. DQGMRES: a quasi-minimal residual algorithm based on incomplete orthogonalization. Technical Report UMSI-93/131, Minnesota Supercomputing Institute, Minneapolis,MN, 1993. submitted.
19. H. A. van der Vorst and C. Vuik. GMRESR: a family of nested gmres methods. *Numerical Linear Algebra with Applications*, 1:–, 194. to appear.
20. J. W. Watts-III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineer Journal*, 21:345–353, 1981.