Block LU Preconditioners for Symmetric and Nonsymmetric Saddle Point Problems*

Leigh Little † Yousef Saad †

June 16, 1999

Abstract

In this paper, a block LU preconditioner for saddle point problems is presented. The main difference between the approach presented here and that of other studies is that an explicit, accurate approximation of the Schur complement matrix is efficiently computed. This is used to compute a preconditioner to the Schur complement matrix that is in turn defines a preconditioner for a global iteration. The results indicate that this preconditioner is effective on problems arising from CFD applications.

Key words: saddle point problem, preconditioning methods, incomplete LU, block LU
AMS subject classifications: 65F10, 65F50, 65N22

1 Introduction

In many applied problems, it is necessary to solve a saddle point problem of the form

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{1}$$

where A and B are matrices of size $n_A \times n_A$ and $n_A \times m_B$ respectively and $m_B \leq n_A$. If in addition, A is symmetric positive definite (SPD) then the problem is a classical problem. If on the other hand A is indefinite or nonsymmetric, the problem is termed generalized.

Such systems of equations arise in the solution of the Stokes and Navier-Stokes equations, linear elasticity theory, and electromagnetics as well as in constrained optimization, when minimizing a quadratic functional subject to the linear constraints $B^T u = g$. The discretized equations are frequently large and sparse, making direct solutions infeasible. The difficulty in obtaining accurate numerical solutions to Equation (1) is that the system is indefinite, as seen from the LDL^T factorization of the global matrix,

$$M_g = \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} \tag{2}$$

*Work supported by NSF under grant NSF/CTS 9873236 and by the Minnesota Supercomputer Institute

†Department of Computer Science and Engineering, University of Minnesota, 55455. e-mail: {little,saad}@cs.umn.edu

where $-S = -B^T A^{-1} B$ is the Schur complement matrix. For the classical problem, M_g has n_A positive and m_B negative eigenvalues while for the generalized problem, little can be said regarding the spectrum of M_g.

When used without preconditioning, Krylov subspace methods can perform poorly on such systems, see , e.g., [19], because they are indefinite. Three main solution approaches to Equation (1) can be distinguished. The first is a condensation approach. The variable u is eliminated from the second equation, which results in the Schur complement system

$$Sp = B^T u - g. \tag{3}$$

This can be solved exactly for p and u can then be obtained from the solution of

$$Au = f - Bp. \tag{4}$$

This approach which has been examined in [6, 22], for example, tends to be very expensive. Each multiplication by S requires an exact (or very precise) solve with the matrix A and matrix-vector products with B and B^T. The solves with A can be efficiently performed using multigrid solvers in the case where A is equivalent to a Laplacian operator, but more general forms for A may not be as easily tractable. A simplified alternative to direct condensation is the Uzawa algorithm [1]. In this case, the equation blocks are iteratively solved in an outer loop using another iterative procedure for the inner loop. If the inner iterations are nearly exact, then the algorithm is called the exact Uzawa algorithm. It has been shown ([2]) that accurate inner loop solves are frequently not necessary. This leads to the so-called inexact algorithm. This was studied in [3] for the symmetric case and in [4] for the nonsymmetric case.

A third approach, and the one taken here, is to directly attack the global system as a whole. It is still necessary to utilize the block nature of the matrix to design an effective preconditioner, but the task is now much simpler since exact solutions with the matrix A are no longer required. This approach has been examined by many researchers. The preconditioners investigated are generally block diagonal or block triangular matrices of the form

$$P_D = \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix}; \qquad P_T = \begin{pmatrix} \hat{A} & B \\ 0 & -\hat{S} \end{pmatrix}$$

respectively. Rusten and Winther [17] consider a block diagonal preconditioner for symmetric positive definite A. Preconditioners used for A include fast Poisson solvers and the incomplete Cholesky factorization. Preconditioning of the Schur complement is obtained by taking $\hat{S} = I$ in the simplest case, or using the same preconditioners as for A applied to a simplified Schur complement approximation of the form $\tilde{S} = B^T B$.

Block diagonal preconditioners are investigated in [23] and [21] for the stabilized Stokes equations, first in the simple case where both preconditioning matrices are diagonal, then with more general preconditioners such as multigrid V-cycles in the latter reference.

The results in [10] extend those of [22] mentioned above to the general nonsymmetric case. Here, $\hat{A}$ is taken to be either A (i.e., an exact preconditioning) or the result of an

iterative solution process using A preconditioned by horizontal- or vertical-line oriented Gauss-Seidel splittings. The Schur complement preconditioner is taken to be a scalar multiple of the pressure mass matrix. This matrix, as well as a preconditioning based on a one-layer overlap Schwartz multiplicative preconditioner are spectrally equivalent to the matrix $h^d I$ where d is the problem dimension and h is some measure of a uniform mesh discretization parameter. This allows simple bounds, independent of the mesh parameter h, to be obtained for the eigenvalues of the right preconditioned matrix when exact solves with A are performed. For inexact solves with A, somewhat generous bounds on these eigenvalues are obtained. In [8], more precise bounds on the eigenvalues for the inexact solves are found by considering the inexact method to be a perturbation of the exact solution. The results in [9] consider more general Schur complement approximations, leading to the $B^T A^{-1} B$ (in this notation) preconditioner. It is shown that this preconditioner is exact in the sense that if all solves with A are exact, then a global iteration will converge in a single step. The exact solves are replaced by approximate multigrid solves with the resulting performance being roughly midway between an exact Schur complement preconditioner and a mass matrix preconditioning.

Block diagonal and triangular preconditioners for stabilized saddle point problems in linear elasticity theory, for which A is SPD are considered in [13, 14]. The preconditioner is optimal in the sense that the iteration counts are independent of the Lamé constant. Here again, mass matrix preconditioning is used for the Schur complement to obtain bounds on the eigenvalues that are independent of the mesh parameter. The solvers are the preconditioned conjugate residual method, Bicgstab and GMRES(10) with either exact solves with A and the stabilizing matrix C or two-level multigrid V-cycle preconditioning.

Block preconditioners of the form

$$P = \begin{pmatrix} D & B^T \\ B & 0 \end{pmatrix}$$

have been investigated by Dyn and Ferguson [6] who consider the case when D is obtained from a JOR, SOR or SSOR splitting of A. Golub and Wathen [11] take D to be a scaled identity, $\mathrm{diag}(A)^{-1}$ and the symmetric part of A. This is shown to have the advantage that all eigenvalues of the preconditioned operator corresponding to the Schur complement block are zero. Simple convergence criteria based on D and A are given.

The approach taken here is similar to the methods above, however, an accurate approximation to the Schur complement approximation is computed. This is then further approximated by standard preconditioners.

The remainder of the paper is as follows. Section 2 presents the proposed block LU preconditioner as well as some details regarding its efficient implementation. Section 3 presents a series of numerical experiments for matrices arising from various disciplines. Finally, conclusions are given in Section 4.

2 Block LU Preconditioners

Consider the general situation where the global matrix is nonsymmetric:

$$M_g = \begin{pmatrix} A & B \\ C^T & 0 \end{pmatrix}. \tag{5}$$

The block LU factorization of the above matrix can be written as

$$M_g = \begin{pmatrix} A & 0 \\ C^T & -S \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix} \tag{6}$$

where the matrix $-S = -C^T A^{-1} B$, is the Schur complement associated with the (2,2) block of M_g. The decomposition suggests a block LU preconditioner of the form

$$\tilde{M}_g = \begin{pmatrix} \tilde{A} & 0 \\ C^T & -\hat{S} \end{pmatrix} \begin{pmatrix} I & \tilde{A}^{-1}B \\ 0 & I \end{pmatrix} \tag{7}$$

where $\tilde{A}$ and $\tilde{S}$ represent some approximation to A and S respectively. The action of the block preconditioner (7) is easily applied provided solutions to systems of the form $\tilde{A}x = b$ and $\tilde{S}y = c$ can be inexpensively obtained. The system

$$\tilde{M}_g \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} w \\ z \end{pmatrix}$$

can be solved by the algorithm below.

ALGORITHM **2.1 Block LU solution procedure**

1. Solve $\tilde{A}x = w$
2. Solve $\tilde{S}y = C^T x - z$
3. Solve $\tilde{A}t = By$
4. Compute $x = x - t$.

The procedure involves two solves with $\tilde{A}$ and one with $\tilde{S}$.

A block LU preconditioner based on the approximate factorization (7) of M_g requires two ingredients: (1) an approximation $\tilde{A}$ to A, for which solves are inexpensive; and (2) an approximation $\tilde{S}$ to S, with which it is again inexpensive to solve linear systems. A variety of options are available for choosing $\tilde{A}$ because A is usually available, either directly or as a good approximation. For example, it is easy to approximate A by an incomplete Cholesky (or ILU) factorization. The more crucial part is selecting an approximation to S. Once this approximation is found, it should then be approximately factored. An approximate factorization of $\tilde{S}$ may also be obtained directly, i.e., without forming $\tilde{S}$ first, as a by-product of some other factorization.

2.1 Approximating the Schur complement

In this section, the problem of obtaining approximations $\tilde{S}$ to the Schur complement is examined. Forming A^{-1} to compute an approximation to $\tilde{S}$ is impractical because it is usually a dense matrix. Two common approximations that are readily computable are

$$\begin{aligned} \tilde{S}_1 &= C^T B, \\ \tilde{S}_2 &= C^T \mathrm{diag}(A)^{-1} B. \end{aligned}$$

However, these approximations are not always accurate enough. One step towards a more accurate approximation, developed for the case where $C = B$, is the $B^T A^{-1} B$ preconditioner of [9] mentioned above. The preconditioner obtained here is similar, but somewhat more direct. While the multigrid solvers used in [9] can be considered optimal, the mesh information may not always be available.

For the symmetric saddle point problem where $C = B$ and A is SPD, a natural approximation for $\tilde{A}$ is in the form of an incomplete Cholesky (ICT) factorization,

$$\tilde{A} = LL^T.$$

This factorization yields a simple form for the Schur complement approximation by computing

$$\begin{aligned} \tilde{S}_3 &= B^T (LL^T)^{-1} B \\ &= B^T L^{-T} L^{-1} B \\ &= X^T X \end{aligned}$$

where $X = L^{-1}B$. Note that the simpler approximations $\tilde{S}_1$ and $\tilde{S}_2$ can be cast into this form by putting $L = I$ and $L = \mathrm{diag}(A)^{\frac{1}{2}}$ respectively.

For later reference it is important to unravel a relationship between the X and Y matrices, and the standard LU (or Cholesky) factorization of the global matrix M_g. Consider the LU factorization of M_g written here in its general (nonsymmetric) form:

$$\begin{pmatrix} A & B \\ C^T & 0 \end{pmatrix} = \begin{pmatrix} L & 0 \\ C^T U^{-1} & L_S \end{pmatrix} \begin{pmatrix} U & L^{-1}B \\ 0 & U_S \end{pmatrix} \equiv \begin{pmatrix} L & 0 \\ Y^T & L_S \end{pmatrix} \begin{pmatrix} U & X \\ 0 & U_S \end{pmatrix}. \qquad (8)$$

Here L, U is the LU factorization of A, L_S is unit lower triangular and U_S is upper triangular. When A is SPD and $C = B$, then $U \equiv L^T$ and $Y \equiv X$. The above relation shows that the matrix X can be viewed as the (1,2) part of the block form of the U factor. It is also important to note that *the L_S, U_S pair is the LU factorization of the Schur complement matrix $-S$ of* (6).

The application of the preconditioner requires solving systems of the form $\tilde{A}x = f$ and $\tilde{S}y = g$ as indicated in Algorithm 2.1. These equations represent approximations used to define a more global preconditioner, so accurate solves are not needed. The equations $Ax = f$ and $\tilde{S}y = g$ may be solved iteratively using $\hat{A}$ as a preconditioner for A and $\hat{S}$

as a preconditioner for S. Flexible accelerators (such as FGMRES [19]), which allow for a variable preconditioner, should be used to solve Equation (1) in this case. However, solving these systems iteratively may represent an expensive alternative. In the extreme situation where the incomplete decompositions are performed exactly, then iteration is not necessary. As is generally observed with similar algorithms, it is sufficient to apply the preconditioning operation directly to the equations and avoid iterating. This approach also presents a storage savings because $\tilde{S}$ can be discarded once the preconditioner $\hat{S}$ has been computed.

2.2 Computing the X factor

This section addresses the issue of computing an approximation to the matrix $X = L^{-1}B$. It may be thought at first that the matrix X is dense and therefore, any attempt to compute an approximation to it may be too expensive. In fact, X is often a fairly sparse matrix and methods for generating an approximation to it can be inexpensive and effective.

If L and B are stored in a sparse row format, it is natural to derive approximation methods by including dropping strategies from the following row-oriented algorithm. The i-th row of X is denoted by X_i and similarly for B and L.

ALGORITHM **2.2 Generic row-oriented computation of X**

1. *For* $i = 1, n_A$ *Do:*
2. $\quad X_i := B_i$
3. $\quad$ *For* $k = 1, 2, \ldots, i-1$ *and if* $L_{ik} \neq 0$ *Do:*
4. $\quad\quad X_i := X_i - L_{ik} * X_k$
5. $\quad$ *EndDo*
6. $\quad X_i := X_i / L_{ii}$
7. *EndDo*

The i-th row of X is obtained by subtracting a linear combination of previous rows and the resulting row is scaled by L_{ii}. Note that for the standard LU factorization $L_{ii} = 1$ so the scaling by L_{ii} is only necessary in this case. This row-oriented process is illustrated in Figure 1.

Two facts are useful to note at this point. First, according to an earlier remark, see Equation (8), the above process will compute the (1,2) part of the U factor in the Gaussian elimination applied to the global matrix M_g. Second, the first few rows of X are sparse and then become denser and denser, as is typical in sparse Gaussian elimination, so dropping strategies should be added to reduce fill-ins.

A common dropping strategy is one that is based on a threshold technique: any fill-in introduced at the completion of the loop in lines 3–5 of Algorithm 2.2 is dropped (replaced by zero) if it is smaller than a certain tolerance, weighted by the initial norm of row B_i. Another popular dropping strategy is one that is based on level-of-fill. A level-of-fill is assigned to each element during the factorization process of Gaussian elimination, see e.g.,

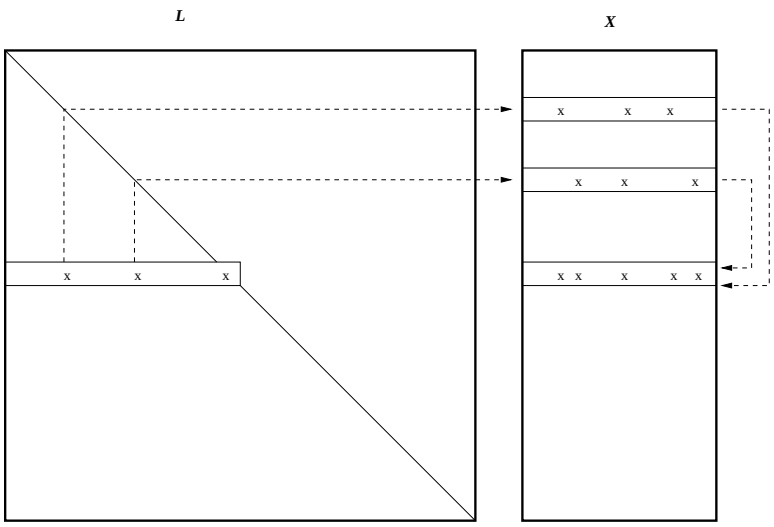

Figure 1: Row-oriented computation of $L^{-1}B$.

[19]. For the sake of completeness we provide here the definition of the level-of-fill of an element. Initially, the level-of-fill $lev(m_{ij})$ of an arbitrary element m_{ij} is infinity if m_{ij} is a zero element and zero otherwise. Every time an element is modified in the Gaussian elimination process by the formula (for a general matrix M),

$$m_{ij} = m_{ij} - m_{ik} \times m_{kj}$$

its level-of-fill is updated by

$$lev(m_{ij}) = \min\{lev(m_{ij}), lev(m_{ik}) + lev(m_{kj}) + 1\}. \tag{9}$$

Here, m_{ij} represents an element of the L matrix if $i > j$ and the U matrix for $i \leq j$. The ILU(p) factorization of the matrix consists of performing the usual Gaussian elimination process but every fill-in element whose level-of-fill exceeds p is dropped.

In the following it is assumed that ILU(p) has been used to obtain L, in incomplete Cholesky factor for A. As a result each element of L has been assigned a level-of-fill at the completion of the ILU algorithm. To define a level-of-fill variant of Algorithm 2.2 all that is required is to add code for updating the fill-level values in the process and dropping those whose level exceeds p. The result is as follows.

ALGORITHM **2.3 Row-oriented Level-p procedure for computing X**

1. *For* $i = 2, n_A$ *Do:*
2. $X_i = B_i$; $lev(X_{ij}) = \{0$ *if* $B_{ij} \neq 0; \infty$ *if* $B_{ij} = 0\}$ *for* $j = 1, 2, \ldots, n_B$.
3. *For* $k = 1, 2, \ldots, i$ *and if* $L_{ik} \neq 0$ *Do:*
4. $X_i := X_i - L_{ik} * X_k$
5. $lev(X_{ij}) = \min\{lev(X_{ij}), lev(L_{ik}) + lev(X_{kj}) + 1\}$, $j = 1, \ldots, i$
6. *EndDo*

7 *Drop each X_{ij} such that $lev(X_{ij}) > p$ for $j = 1, \ldots, m_B$*
8 $X_i = X_i / L_{ii}$
9 *EndDo*

The factorization (8) suggests that the X matrix obtained from the above algorithm is the same as the one that would be obtained from the (1,2) block of the U factor of an incomplete LU factorization, using the same level-of-fill. This is clearly true as a simple comparison with the IKJ version of the Gaussian elimination would show (see Algorithm 10.2, in [19]). If an IKJ version of Gaussian elimination, modified with the level-based dropping strategy, is applied to M_g then the X factor can extracted from the U-part of the ILU factorization. As was observed before, the L_S, U_S parts in (8) constitute the LU factors of the Schur complement matrix $-S$. It would be interesting to modify the Gaussian elimination process in order to obtain the matrix $-S$ directly instead of its factored form.

Such a modification, called the restricted version of Gaussian elimination was introduced and exploited in [20]. We reproduce it here for the sake of completeness.

ALGORITHM **2.4 Restricted Gaussian Elimination algorithm**

1. *For* $i = 1, n_G$
2. *For* $k = 1, \min(i-1, n_A)$
3. $m_{i,k} = m_{i,k} / m_{k,k}$
4. *For* $j = k+1, n_G$ *Do*
5. $m_{i,j} := m_{i,j} - m_{i,k} * m_{k,j}$
6. *EndFor(j)*
7. *EndFor(k)*
8. *EndFor(i)*

In this algorithm, the standard elimination process is carried out for the first n_A rows of M_g. For the remaining rows, the elimination is carried out only to column n_A. It is easy to see from what was stated above (see also [20]) that the (2,2) block in the matrix M resulting from this factorization is actually the Schur complement matrix associated with the C block. Thus, sparse Gaussian elimination techniques compute LU factorizations of these Schur complement matrices.

Assume now that a dropping strategy is used to drop fill-in elements from the blocks in the above restricted Gaussian elimination procedure. In the following description it is assumed that a level-of-fill strategy is used. The level-of-fill strategy can be trivially adapted to Algorithm 2.4. The block ILU factorization resulting from the corresponding modification of Algorithm 2.4 is

$$\begin{pmatrix} A & B \\ C^T & 0 \end{pmatrix} = \begin{pmatrix} L & 0 \\ Y^T & I \end{pmatrix} \begin{pmatrix} U & X \\ 0 & -\tilde{S} \end{pmatrix} + R \tag{10}$$

where R represents the matrix of elements that have been dropped. The matrix X is an approximation to $L^{-1}B$. Thus, if no elements are dropped $X = L^{-1}B$, with $LL^T = A$, and $\tilde{S} = S = Y^T X$.

A trivial observation is that a zero-fill strategy will fail past row n_A due to the large zero (2,2) block. Every element in this block will have a level-of-fill of 1 or higher during the elimination. This suggests using a different level-of-fill for the two different parts of the global matrix: *a level-of-fill of p for the (1,1), (1,2), and (2,1) parts and a level-of-fill of $2p+1$ for the (2,2) part.* The simplest of these strategies which corresponds to level-1 Schur complement preconditioner is obtained for $p = 0$. It results in a matrix X which has the same sparsity pattern as B. More generally, we can state the following proposition.

Proposition 2.1 *Assume that a dropping strategy is applied in Algorithm 2.4 with a level-of-fill tolerance of p for the (1,1), (1,2), (2,1) blocks and $2p+1$ for the (2,2) block. Then the resulting matrix $-\tilde{S}$ in (10) is equal to $Y^T X$, where X is obtained by Algorithm 2.2 using a level-of-fill of p.*

Proof. First we refer to the comment following Algorithm 2.3 which showed that the (1,2) factor X in (10) is identical with that obtained by Algorithm 2.3 using the same level-of-fill p. Next, according to the definition of the levels in (9), each element m_{ij} in the (2,2) block, will have a level-of-fill not exceeding $2p+1$. So if a tolerance level of $2p+1$ is used for this block, all (fill-in) elements introduced by Gaussian elimination in this block will be kept, i.e., no element is dropped. In this case, the (2,2) block of the matrix R in (10) is zero and therefore comparing the (2,2) blocks on both sides of (10), we obtain that $Y^T X - \tilde{S} = 0$ which is the desired result. ■

The proof indicates that no dropping is employed in the (2,2) block, i.e., all fill-ins in this block are kept. As a result a variant of the assumptions in the proposition is to state that *a dropping strategy is applied in Algorithm 2.4 with a level-of-fill tolerance of p for the (1,1), (1,2), (2,1) blocks and no dropping in the (2,2) block.* Under these alternative assumptions the conclusion is the same.

An example of this process is given in Figure 2 two different fill-levels. X_0 indicates that X is computed using Algorithm 2.2 and restricting X to have the same sparsity structure as B (a zero-fill procedure) while X_5 means that X is computed using Algorithm 2.2 and restricting each row to have a maximum of 5 elements. This is slightly different than the level-of-fill approach given in Algorithm 2.3. In this example, A and B have very narrow bandwidths as does X_0. As a result, $X_0^T X_0$ has a small bandwidth and is quite sparse. For $p = 5$, a moderate amount of fill-in occurs in X_5 while $X_5^T X_5$ is much more dense. In contrast, the exact Schur complement matrix for this case is a full matrix with 665856 nonzeros.

2.3 Preconditioning $\tilde{S}$

The procedure described above can produce as accurate a Schur complement approximation as desired by increasing p when constructing X. However, to apply the preconditioner,

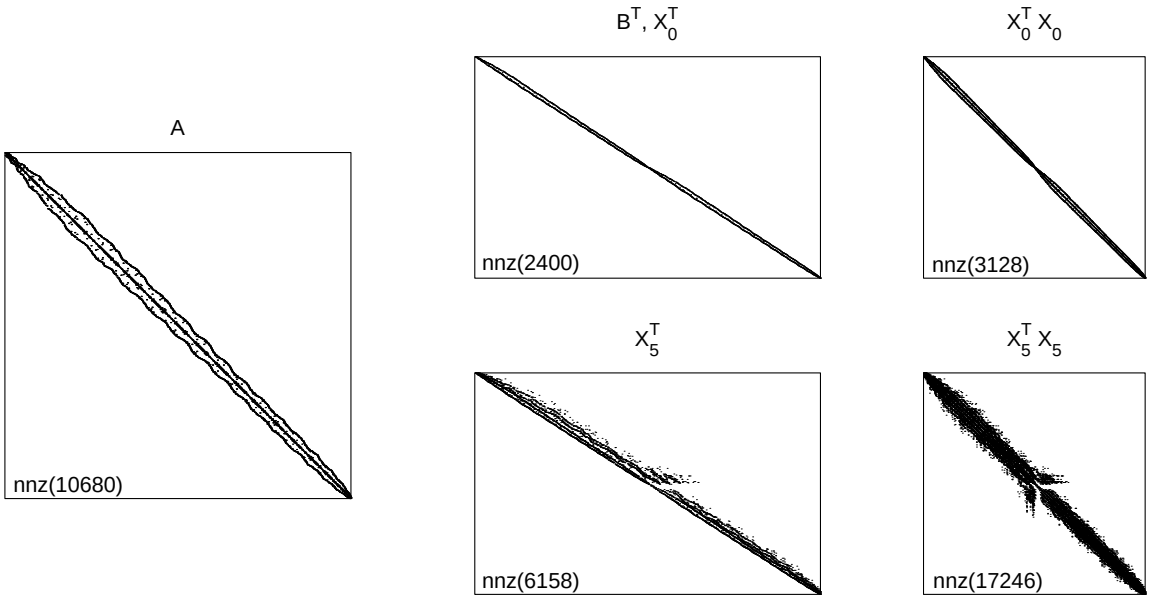

Figure 2: An example of the resulting sparsity pattern of X and $X^T X$ when X is computed using Algorithm 2.2 and restricting X to have the same sparsity structure as B (X_0) and when X is allowed to have a maximum of 5 elements per row (X_5). A is of dimension 1272 and B is of dimension 1272×816.

it is still necessary to solve systems with $\tilde{S}$, or some reasonable approximation.

When $C = B$ and A is SPD, all forms of $\tilde{S}$ given above are SPD (or symmetric positive semi-definite if B is rank deficient). It is possible to use another ICT preconditioner [12] for $\tilde{S}$, or more generally, an incomplete LDL^T factorization should $\tilde{S}$ be only symmetric positive semi-definite. Using an ILUT preconditioner is another alternative though the ICT forms would be preferable since they exploit symmetry.

The form of $\tilde{S}$ indicates that the Schur complement equation is closely related to the normal equations. This allows a preconditioner based on incomplete QR (IQR) factorizations [18]. The idea here is to decompose X so that $X \approx QR$ where R is upper triangular and Q has nearly orthonormal columns. For sufficiently large fill-in, the approximation $Q^T Q \approx I$ then holds. This gives

$$X^T X \approx R^T Q^T Q R \approx R^T R.$$

This preconditioner also results in a memory storage saving because once R has been computed, the matrix Q is no longer needed and can be discarded.

The overall preconditioner for the case when M_g is symmetric and A is SPD can now be described.

1) Compute $\hat{A} = LL^T$ via incomplete Cholesky factorization.

2) Compute the solution to $LX = B$ using Algorithm 2.3.

3) Compute $\hat{S}$ in the form of an ICT or ILUT of $X^T X$ or an IQR of X.

4) Apply Algorithm 2.1, replacing $\tilde{A}$ and $\tilde{S}$ with $\hat{A}$ and $\hat{S}$ respectively.

The modifications for the generalized saddle point problem is straightforward. As an ICT factorization for A is no longer possible, decompose A as $\tilde{A} = LU$ via an incomplete LU factorization, then define

$$\begin{aligned} \tilde{S}_3 &= C^T (LU)^{-1} B \\ &= C^T U^{-1} L^{-1} B \\ &= Y^T X \end{aligned}$$

where X is as before and $Y = U^{-T} B$. The preconditioning options for $\tilde{S}_3$ are now somewhat restricted. In addition to an ILUT preconditioner, sparse approximate inverse preconditioning can also be used though only ILUT preconditioning is examined here.

3 Numerical Experiments

In this section, a global iteration for Equation (1) using the block LU preconditioner will be compared with other solution techniques. A total of five test cases will be examined.

3.1 Description of Test Cases

The test cases that will be investigated here are described briefly.

Case 1 This is a symmetric problem that arises from a magnetostatic problem and is provided by [16].

Case 2 This is a symmetric problem that arises from the Navier-Stokes equations in the simulation of the free sedimentation of 60 particles in a Newtonian fluid and is provided by [5].

Case 3 This symmetric problem is from the Stokes equations in the simulation of the L-shaped driven cavity at Reynolds number 1000.

Case 4 This nonsymmetric problem is from the solution of the Oseen equations in an L-shaped cavity at Reynolds number 1000 with a circular vortex convective wind.

Case 5 This is the same as Case 4, but the strength of the convective wind has been increased by a factor of 10. Cases 3–5 are from [15].

The matrix properties are summarized in Table 1.

Condition numbers are obtained from the MATLAB condition number estimator. In Cases 2–5, the matrix B is nearly rank deficient. The main difference between Case 2 and Cases 3–5 is that Case 2 uses a P1-iso-P2 finite element formulation while Cases 3–5 use a P1-bubble-P1 formulation. It should be noted that the implementation of the bubble

Case	n_A	nna(A)	K(A)	m_B	nnz(B)	Symmetry
1	8980	78740	2.827(4)	5900	17570	Y
2	18844	227985	2.672(3)	3284	94662	Y
3	12418	86402	9.091(0)	2113	41785	Y
4	12418	86402	1.833(2)	2113	41785	N
5	12418	86402	1.467(3)	2113	41785	N

Table 1: Matrix properties of test cases.

formulation used here is atypical. As the bubble nodes lead to large diagonal blocks in the the matrix, these nodes are usually statically condensed at the element level [15]. However, in the case of Navier-Stokes equations, this leads to a nonstandard form for the saddle point problem because the off-diagonal blocks are no longer transposes of each other. Also, a stabilization matrix arises in the Schur complement block. For this reason, the bubble nodes have been retained in the system matrices.

3.2 Solution Approaches and Experiment Parameters

Three approaches are used to solve Equation (1).

Method M1: Solve Equation (1) iteratively using the block preconditioner (7). For this case, the solver is chosen to be right preconditioned GMRES(20). For preconditioning of the A block, both an ICT, with a maximum fill-in level of 10, and simple diagonal preconditioning are used. The drop tolerance has been set to zero. The Schur complement is approximated using the two simple approximations $\tilde{S}_1$ and $\tilde{S}_2$ as well as the more accurate $\tilde{S}_3$. For preconditioning the Schur complement block, the ICT, ILUT and IQR preconditioners with a fill-in level of 20 and no drop tolerance are used.

Method M2: Condensation method of [6, 22]. Here, the right preconditioned conjugate gradient (CG) method is used to solve both the Schur complement block and systems involving A in the case when A is SPD. When A is nonsymmetric, systems associated with these blocks are solved by GMRES(20). The same preconditioners for A and $\tilde{S}$ as above are used here. Note that when solving the Schur complement equation (3), the exact S must be used for the coefficient matrix.

Method M3: Ignore the block structure of the global system and use right preconditioned GMRES(20) as the solver, combined with an ILUT(20) preconditioner applied to the global coefficient matrix. The drop tolerance was $1.0E(-4)$.

For all tests, the convergence criterion is a reduction of the initial residual by a factor of $1.0E(-8)$ except for method M2, where in addition the solves with the matrix A have a reduction of $1.0E(-12)$. A maximum of 250 iterations is allowed. The matrix X is computed using the zero-fill approach. The initial guess is taken to be a vector of all zeros

and if no right hand side is provided, the right hand side is taken to be a vector of the form

$$\begin{pmatrix} f \\ 0 \end{pmatrix}; \qquad f_i = \frac{i}{n_A + m_B}.$$

It should be noted that these parameters are not optimal for all of the test cases. The main goal is in comparing how the $\tilde{S}_3$ preconditioner performs as compared to the simpler forms $\tilde{S}_1$ and $\tilde{S}_2$, - so a fixed set of operating parameters that allowed for most of the combinations to converge was selected.

With the exception of the condition number estimates, all programs were written in Fortran-90 and executed on an SGI Challenge L with 512 MB of memory and a R-10000, 196 MHz processor.

3.3 Comparison of Solution Techniques

For each of the Cases 1–3, the three Schur complement approximations are tested for the solution methods M1 and M2 using both diagonal and ICT preconditioning for the matrix A. The Schur complement equation is preconditioned using ICT. Method M3 is also tested. This gives a total of 13 tests for each case.

Condition number estimates of the Schur complement matrices for all three approximations and Cases 1–5 are given in Table 2.

Case	K($\tilde{S}_1$)	K($\tilde{S}_2$)	K($\tilde{S}_3$)
1	2.22E(5)	2.22E(5)	2.47E(3)
2	1.25E(5)	1.35E(4)	6.15E(3)
3	5.65E(2)	1.09E(3)	1.10E(3)
4	5.65E(2)	6.57E(2)	1.28E(8)
5	5.65E(2)	1.08E(3)	3.48E(7)

Table 2: Condition numbers of the three different Schur complement approximations for Cases 1–5.

The performance of the various methods is quite surprising for Case 1. The only methods that converge within the parameter ranges considered are methods M1 and M2 with diagonal preconditioning for A and the $\tilde{S}_1$ Schur complement approximation. This particular matrix has been scaled so that the diagonal of A is an identity, hence there is actually no preconditioning for A. The convergence curves are shown in Figure 3. Of the methods that did converge, M1 is superior. The preconditioning times for methods M1 and M2 are similar, but method M1 converged in about one-sixth the time as method M2. This is most likely due to the necessity of performing highly accurate solves with A in the solution of (3), indicated by the fairly large condition number of A (see Table 1).

The interesting observation is that the more accurate ICT factorization for A and accurate Schur complement approximation $\tilde{S}_3$ perform very poorly for this case despite the fact that the condition number of $\tilde{S}_3$ is 2 orders of magnitude smaller. The only way

to make either method M1 or M2 converge with these approximations is to increase the allowable fill-in parameter in the A and $\tilde{S}_3$ preconditioners to 60 and 120 respectively. This results in an excessive preconditioning time as well as an expensive solution process, since the preconditioners are much more expensive to apply.

Rusten and Winther [17] indicate that for appropriately scaled Stokes problems, the combination of identity preconditioning and ICT preconditioning of $\tilde{S}_1$ performs quite well. However, this does not explain the poor performance of ICT preconditioning for A. This stems from the fact that in this case A is SPD but not diagonally dominant. This carries over to the L factor in the ICT of A. As a result, the zero-fill in procedure for computing X is not a good approach, since it assumes that elements away from the diagonal of L are not important.

Figure 4 shows the history for the best case in Figure 3 using ICT, ILUT and IQR preconditioning for the Schur complement. Of these, the ICT appears to be most effective. The convergence rate of the ILUT preconditioning is nearly identical to ICT, however, the preconditioning time is much greater. The IQR preconditioning time is quite good; less than ICT in fact however, the convergence rate is only about one half that of ICT. As it turns out, this is an artifact of the approximation $Q^T Q \approx I$. If the allowable fill-in in computing Q is tripled, then a similar convergence rate is obtained. However, this increases the preconditioning time to about 1.5 times that needed for the ILUT.

For Case 2, the $\tilde{S}_3$ approximation performs quite well and in fact, is the only approximation that converges within the defined parameters. Convergence curves are shown in Figure 5. Method M1 performs the fastest, followed by methods M3 and M2. The preconditioning times of methods M1 and M2 are the same, but again, the accurate solves with A greatly degrade the convergence rate of method M2.

The simple Schur complement approximations $\tilde{S}_1$ and $\tilde{S}_2$ perform poorly because the matrix B is nearly rank deficient, hence the conditioning of the Schur complement preconditioner is quite poor. An attempt was made to account for this deficiency by removing either the first or last column from B, but this seemed to have no effect on the convergence behavior. Ideally, the column that is the most linearly dependent should be removed, but there is no computationally efficient way to determine this.

An attempt was made to scale the equations in the manner suggested in [17] to see if results similar to Case 1 could be obtained. This amounts to symmetrically scaling A by $\text{diag}(A)^{-\frac{1}{2}}$ and B by $Re^{\frac{1}{2}}$ where Re is the Reynolds number. Unfortunately, the ICT factorization for A under this scaling fails due to a zero pivot. The common technique of specifying a minimum value of the pivot in this case fixed this problem, but the overall iterative method did not converge for any variations of methods M1 and M2.

Figure 6 shows the convergence histories for the best case in Figure 5 using ICT, ILUT and IQR preconditioning. The results are qualitatively identical to those discussed above for Figure 4.

The results for Case 3 are quite different from the previous cases. Nearly all approaches are successful. The only technique that fails within the defined parameters is

method M2 with diagonal preconditioning for A and the $\tilde{S}_2$ Schur complement approximation. Some representative convergence curves are shown in Figure 7 and convergence information for all cases is given in Table 3. As in Case 2, the best performance is obtained from method M1 using the $\tilde{S}_3$ Schur complement approximation and ICT preconditioning for A though this is only slightly better than diagonal preconditioning for A. As one might suspect from the tabular data, A is very diagonally dominant. In addition, the condition number of A is quite small hence, method M2 is relatively more effective than in the previous 2 cases. The worst case is method M3. Though the convergence rate is acceptable, the preconditioning time is roughly factor of 25–30 larger than all the other methods.

Figure 8 shows convergence curves for method M1 with ICT preconditioning for A and the $\tilde{S}_3$ approximation using ICT, ILUT and IQR preconditioning. The results are similar to Figures 4 and 6, though the IQR seems to be more effective than in Cases 1–2.

Method	A	S	It	P	S	Method	A	S	It	P	S
M1	C	1	68	2.46	13.4	M2	C	1	32	2.46	14.2
M1	C	2	158	2.39	30.9	M2	C	2	98	2.39	41.9
M1	C	3	20	2.56	4.17	M2	C	3	25	2.56	11.4
M1	D	1	196	1.68	26.6	M2	D	1	32	1.68	27.2
M1	D	2	DNC			M2	D	2	98	1.58	79.5
M1	D	3	24	2.41	4.48	M2	D	3	25	2.41	21.8
M3			52	95.3	9.18						

Table 3: Convergence results for Case 3. The first six column headings indicate, in order, the preconditioning type for A (C for Cholesky and D for diagonal), the Schur complement approximation type, the iteration count, the preconditioning time and the solution time.

For the weakly nonsymmetric Case 4, only $\tilde{S}_3$ has been computed, though it would be possible to use $\tilde{S}_1$ and $\tilde{S}_2$ as well. ILUT preconditioning is used for both A and the Schur complement. The results are shown in Figure 9. Method M1 is far superior to methods M2 and M3. The overall solution time for method M1 is not much greater than for Case 3 despite the fact the the condition number of $\tilde{S}_3$ (see Table 2) is five orders of magnitude higher. The solution time for method M3 is the nearly same as for method M1, but the preconditioning time is so great that M3 is unusable in practice. Method M2 takes 8 times longer to converge.

The observations for the highly nonsymmetric Case 5 are similar, though method M2 is by far the most inefficient. The condition number of A is larger than in Case 4, hence it is reasonable to expect that longer times are required to solve (3). Also, the number of iterations required for all three methods increases greatly. It is known that for convection dominated problems such as this, the ILU class of preconditioners can be very poorly conditioned [7]. This further influences the accuracy of $\tilde{S}_3$ which leads to the large condition number in Table 2.

Finally, a further comment on Table 2. There does not appear to be any direct relation between the magnitude of the condition number and the performance of the

solution method. As an example, in Case 1, the condition number of $\tilde{S}_1$ is two orders of magnitude higher than $\tilde{S}_3$ yet the former approximation results in a preconditioner that will solve Equation (1) quite effectively while a preconditioner based on the latter approximation stagnates. Conversely, for Case 3, $\tilde{S}_3$ has a larger condition number than $\tilde{S}_1$, yet $\tilde{S}_3$ performs (slightly) better.

3.4 Effectiveness of the Zero-fill strategy for X

In this section the differences in computing X by using a fixed amount of allowable element fill-in and employing a tolerance-based dropping strategy versus restricting X to have the same sparsity structure as B are compared. Figures 11–12 show the convergence histories for Cases 2 and 3 using method M1 and ICT preconditioning for the Schur complement equation. In each plot, the convergence history for the zero fill-in version of X as well using fill–in amounts of 10, 15 and 20. Case 1 cannot be similarly compared since the $\tilde{S}_3$ approximation stagnates for the set of parameters considered and this discussion does not apply to the $\tilde{S}_1$ and $\tilde{S}_2$ approximations. The corresponding condition numbers of the Schur complements are given in Table 4. Note that the convergence rates for the zero-fill and fill-in level 20 cases are nearly identical though the preconditioning times are about 2.5 to 3 times greater for a fill-in level of 20. The total time to compute X increases linearly with the fill-in allowed.

It is curious to note that as the fill-in level increases, the number of iterations decreases while the conditioning of the Schur complement approximation increases. These results indicate that the zero-fill version of X gives a better Schur complement approximation over the range of allowable fill-in considered when A is diagonally dominant in the sense that a good approximation can be obtained at a reasonable cost.

	Case 1		Case 2		Case 3	
Fill-in	K(X^TX)	nnz(X^TX)	K(X^TX)	nnz(X^TX)	K(X^TX)	nnz(X^TX)
0	2.47E(3)*	23340	6.16E(3)	182794	1.10E(3)	38839
10	8.00E(4)*	407096	1.22E(4)	108630	1.17E(4)	171699
15	1.44E(5)*	686928	1.69E(4)	164482	1.26E(4)	298176
20	1.69E(5)*	938748	1.76E(4)	216776	1.30E(4)	416978

Table 4: Comparison of restricting X to have the same sparsity structure as B versus using a limited amount of fill-in per row of X for Cases 1–3. A fill-in of 0 indicates that X is restricted to the same sparsity pattern as B. The '*' indicates that the solution process either stagnates or does not converge.

4 Conclusions

The block LU preconditioner presented in this work appears to be effective when combined with a global iterative approach in the solution of saddle point problems. In particular, the

computation of the Schur complement approximation $\tilde{S}_3$ using the zero-fill approach 2.3 appears to be both computationally efficient and numerically effective in the case where A is SPD and diagonally dominant.

For the generalized saddle point problem, convergent solutions can be obtained, but the preprocessing part which consists of computing the preconditioner is as yet inefficient relative to its symmetric counterpart. This is due to the relatively large time required by the ILUT routine to compute the preconditioning factors. One way to improve this would be to improve the efficiency of the ILUT routine.

References

[1] K. Arrow, L. Hurwicz, H. Uzawa, *Studies in nonlinear programming,* Stanford University Press, Stanford, CA, 1958.

[2] R. E. Bank, B. D. Welfert, H. Yserentant, *A class of iterative methods for solving saddle point problems,* Numer. Math., 55 (1990), pp. 645–666.

[3] J. H. Bramble, J. E. Pasciak, A. T. Vassilev, *Analysis of the inexact Uzawa algorithm for saddle point problems,* SIAM J. Numer. Anal., 5 (1997), pp. 1072–1092.

[4] J. H. Bramble,A. T. Vassilev, J. E. Pasciak, *Uzawa type algorithms for nonsymmetric saddle point problems,* To Appear, Math. Comp.

[5] H. Choi *Personal communication,* Univ. of Minnesota, Department of Aerospace Engineering and Mechanics, 1999.

[6] N. Dyn, W. Ferguson *The numerical solution of equality–constrained quadratic programming problems,* Math. Comput., 41 (1983), pp. 165–170.

[7] H. Elman, *A stability analysis of incomplete LU factorizations,* Math. Comput., 47 (1986), pp. 191–217.

[8] H. Elman, *Perturbation of eigenvalues of preconditioned Navier–Stokes operators,* Technical Report CS-TR-3559, Department of Computer Science, University of Maryland, (1996).

[9] H. Elman, *Preconditioning for the steady–state Navier–stokes equations at low viscosity,* Technical Report CS-TR-3712, Department of Computer Science, University of Maryland, (1996).

[10] H. Elman, D. Silvester, *Fast nonsymmetric iterations and preconditioning for Navier–Stokes equations,* Technical Report CS-TR-3283, Department of Computer Science, University of Maryland, (1994).

[11] G. Golub, A. Wathen, *An iteration for indefinite systems and its application to the Navier–Stokes equations,* SIAM J. Sci. Comput., 19 (1998), pp. 530–539.

[12] M. Jones, P. Plassman, *An improved incomplete Choleski factorization,* ACM TOMS, 21 (1995), pp. 5–17.

[13] A. Klawonn, *Block–triangular preconditioners for saddle point problems with a penalty term,* SIAM J. Sci. Comput., 19 (1998), pp. 172–184.

[14] A. Klawonn, *An optimal preconditioner for a class of saddle point problems with a penalty term,* SIAM J. Sci. Comput., 19 (1998), pp. 540-552.

[15] L. Little, *A finite–element solver for the Navier–Stokes equations using a preconditioned adaptive Bicgstab(l) method,* Ph.D. Thesis, Arizona State Univ, 1998.

[16] I. Perugia, V. Simoncini, M. Arioli, *Linear algebra methods in a mixed approximation of magnetostatic problems,* IAN-CNR Tech. Report 1060, 1997.

[17] T. Rusten, R. Winther, *A preconditioned iterative method for saddlepoint problems,* SIAM J. Matrix Anal. Appl., 13 (1992), pp. 887-904.

[18] Y. Saad, *Preconditioning techniques for nonsymmetric and indefinite linear systems,* J. Comput. Appl. Math., 24 (1988), pp. 89-105.

[19] Y. Saad, *Iterative methods for sparse linear systems,* PWS, New Jersey, 1996.

[20] Y. Saad, J. Zhang, *BILUTM: A domain–based multi–level block ILUT preconditioner for general sparse matrices,* Technical Report UMSI–97–118, Minnesota Supercomputing Institute, Minneapolis, MN, 1998.

[21] D. J. Silvester, A. J. Wathen, *Fast iterative solution of stabilised Stokes systems Part II: using general block preconditioners,* SIAM J. Numer. Anal., 31 (1994), pp. 1352–1367.

[22] R. Verfurth, *A combined conjugate gradient–multigrid algorithm for the numerical solution of the Stokes problem,* IMA J. of Num. Anal., 4 (1984), pp. 441–455.

[23] A. J. Wathen, D. J. Silvester, *Fast iterative solution of stabilised Stokes systems Part I: Using simple diagonal preconditioners,* SIAM J. Numer. Anal., 30 (1993), pp. 630–649.

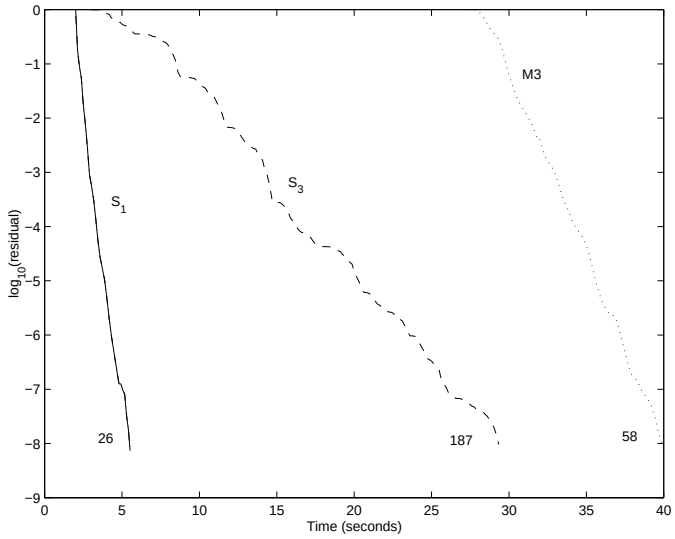

Figure 3: Convergence histories for Case 1 using method M1. The matrix A is not preconditioned and the Schur complement is preconditioned by ICT. The trailing number is the total number of iterations required and the preconditioning time is indicated by the starting time of the convergence curve.

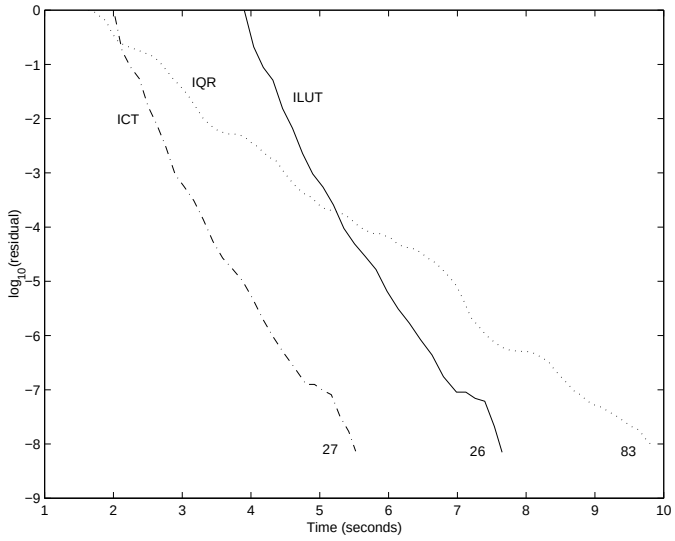

Figure 4: Convergence histories for Case 1 using method M1 and ICT, ILUT and IQR preconditioning for the Schur complement equation. The $\tilde{S}_1$ Schur complement approximation is used along with identity preconditioning for A. Preconditioning time is indicated by the starting time of the convergence curve.

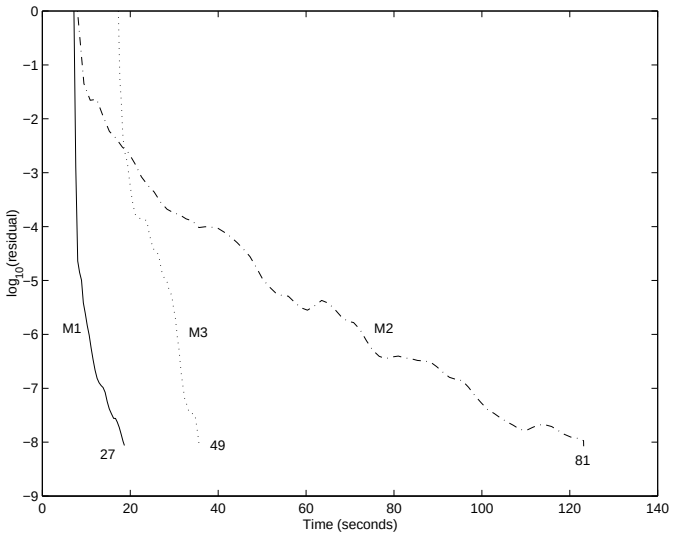

Figure 5: Convergence histories for Case 2 using ICT preconditioning for the Schur complement equation and A. The Schur complement approximation is $\tilde{S}_3$.

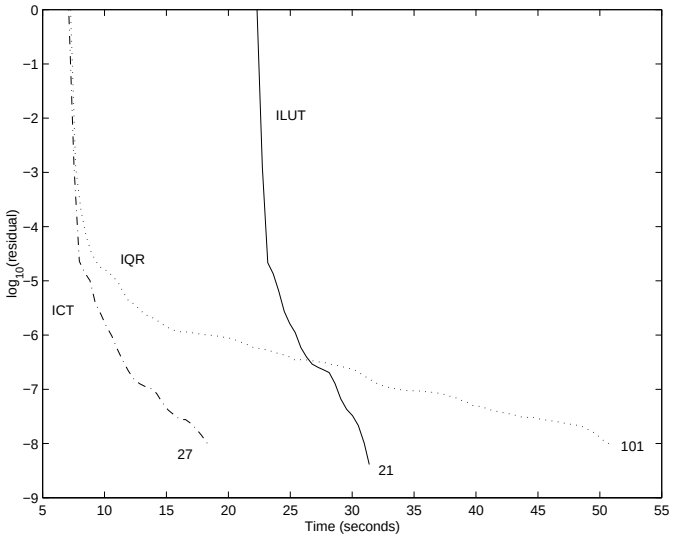

Figure 6: Convergence histories for Case 2 for ICT, ILUT and IQR preconditioning for the Schur complement equation. The $\tilde{S}_3$ Schur complement approximation is used along with ICT preconditioning for A.

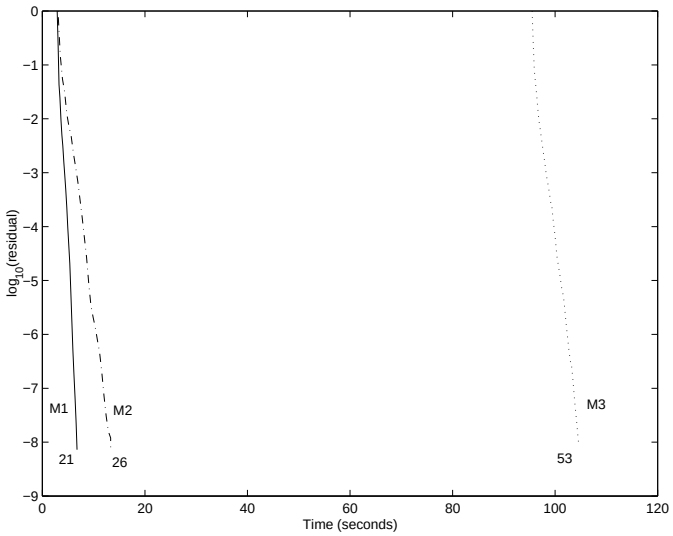

Figure 7: Convergence histories for Case 3 using ICT preconditioning for the Schur complement equation and A. The $\tilde{S}_3$ Schur complement approximation is used.

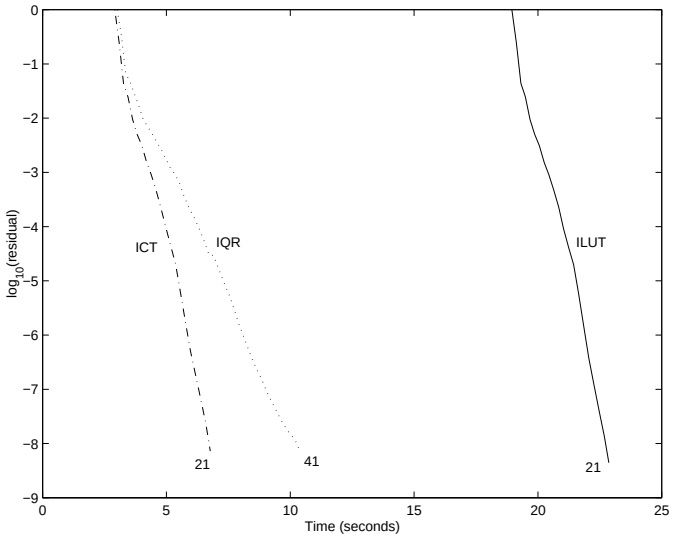

Figure 8: Convergence histories for Case 3 for ICT, ILUT and IQR preconditioning for the Schur complement equation. The $\tilde{S}_3$ Schur complement approximation is used along with ICT preconditioning for A.

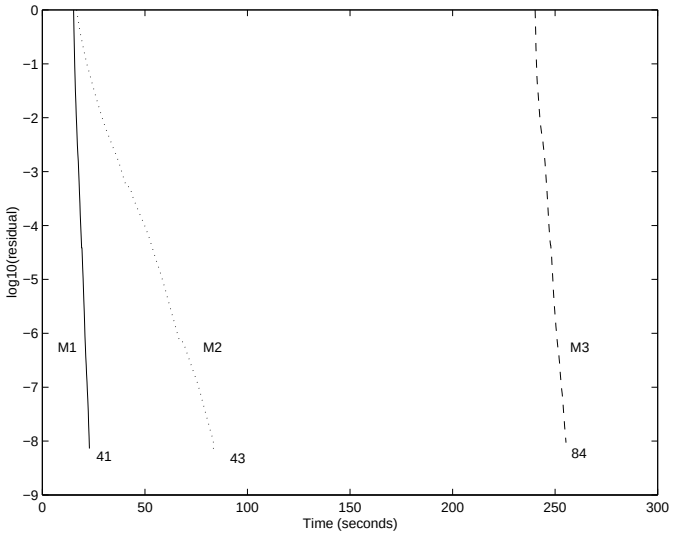

Figure 9: Convergence histories for Case 4. ILUT preconditioning is used for A and the Schur complement.

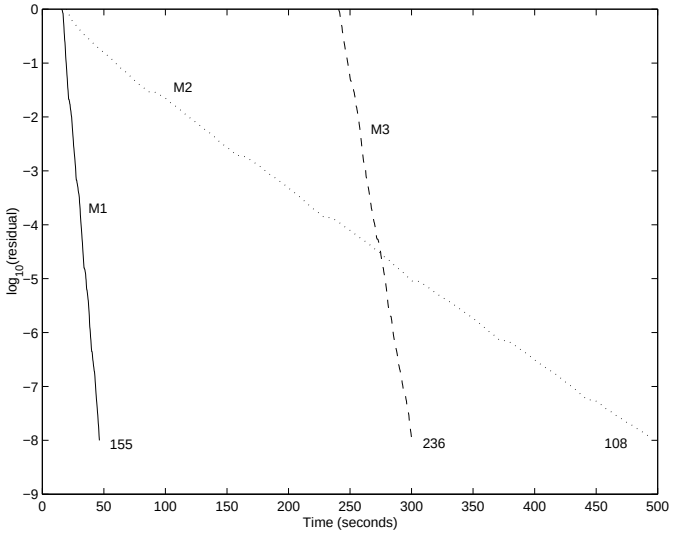

Figure 10: Convergence histories for Case 5. ILUT preconditioning is used for A and the Schur complement.

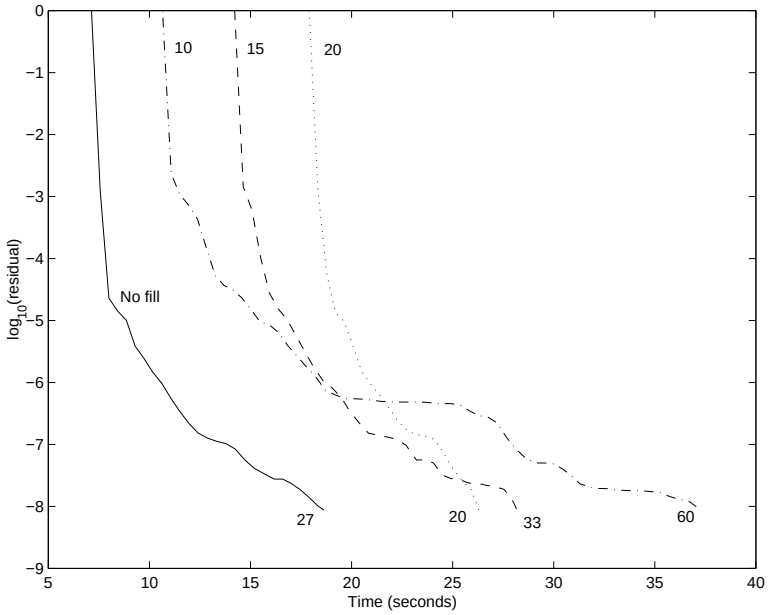

Figure 11: Comparison of convergence histories for Case 2 when X is computed from Algorithm 2.2 using a zero-fill approach and allowing X to have a maximum of 10, 15 and 20 elements per row. ICT preconditioning is used for the Schur complement equation.

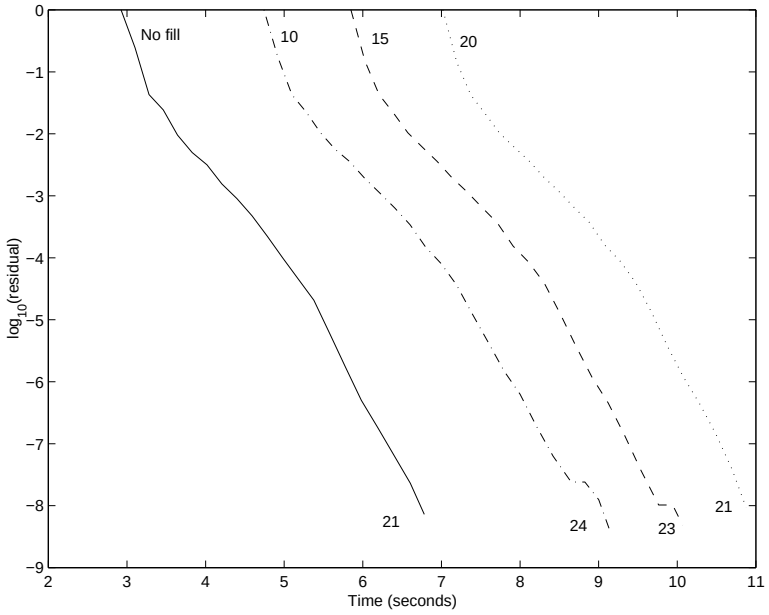

Figure 12: Comparison of convergence histories for Case 3 when X is computed from Algorithm 2.2 using a zero-fill approach and allowing X to have a maximum of 10, 15 and 20 elements per row. ICT preconditioning is used for the Schur complement equation.