Experimental Study of ILU Preconditioners for Indefinite Matrices*

Edmond Chow and Yousef Saad
Department of Computer Science, and
Minnesota Supercomputer Institute
University of Minnesota
Minneapolis, MN 55455

June 8, 1997

Abstract

Incomplete LU factorization preconditioners have been surprisingly successful for many cases of general nonsymmetric and indefinite matrices. However, their failure rate is still too high for them to be useful as black-box library software for general matrices. Besides fatal breakdowns due to zero pivots, the major causes of failure are inaccuracy, and instability of the triangular solves. When there are small pivots, both these problems can occur, but these problems can also occur without small pivots. Through examples from actual problems, this paper shows how these problems evince themselves, how these problems can be detected, and how these problems can sometimes be circumvented through pivoting, reordering, scaling, perturbing diagonal elements, and preserving symmetric structure. The goal of this paper is to gain a better practical understanding of ILU preconditioners and help improve their reliability.

1 Introduction

The incomplete LU factorization preconditioners were originally developed for M-matrices, for which properties such as existence and a form of stability can be proved [25] (see also [40]). However, ILU preconditioners have been successfully applied in much more general situations. In the general symmetric case, diagonal perturbations of the matrix are required to help guarantee the existence of a symmetric factorization [23, 24, 27]. These perturbations may be applied before the factorization, or during the factorization when a small or negative pivot is encountered. In the nonsymmetric case, there may be another problem: the incomplete factors L and U may be much worsely conditioned than the original matrix A. A coupled effect is that the long recurrences associated with solving with these factors are unstable [6, 16]. A remedy is also to use diagonal perturbations, this time to make the factors diagonally dominant [27, 37, 17], but the perturbations in this case may need to be very large.

*Work supported in part by the National Science Foundation under grant NSF/CCR-9618827 and in part by NASA under grant NAG2-904.

ILU preconditioners have also been applied successfully to indefinite matrices, i.e., matrices with indefinite symmetric parts. However, the problems described above can be more severe and more probable:

1. *Inaccuracy due to very small pivots.* Pivots can be arbitrarily small, and often lead to unstable and therefore inaccurate factorizations, i.e., the size of the elements in the factors can grow uncontrollably, and the factorization becomes inaccurate. By accuracy, we mean the closeness of LU to A. However, some pivots, particularly near the end of a factorization, may not be used in the factorization, and small values of these pivots have no effect on the stability of the factorization.

2. *Unstable triangular solves.* The incomplete factors of an indefinite matrix are often far from being diagonally dominant, which makes unstable triangular solves more likely. A sign of unstable triangular solves is when $\|L^{-1}\|$ and $\|U^{-1}\|$ are extremely large while the off-diagonal entries of L and U are reasonably bounded. If there are very small pivots, then the triangular solves will be unstable. However, this problem also occurs without the presence of very small pivots.

In *complete* LU factorizations, the main difficulty is the first one: small pivots leading to unstable and inaccurate factorizations. Large elements in the factors directly impact the backward error. The common remedy here is to use a pivoting scheme so that the size of the elements in the factors can be bounded. The second problem of unstable triangular solves is rare for complete factorizations, and thus the problem seems to be related to the effect of dropping nonzeros in incomplete factorizations.

In contrast, for incomplete factorizations, the first problem is much less severe. The growth of the elements in the factors depends on how often each element is updated. In incomplete factorizations, each element is updated far fewer times than in complete factorizations. As long as extremely small pivots are avoided, the growth of the elements in the incomplete factors is not a problem. However, triangular solves can be unstable even though a factorization is stable.

There are two other problems that exist for incomplete factorizations which have not yet been mentioned:

3. *Inaccuracy due to dropping.* Factorizations are made incomplete by dropping nonzeros to make the factorization more economical to store, compute, and solve with. Each nonzero that is dropped contributes to the 'error' in the factorization, i.e., contributes to E in the relation $A = LU + E$. However, this error is not a very serious problem as long as accuracy can be improved by allowing more fill-in or using a different dropping scheme or sparsity pattern. If the inaccuracy is not due to dropping, but is due instead to small pivots and an unstable factorization, for example, then simply increasing the allowed fill-in will generally not help.

4. *Zero pivots.* The pivots of an incomplete factorization can be arbitrarily small, even zero. The most common cause of zero pivots is an irregular structure or ordering of the matrix, one that has a null column above or null row to the left of a zero diagonal element. This is referred to as a *structurally* zero pivot. When a matrix has zeros on the diagonal, this problem can be common unless a careful ordering is used. Zero pivots can also be caused

numerically, i.e., when a nonzero diagonal element becomes zero. Numerically zero pivots can be caused by very small pivots which cause a row to be 'swamped out' by an extremely large factor of the pivotal row.

The above four problems will often occur together, and one problem may mask another. For example, a factorization that is initially inaccurate due to dropping can produce small pivots; these small pivots in turn can make the factorization unstable and even more inaccurate; the inaccuracy may lead to a small pivot which induces a numerically zero pivot. When the factorization fails on the zero pivot, none of the preceding problems may have been noticed.

The four problems may also interact in complex ways that are difficult to predict. For example, by allowing more fill-in to improve the accuracy, the new factorization may happen to have smaller pivots; this in turn may cause the triangular solves to be unstable.

To try to understand what can happen in an incomplete factorization, a number of statistics can be monitored. These statistics, shown in Table 1, can be monitored during the course of a factorization, or after the factorization has been computed.

Statistic	Meaning
condest	$\|(LU)^{-1}e\|_\infty, \quad e = (1, \ldots, 1)^T$
1/pivot	size of reciprocal of the smallest pivot
max(L+U)	size of largest element in L and U factors

Table 1: Statistics that can be used to evaluate an incomplete factorization.

Probably the most useful statistic is 'condest,' which measures the stability of the triangular solves. It simply measures $\|(LU)^{-1}e\|_\infty$ where e is the vector of all ones. Note that this statistic is also a lower bound for $\|(LU)^{-1}\|_\infty$ and indicates a relation between unstable triangular solves and poorly conditioned L and U factors. We refer to this statistic as the condition estimate of $(LU)^{-1}$.

The second statistic is needed to help interpret this condition estimate. The condition estimate will certainly be poor if there are very small pivots. Thus when condest is very large, it should be compared to the size of the reciprocal of the smallest pivot. If these two quantities are about the same size, then we assume that $\|(LU)^{-1}\|_\infty$ is large due to at least one very small pivot. If condest is much larger than 1/pivot (e.g., condest greater than the square of 1/pivot) then we assume that the recurrences associated with the triangular solves are unstable.

The third statistic is the size of the largest element in the L and U factors. A large value of this statistic in relation to the size of the elements in A indicates an unstable and thus inaccurate factorization. We will see in the numerical experiments in Section 6 that for incomplete factorizations, max(L+U) is never large unless 1/pivot is large. In addition, when max(L+U) is large, it is usually about the same size as 1/pivot (assuming that the maximum entries in A are $O(1)$). Occasionally, we will find very small pivots, but max(L+U) remains small. This occurs when the small pivot is not used in the factorization.

Usually, these statistics are only meaningful when their values are very large, e.g., on the order of 10^{15}. Extremely large values, particularly of the condition estimate, can be used to predict when the ILU preconditioner will fail. When all three statistics are reasonably small, and an ILU preconditioner does not help an iterative method converge, it is our experience that the cause of failure is inaccuracy due to dropping.

The chart in Figure 1 summarizes some of these statements. Note that there are no cases of small condest and large 1/pivot. Also, although there may be cases when 1/pivot is very large, as long as condest is much larger, we will still label this as an unstable triangular solve.

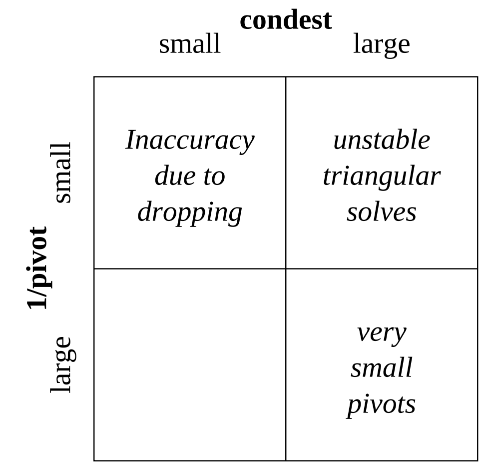


Figure 1: How to interpret the statistics.

The results of numerical tests are given in Section 6, and the actual values of these statistics will be shown and discussed. In the next few sections, we discuss the particular difficulties of level-based and threshold-based factorizations, and show the relative merits of pivoting, preserving symmetric structure, and shifting or perturbing the diagonal of A to try to make ILU preconditioners more reliable.

2 Dropping strategies for ILU

The 'error' in an incomplete factorization LU of a matrix A is the term E in

$$A = LU + E. \tag{1}$$

By only dropping small nonzero entries in L and U, the size of the entries in E can be kept small. This is important because, for symmetric linear systems, the size of E is very strongly related to the convergence rate of an ILU-preconditioned iteration [14].

However, for nonsymmetric and for indefinite problems the performance is much less predictable. The factorization error is important, but just as important is the stability of the triangular solves, i.e., the norm of the *preconditioned* error $L^{-1}EU^{-1}$ in the preconditioned version of (1)

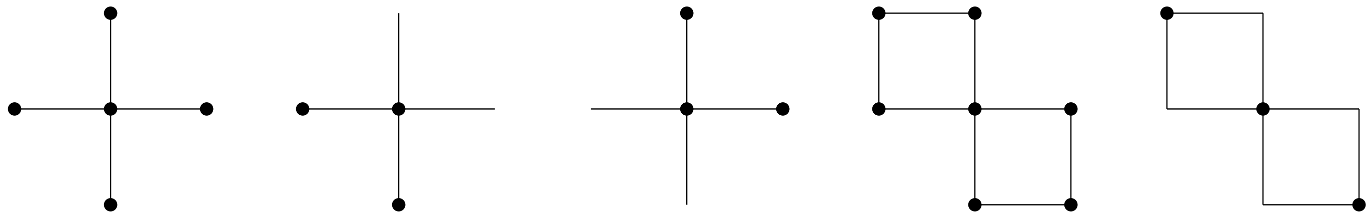
$$L^{-1}AU^{-1} = I + L^{-1}EU^{-1}.$$

When A is indefinite or has a large nonsymmetric part, then L^{-1} and U^{-1} may have very large norms, causing $\|L^{-1}EU^{-1}\|$ to be very large.

For indefinite matrices, the behavior of ILU preconditioners that drop small nonzero entries predicted by the matrix 'structure' and methods that drop based on matrix 'values' can be very different. In particular, the latter methods can be more accurate, but are also more prone to unstable triangular solves.

2.1 Dropping fill-in based on matrix structure

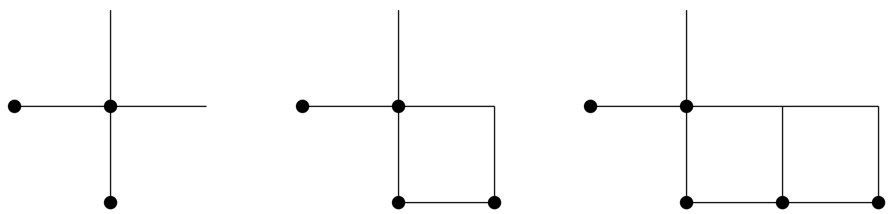
The original incomplete factorizations were developed for solving finite difference equations for elliptic partial differential equations. For these problems, the structure of the incomplete triangular factors was chosen based on the structure of the gridpoint operators [7, 28, 29, 39] (see also the review [9]) and the resulting structure of the error matrix E. In most cases, the gridpoint operator was a five-point stencil, and the stencil for the lower (upper) triangular factor was chosen to have the same pattern as the lower (upper) triangular part of the original stencil. These stencils are illustrated in Figure 2, along with stencils for the approximation LU and its error $E = A - LU$.



(a) (b) (c) (d) (e)

Figure 2: Stencils of (a) A, (b) L, (c) U, (d) LU, (e) $A - LU$.

To get a more accurate factorization, a larger stencil for the factors can be chosen, for example, by attempting to reduce the error $A - LU$. This can be done by considering the stencil of LU as the new stencil to be approximated [21]. Successively larger stencils for the lower-triangular factor defined this way are shown in Figure 3.



(a) (b) (c)

Figure 3: Increasingly larger stencils for L.

Incomplete factorizations were named as such when an approximate Gaussian elimination process was defined that gives sparse factors of any given pattern [25]. This generalized the earlier work on stencils to arbitrarily structured M-matrices. Choices of effective sparsity patterns for the factors were given for five- and seven-point matrices [26]. Note that the sparsity pattern should include the full diagonal, even if there are zeros on the diagonal, as in the case of some indefinite matrices.

To get more accurate factorizations for general sparse matrices, the concept of level-of-fill was introduced [41]. Suppose a matrix A has diagonal elements of size $O(\epsilon^0)$ and off-diagonal elements of size $O(\epsilon^1)$, with $\epsilon<1$, and the exponent on ϵ indicating the *level* of the nonzero element. As an incomplete factorization proceeds, an element a_{ij} is updated with an expression of the form

$$a_{ij} := a_{ij} - a_{ik}a_{kj}.$$

If lev_{ij} is the current level of element a_{ij}, $i \neq j$, then according to the model matrix, the size of the updated element is approximately

$$\epsilon^{lev_{ij}} - \epsilon^{lev_{ik}} \times \epsilon^{lev_{kj}},$$

i.e., roughly the maximum of the two sizes $\epsilon^{lev_{ij}}$ and $\epsilon^{lev_{ik}+lev_{kj}}$. This can be used to compute a level for each element before the actual factorization. By excluding nonzeros in the factorization that have high level, i.e., that are created by a chain of induced nonzeros, then essentially small nonzeros are dropped. For five-point matrices, retaining successively higher level elements gives the same successively more accurate stencils as those in Figure 3. Note that to agree with the literature, we need to redefine level as one less than what was used above. The ILU(k) factorization is thus defined as a factorization that retains all elements with level up to k. ILU(0) retains only the original nonzeros of the matrix.

There is also a characterization of level-of-fill based on the graph of the original matrix [12]. To illustrate this, consider the graph of Figure 4 from [20] and the elimination of the nodes in the numbered order. In the *complete* factorization of the associated matrix, there will be a fill-in between nodes 4 and 6, from the successive eliminations of nodes 1 and 2 [31]. This is because there exists a path (4, 2, 1, 6) in the graph. The level of the fill-in is one less than the length of the shortest path between nodes 4 and 6 through the eliminated nodes 1 and 2. In this case, the level is 2. Assuming that the nodes are eliminated in the natural order, then in general, the level of element a_{ij} is equal to one less than the length of the shortest path $(i, u_1, \ldots, u_m, j)$ in the original matrix, where the u_k, the eliminated nodes, are numbered less than both i and j. Note the very strong dependence on the order of elimination.

Each new edge in the path corresponds to multiplying by ϵ and inducing a nonzero in the model matrix. This graph-based characterization can also be used to determine the stencils in Figure 3. The important nonzeros or edges determined by these structural dropping schemes are, in some sense, those between nearby nodes.

In practice, any form (order of the loops) of Gaussian elimination may be used to compute an incomplete factorization. However, the most computationally efficient form is probably the row-wise or column-wise form. A full-length work vector is used to hold the current row or column that is being computed, which helps minimize searching for nonzero entries. The 'submatrix' form is more expensive to use, but it is more flexible and makes it possible to perform symmetric pivoting [5, 12]. The 'bordered' form will be introduced in Section 4.

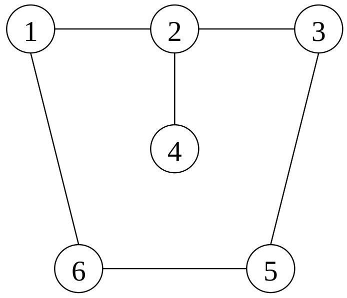


Figure 4: Example graph.

Algorithm 2.1 illustrates the row-wise incomplete factorization of a matrix A. This form is suitable for sparse matrices stored in row-contiguous data structures. The algorithm computes computes each row of L and U together, where L is unit lower triangular, and U is upper triangular. The predetermined sparsity pattern of $L + U$ is S, and w is the full-length work vector mentioned above.

Algorithm **2.1 Row-by-row ILU factorization**

For $i = 1, \ldots, n$, *do*
    $w_j := a_{i,j}$, $(i,j) \in S$, $\quad w_j = 0$ *otherwise*
    For $k = 1, \ldots, i-1$ *and if* $(i,k) \in S$, *do*
        $w_k := w_k / u_{kk}$
        For $j = k+1, \ldots, n$ *and if* $(i,j) \in S$, *do*
            $w_j := w_j - w_k u_{kj}$
        Enddo
    Enddo
    $l_{ij} := w_j, \quad j = 1, \ldots, i-1$ *and* $l_{ii} := 1$
    $u_{ij} := w_j, \quad j = i, \ldots, n$
Enddo

2.2 Dropping strategies based on numerical threshold

There are many cases of matrices, particularly non-diagonally dominant and indefinite matrices, where the model of the matrix for the level-of-fill concept above is inappropriate. For these matrices, level-of-fill may be less effective at predicting the locations of the largest entries in the factorization. As an alternative to dropping techniques based on structure, fill-in can be dropped *during* the factorization, based on their *numerical* size [27, 34, 36, 44]. This is a kind of greedy approach to minimizing E in (1).

Numerical dropping strategies generally yield more accurate factorizations with the same amount of fill-in than level-of-fill methods. This is true even for some diagonally dominant M-matrices. In general, ILU based on numerical dropping can solve more problems, and in fewer steps than ILU based on matrix structure. However, there are some drawbacks which will be described at the end of this subsection.

To describe a threshold-based numerical dropping strategy, Figure 5 shows $A = LU$ in the row-wise computation of row i of the factorization. Rows 1 to $i-1$ in L and U have been completed, and rows $i+1$ to the end in A have not yet been accessed. The matrix equation

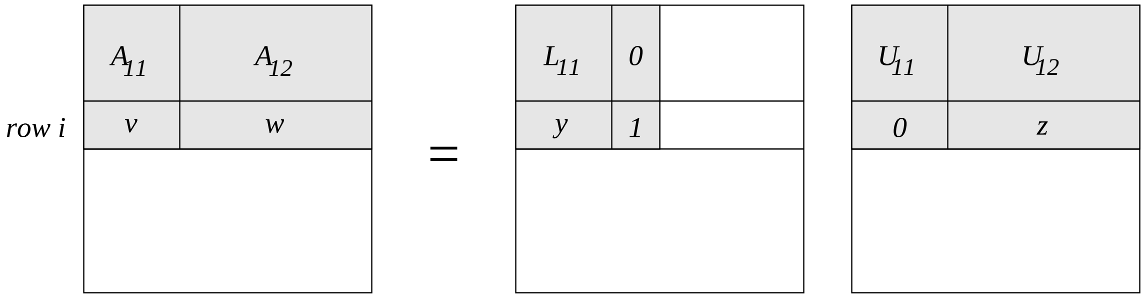


Figure 5: $A = LU$ in the computation of row i of the factorization.

represented by the shaded regions is

$$\begin{pmatrix} A_{11} & A_{12} \\ v & w \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ y & 1 \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & z \end{pmatrix} \tag{2}$$

which means that y and z (the rows to be computed in L and U) can be determined by first solving a lower-triangular system

$$U_{11}^T y^T = v^T \tag{3}$$

and then substituting y into

$$z = w - yU_{12}. \tag{4}$$

The simplest way to perform numerical dropping is to drop small entries in y and z after these vectors are computed. For example, given a parameter *droptol*, entries in y less than *droptol* are set to zero, while entries in z less than $z_1 \times$ *droptol* are set to zero, where z_1 is the first component of the vector z. (A threshold of $\|z\| \times$ *droptol* is often used instead if there is a danger that z_1 is very small.)

However, it is also possible to drop small entries in y^T during the triangular solve (3). Algorithm 2.2 shows this operation without dropping. Note that the triangular solve is performed with saxpy operations because only the columns of U_{11}^T are available, and not all columns of U_{11}^T may be needed.

ALGORITHM **2.2 Solving $U_{11}^T y^T = v^T$ for the factorization using saxpy operations, without dropping**

```
y := v
For k = 1, ..., i-1, do
    y_k := y_k / u_kk
    For j = k+1, ..., i-1, do
        y_j := y_j - y_k u_kj
    Enddo
Enddo
```

To make this algorithm approximate, values of y_k less than *droptol* can be dropped after Line 3. Then the work in loop 4–6 (i.e., column k of U_{11}^T) can be saved. In fact, this type of dropping introduces less error into the factorization. Consider the factorization

$$\begin{pmatrix} b & f \\ e & c \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ e/b & 1 \end{pmatrix} \begin{pmatrix} b & f \\ 0 & c - ef/b \end{pmatrix}. \tag{5}$$

If e/b is small and is dropped before it is used, then the error E of the factorization is

$$\begin{pmatrix} b & f \\ e & c \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} b & f \\ 0 & c \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ e & 0 \end{pmatrix}.$$

If e/b is dropped *after* it is used in the loop 4–6, then the error in the factorization is

$$\begin{pmatrix} b & f \\ e & c \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} b & f \\ 0 & c - ef/b \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ e & ef/b \end{pmatrix}.$$

Surprisingly, this means it can be advantageous to sparsify a matrix (i.e., drop small values) before starting an incomplete factorization.

Although numerical threshold-based ILU is generally more accurate than level-based ILU, their differences in behavior with respect to other factors must be considered. For example, by systematically retaining the largest elements in L and U in threshold-based ILU, the factorization is more prone to unstable triangular solves, because the off-diagonal elements are generally larger. The largest $(LU)^{-1}$ condition estimates that we see are those produced by threshold-based ILU rather than level-based ILU.

An even more serious problem is the erroneously large entries that may have been computed via a small and inaccurate pivot. In threshold-ILU, these large entries are propagated during the factorization due to their size. This does not happen when a level-of-fill rule is used.

Practical implementations of threshold-based ILU include an additional parameter besides *droptol* called *lfil*. This is the maximum number of nonzeros in each row y and z when a row of the factorization is computed, i.e., the largest *lfil* entries are retained in each of y and z. This implementation is called ILUT(*droptol*, *lfil*) [34]. The *lfil* parameter makes the storage requirements for the preconditioner known beforehand.

However, by limiting the fill-in in each row but not each column, a very nonsymmetric preconditioner may be produced. Figure 6 illustrates the pattern of a pair of L and U factors together, for a matrix that has a symmetric pattern. The vertical striping in the Figure is characteristic of the problem. A consequence is that columns with small elements may never receive fill-in, and never create fill-in onto a possibly small or zero diagonal element. In Section 4, a bordered form of factorization will be described that circumvents this problem.

A common cause for this problem is the unequal scaling of the rows or columns of a matrix when there are different types of equations and variables. Thus, when threshold-based ILU preconditioners are used, it is often recommended that matrices are first scaled so that each column has unit 2-norm, and then scaled again so that each row has unit 2-norm. However there are side-effects to this scaling: it may improve the conditioning of the matrix, but it may increase the degree of non-normality of the matrix.

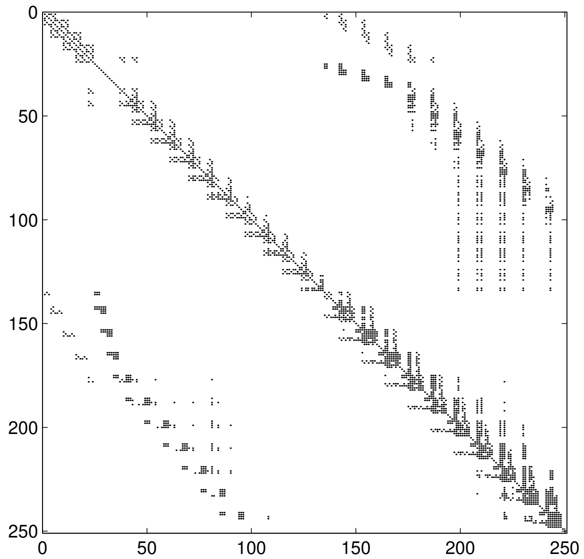


Figure 6: Poor ILUT pattern (of $L+U$).

3 Pivoting for incomplete factorizations

In *complete* factorizations, pivoting is required for nonsymmetric and indefinite matrices to prevent excessive growth of the entries of the factors. As mentioned in Section 1, this type of instability is not a serious problem for incomplete factorizations, and thus pivoting has not generally been used. However, there still needs to be a mechanism to help avoid zero and very small pivots. Probably the most popular mechanism is to replace these small pivots by a larger value, a technique that we discuss in Section 5. However, this technique may cause too much inaccuracy in the preconditioner, particularly if many replacements need to be made. An alternative which is particularly suitable for very unstructured matrices with many structurally zero pivots is to use *pivoting*.

The simplest way to incorporate pivoting in an incomplete factorization computed row-wise is to perform column (partial) pivoting. This is because no column data structures are available for the searching required for row pivoting. In row i of the factorization, after z is computed by (4), the column with the largest entry in magnitude in z is exchanged with column i. The entry z_1, which will be the pivot for that row, is the largest entry in z.

In the implementation of column pivoting, no actual column exchanges are made, and the new row indices are determined through permutation vectors. The permutation vectors are updated with each column exchange. This variant of incomplete factorization combined with the dropping strategy of ILUT is called ILUTP. See [33] for more details.

Unlike the case with complete factorizations, pivoting for incomplete factorizations cannot guarantee that a nonzero pivot can always be found, i.e., z may be all zero and failures due to zero pivots can still occur. In fact, a poor pivoting sequence can occasionally trap a factorization into a zero pivot, even if the factorization would have succeeded without pivoting.

A tolerance parameter *permtol* can be included to determine whether or not to perform a permutation. The largest nondiagonal element a_{ij} that satisfies *permtol* $\times|a_{ij}|>|a_{ii}|$ is permuted into the diagonal position. This type of parameter is used in sparse direct factorization codes to balance stability with the preservation of sparsity.

In *block* incomplete factorizations (BILU), where each entry in Algorithm 2.1 is actually a small dense block, a form of pivoting is also occurring. When the inverse of u_{kk} is taken in Line 4 of the Algorithm, it is assumed to be computed with pivoting if necessary. Thus, one way to deal with zero diagonal entries that might lead to zero pivots is to use *blocking:* guarantee each zero diagonal entry is within a small nonsingular block. ILUTP can be used to simulate this type of pivoting by only searching for pivots within the current block.

The idea of *blocking* is similar to the idea of diagonal pivoting for complete factorizations of symmetric indefinite matrices [8, 13], where 2 by 2 pivot blocks are allowed. Here, permutations are allowed to find a 2 by 2 pivot block that is well conditioned.

4 Preserving symmetric structure for threshold-based ILU

For matrices with symmetric structure, threshold-based ILU will not generally produce L and U factors that are symmetric to each other, particularly when the *lfil* parameter is used. However, the symmetric structure can be preserved with an incomplete form of LDU Gaussian elimination based on bordering [11, 30, 35]. Let A_{k+1} be the $(k+1)$-st leading principal submatrix of A and assume we have the decomposition $A_k = L_k D_k U_k$. Then we can compute the factorization of A_{k+1} using

$$\begin{pmatrix} A_k & v_k \\ w_k & \alpha_{k+1} \end{pmatrix} = \begin{pmatrix} L_k & 0 \\ y_k & 1 \end{pmatrix} \begin{pmatrix} D_k & 0 \\ 0 & d_{k+1} \end{pmatrix} \begin{pmatrix} U_k & z_k \\ 0 & 1 \end{pmatrix}$$

in which

$$\begin{aligned} z_k &= D_k^{-1} L_k^{-1} v_k && (6) \\ y_k &= w_k U_k^{-1} D_k^{-1} && (7) \\ d_{k+1} &= \alpha_{k+1} - y_k D_k z_k. && (8) \end{aligned}$$

Thus, we obtain each row and column of the factorization by approximately solving two lower triangular systems and computing a scaled dot product.

The lower triangular systems (6) and (7) are solved the same way as system (3) was solved, i.e., with numerical dropping. However, in this case, the lower triangular matrices are only available by rows, not by columns. A companion data structure that gives access to the columns is needed. For details, see [11], which also describes how sparse approximate solutions to the triangular systems can be found using approximate inverse techniques.

In order for z_k and y_k to have the same sparsity pattern, the systems (6) and (7) are solved simultaneously. Corresponding entries in y_k and z_k are both kept or both dropped, to try to maximize the absolute value of $y_k D_k z_k$. Half of the book-keeping for the sparse computations can be saved because of the symmetric pattern.

There are two advantages to this form of factorization. First, fill-in onto the diagonal is guaranteed as long as all v_k and w_k are nonzero. Second, since L_k and U_k are available after step k, a running condition estimate $\|(L_k U_k)^{-1}\|_\infty$ can be monitored.

5 Stabilized ILU

One possibility to affront the problem of small pivots is simply to replace them by larger values. Algorithmically, the new pivots should be chosen large enough to ensure that they do not create extremely large off-diagonal elements. Small pivots can lead to unstable and inaccurate factorizations, and unstable triangular solves. Thus we call such a technique a *stabilized* incomplete factorization.

The trade-off is always between stable factorizations and solves, and a factorization that is accurate or close enough to the original matrix A. It is clear that if the matrix is diagonally dominant, or well-conditioned, stabilization is not necessary, and any modification to the original matrix will cause the factorization to be inaccurate. On the other hand, some matrices will give factorizations that are unstable and therefore inaccurate without stabilization. Stabilization will help here, but too large a stabilization (e.g., too large a diagonal shift) will again cause the factorization to be inaccurate. It is obvious that a successful balance between these two may not always be found, in which case some other technique must be brought into play.

For positive definite matrices, Kershaw [23] suggested replacing negative or zero pivots with small positive values, and continuing with the factorization. For symmetric incomplete factorizations by threshold, Munksgaard [27] proposed the same kind of modification, making the pivot element comparable to the sum of the magnitudes of the off-diagonal elements in a row. Manteuffel [24] proposed the factorization of a shifted matrix $A + \alpha I$, and when A is symmetric, proved that there exists a scalar $\alpha>0$ such that the factorization for any sparsity pattern exists. Robert [32] later extended this result to positive real matrices. Even before incomplete factorizations were used widely, Jennings and Malik [22] augmented the entries on the diagonal of a sparsified matrix to guarantee it is positive definite for a complete factorization.

Besides guaranteeing existence, Manteuffel [24] noticed that the shift α that gave the best convergence of the iterative method was not the smallest one that makes the factorization exist, but one slightly larger. The shift should make the pivots large enough so that the matrix is not too poorly conditioned. Van der Vorst [37] found the same result for nonsymmetric matrices, and suggested modifications to the diagonal to make the resulting factors diagonally dominant. He called this a 'stabilized' incomplete factorization.

In general, a major difficulty is the determination of the threshold value for the pivots, or the shift α. For irregularly structured symmetric matrices, Saad [35] gave a heuristic formula for the shift to help ensure that each pivot will be greater than some small positive value. Numerical experiments for positive definite matrices show that convergence improves sharply as α is increased toward the optimal α, and then deteriorates slowly [24, 35].

Shifted or stabilized factorizations are not to be confused with modified ILU (MILU) factorizations [21] where the row-sum criteria

$$Ae = LUe, \quad e = (1, 1, \cdots, 1)^T$$

is satisfied by modifying the diagonal of L or U. For M-matrices, the modification actually decreases the size of the pivots, making the factorization less stable. Perturbed factorizations that add a small value to the diagonal opposite in direction to the modification, help guarantee a bound on the largest eigenvalue of the preconditioned system; see [4, Ch. 10] for a review.

These methods apply to elliptic problems in one variable, where they lower the order of the spectral condition number of the preconditioned matrix.

Relaxed ILU (RILU) [2, 3] parameterizes the fraction of the modification to perform, giving it the same effect as the perturbation. Negative relaxation factors in RILU have a stabilizing effect for M-matrices. They were used for multigrid smoothing by Wittum [42] in his ILU_β method. The diagonal is augmented with β times the sum of the magnitudes of the dropped elements. ILU_0 corresponds to the regular, unmodified factorization, ILU_{-1} corresponds to MILU, and ILU_1 corresponds to the modification of Jennings and Malik [22]. For elliptic problems that are not M-matrices, modification may also have a stabilizing effect if it increases the value on the diagonal. Elman [17] used this as part of his criteria to modify certain rows and not others, in his stabilized factorization based on RILU.

Recently, the method of diagonal compensation [1] has been developed for preconditioning positive definite matrices with incomplete factorizations. Essentially, the SPD matrix is modified into an M-matrix, for example, by dropping positive off-diagonal elements and adding them to the diagonal. An ILU factorization computed on this M-matrix (which must exist) is often a good preconditioner for the original matrix. This can be viewed as another form of stabilization.

Dynamic stabilization strategies

When we focus on nonsymmetric and indefinite problems, negative pivots are acceptable and even expected. However, shifts such as $A + \alpha I$ are inadequate because they may shift the eigenvalues of A arbitrarily close to the origin; shifts of α may *decrease* the magnitude of the pivot. To avoid this, one can use a different shift for each row, computing them dynamically, during the factorization. The sign of the shift depends on the sign of the pivot. This idea will be tested in Section 6.

This shift is usually much larger than what is required to plainly avoid very small pivots. The larger shift has the effect of making the problem much better conditioned. Stabilization essentially amounts to the factorization of a better conditioned matrix. This is an important effect of stabilization that was discovered experimentally [24, 37].

For block incomplete factorizations, the pivot is a block. The equivalent of a small pivot in this case is a block that is very poorly conditioned, with an inverse that has very large entries. It is possible to perform a shift for a block by shifting its singular values away from zero [43].

Given the singular value decomposition of a block $A = U\Sigma V^T$, a shifted inverse

$$A^{-1} \approx V\bar{\Sigma}^{-1}U^T \tag{9}$$

can be produced, where $\bar{\Sigma}$ is Σ with its singular values thresholded by a function of the largest singular value, such as $\alpha\sigma_1$, where $0 \leq \alpha \leq 1$ is a parameter. This approximate inverse has condition number no worse than $1/\alpha$.

6 Numerical experiments

6.1 Test matrices

Test matrices from a wide variety of applications were selected from the Harwell-Boeing, UMFPACK, and SPARSKIT collections. Many of these matrices are available from 'MatrixMarket,' a repository organized by the National Institute of Standards and Technology.

Results are only shown for test problems that could not be solved using ILU(0) as a preconditioner. For the Harwell-Boeing and UMFPACK collections, when there is a set of related matrices, only the results for one or two matrices in the set are shown. Some of the test matrices are small. However, the difficulties that they encounter (e.g., zero pivots) are mostly representative of those for larger matrices. For large matrices, we expect unstable triangular solves to be more severe, since there will be longer associated recurrences. We also expect more problems which fail to converge due to insufficient amounts of fill-in to achieve an accurate factorization.

In the Harwell-Boeing collection, the RUA (real, unsymmetric, assembled) matrices were tested with ILU(0) preconditioning. Of the 97 problems, 35 were successfully solved, 56 failed due to zero pivots, and 5 did not converge. There were a large number of failures due to zero pivots because of the large number of very unstructured matrices in the collection.

Besides this breadth of test problems, we also examined in depth a set of test problems in SPARSKIT from the solution of the incompressible Navier-Stokes equations. We generated this test set with the FIDAP fluid dynamics analysis package [18, 19]. The example problems provided by FIDAP were solved using the fully-coupled solution method, and we extracted the first linear systems in the nonlinear iterations. The incompressibility condition gives these matrices zeros on their diagonals. The FIDAP matrices have a symmetric pattern.

Besides poor orderings that give structurally zero pivots, poor orderings can also give singular leading principal submatrices. We have found this to be common in the FIDAP test matrices. Leading principal submatrices that are singular often end with zero diagonal elements (suggesting that the equation that sets the absolute pressure was at the end). Matrices with this ordering cannot be factored exactly, but approximate factorizations are often useful. Nevertheless, singular leading principal blocks run the risk of producing very small or zero pivots, especially when the amount of fill-in is increased. (The direct solver in FIDAP does not perform pivoting, but replaces zero and small pivots with the 'clipping constant,' which has default value 10^{-8}).

The matrices are listed in Table 2, along with a description, their sizes, and their number of nonzero entries. All the matrices were scaled so that their columns have unit two-norms, and then scaled again so that their rows have unit two-norms. The importance of scaling was discussed in Section 2. Scaling also normalizes the statistics presented in Section 1.

In the numerical tests in the following subsections, the iterative method used to solve these problems was right-preconditioned GMRES restarted every 50 steps. When no right-hand side was provided, a vector of all ones was used. The iterations began with a zero initial guess and were stopped when the exact residual norm was reduced by 8 orders of magnitude, or when 500 steps were taken. The latter case is indicated by a dagger (†) in the following Tables.

6.2 Experiments with level-based ILU

We begin by showing how the statistics presented in Table 1 can be used to determine what difficulties are arising when an incomplete factorization fails. Table 3 lists problems that could not be solved using ILU(0) as a preconditioner, along with their values of the statistics, and the causes of failure as classified by Figure 1 and the comments in Section 1. We considered 'condest' to be large when it was larger than 10^{10}, and used this value to classify the failures. (The value of the statistics is 'Inf' when a zero pivot was encountered, and the cause of failure is understood to be a zero pivot.)

Note that 'condest' can be very large; a value of 10^{96} obviously indicates a factorization that is useless. Also, the factorization is always stable unless there is a very small pivot (i.e., max(L+U) is never large unless 1/pivot is large). When max(L+U) is large, it is usually about the same size as 1/pivot. There are no cases where max(L+U) is large but 'condest' is small.

About half of the failures in Table 3 were due to structurally zero pivots (FIDAP024 was the only case of a numerically zero pivot). These mostly correspond to very unstructured matrices, such as problem 'lhr01' whose nonzero pattern is shown in Figure 7. Reordering and partial pivoting will be used to try to remedy the problem of structurally zero pivots. Failures due to small pivots or unstable triangular solves can be avoided to some extent by using pivoting, as discussed in Section 3, or by using a stabilization as discussed in Section 5.

Six failures were classified as due to 'inaccuracy' due to dropping. For these problems, we checked whether or not allowing more fill-in would help solve these problems. The results are shown in Table 4. Only UTM5940 could not be solved with ILU with level as high as 2. The values of the statistics did not increase dramatically, i.e., no other effects seemed to come into play. In the case of FIDAP006, there was one very small pivot (10^{-15}) at the end of the factorization, but all other pivots were greater than 10^{-2} in magnitude. Increasing the fill-in for the other problems (with failures not classified as 'inaccuracy') will not generally help, unless large amounts of fill-in is used.

6.3 Experiments with threshold-based ILU with pivoting

To remedy the problem of structurally zero pivots, we use partial pivoting. Table 5 shows the results using ILUTP. We used a permutation tolerance *permtol* of 1, meaning that whenever an off-diagonal element is larger than the diagonal element, a permutation occurs. Fill-in was controlled using *lfil* set to 30, i.e., 30 nonzeros in each row of L and U was allowed. This relatively large value of *lfil* helps ensure that nonzero pivots can be found.

The results show that there are only two cases where a nonzero pivot could not be found, whereas there were 19 cases of zero pivots with ILU(0). In the 17 cases that pivoting helped, all problems except three could now be solved. This suggests that ILU(0) had failed on problems due to structurally zero pivots, which were otherwise fairly easy to solve.

Pivoting can also to some extent help avoid very small pivots and enhance the stability of the triangular solves. To illustrate this, in Table 6 we show the same experiment with ILUTP as above, but use a smaller *permtol* of 0.01. (We do not use a *permtol* of 0 since this result with no pivoting is extremely poor, i.e., ILUT performs very poorly on this test set without pivoting due to the problems discussed in section 2.2.) There are four more failures, and the results here are poorer. There are three interesting cases: GEMAT11, WIGTO966, and FIDAPM03. These problems failed due to unstable triangular solves with *permtol* of 0.01, but were solved successfully when *permtol* of 1 was used.

There are cases, with both values of *permtol*, where ILUTP encountered a zero pivot while no zero pivots were encountered with ILU(0). Thus it is not rare for ILUTP to produce a poor pivoting sequence.

We emphasize that the matrices were scaled as described in section 2.2. There were many zero pivots and extremely large values of condest when scaling was not used.

6.4 Experiments with ILUTS and reordering

ILUT in bordered or 'skyline' form (ILUTS) was tested on the FIDAP matrices, since these matrices have symmetric structure. However, it is difficult to perform pivoting on matrices stored in bordered form. Thus some sort of preordering must be used instead.

For the FIDAP matrices, there is an obvious reordering that that may give a good factorization. In the original matrices, the unknowns were ordered element by element, with the continuity equations ordered last for each element. A better ordering is to order the continuity equations at the end of all other equations for all elements. This ordering gives a zero block in the lower right-hand corner of the matrix, and we call this a *block* reordering (the matrix is a 2 by 2 block matrix). This ordering ensures that there are no structurally zero pivots.

Table 7 shows the results of ILUTS using this reordering. The fill-in was controlled to be not more than the fill-in for ILU(0). For comparison, we show in Table 8 the results of ILU(0) and ILUT, all with comparable amounts of fill-in. ILUTS was the most reliable preconditioner. When the block reordering is used, none of the preconditioners encountered zero pivots, and all values of 1/pivot are less than 10^8 (not shown). The results without this reordering are very poor for ILUTS; for ILUT, all the failures shown for the original ordering were due to zero pivots.

If we increase the amount of fill-in but do not use this reordering, ILU(1), for example, can only help solve 3 FIDAP problems. However, if the block reordering is used, ILU(1) helps solve *all* the problems (not shown). Also as fill-in is increased, the results of ILUT and ILUTS become very similar (not shown).

6.5 Experiments with stabilized ILU

Stabilization can be an effective option when 'condest' is large, or there are small pivots. We tested the problems in Table 2 with a stabilized version of ILUT. Pivots whose absolute value were smaller than a parameter *thresh* were replaced by *thresh* with the original sign of the pivot. This worked very well for the FIDAP matrices. For the very unstructured matrices, this strategy did not help. Pivoting is a better solution for this latter class of matrices.

Table 9 shows the result for the FIDAP matrices of ILUT with *lfil* parameter 30, and *thresh* set to 0.5 (i.e., $1/\text{pivtol} \leq 2$), a relatively large value. As mentioned, this shift is usually much larger than what is suggested in the literature to plainly avoid very small pivots. The larger shift has the effect of making the problem much better conditioned. Without the shift, we could only solve problems FIDAP006, FIDAPM02, and FIDAPM08.

In Table 10(a) we perform a parameter study of the effect of changing *thresh* for the WIGTO966 problem. Pointwise ILU(0) was used, with GMRES(100) and a tolerance of 10^{-6}. As *thresh* is increased, condest decreases. This was always true in our experiments. However, the best threshold balances the accuracy of the factorization and the stability of the triangular solves.

The WIGTO966 matrix comes from an Euler model of an airfoil with four degrees of freedom at each grid point. Thus we can use block ILU with a block size of 4, and illustrate the use of a block shift (9). Table 10(b) shows the results. Here, *thresh* is the ratio of the largest singular value to the smallest in (9). Our experiments with other problems generally show that when shifting is successful, it does not matter if a pointwise or block shift is used.

6.6 Harder problems

There are several problems in Table 2 for which we have not yet presented a successful solution method. Consider first the matrix 'lhr01.' ILU(0) and ILUTP both encounter zero pivots when trying to approximately factor this matrix. Figure 7 shows its nonzero pattern, and Figure 8 is a close-up of the top-left 100×300 block. For rows 25 to 60, there are not many choices for good pivots, when column pivoting is used. However, there may be many good choices of pivots if row pivoting is used. Thus we used a column-wise ILU algorithm with row pivoting (actually, we only computed the ILU factors of the transposed matrix data structure), and no zero pivots were encountered. Table 11 shows the problem was solved in 134 steps. By not applying these algorithms blindly, for example, by looking at the structure of the matrix in this case, we were able to make ILU work.

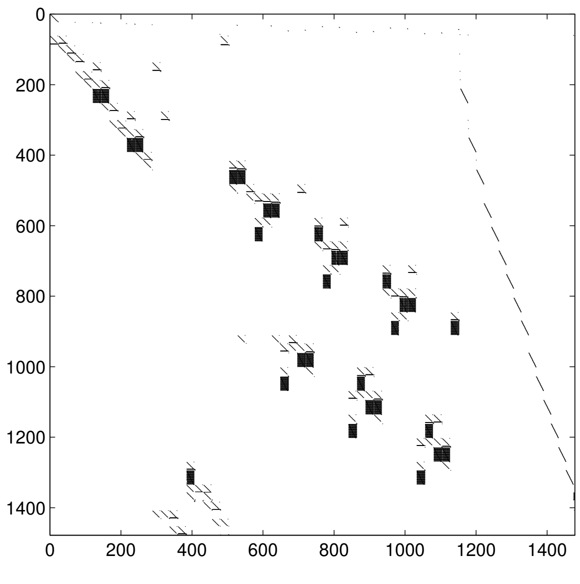


Figure 7: Nonzero pattern for 'lhr01.'

Another problem for which we had difficulty was GRE1107. ILUTP(30) with *permtol* 1.0 suggests that the difficulty is inaccuracy, and we start from there. We increased *lfil* to 50,

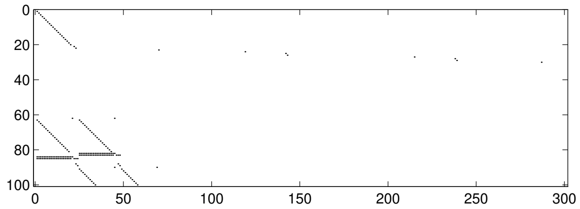


Figure 8: Nonzero pattern for 'lhr01,' top-left 100×300 block.

but the GMRES solver was still stagnating. By looking at the convergence history, Figure 9, convergence is steady until GMRES restarts at step 50. However, GMRES will converge in 94 steps if we do not restart (we tried GMRES(100)). In this case, we were able to make ILU work by being aware that the Krylov subspace basis needed to be larger. (There was no convergence with ILUTP(30) and GMRES(100).)

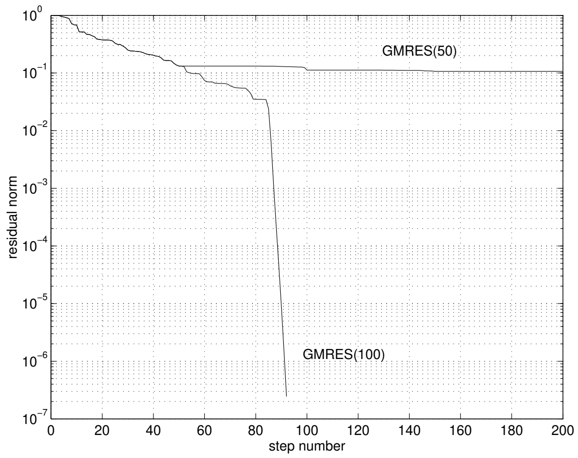


Figure 9: Convergence history for GRE1107.

We also could not solve UTM5940, this time due to inaccuracy in ILU(0) or unstable solves in ILUTP. For this problem, we know that zero pivots are not encountered with the original ordering, so we tried ILUT(30) without pivoting. The failure in this case could be classified as inaccuracy due to dropping, i.e., it was the pivoting that made the solves unstable in this case. Next we tried increasing *lfil* to 50, and the solution was found in 399 steps. The contributor of this matrix, Peter Brown, had found that reordering this matrix with reverse Cuthill-McKee (RCM) ordering makes ILU more effective. The solution, keeping all other parameters the same, was found in 37 steps in this case. In general, reordering has a large effect on the accuracy of ILU preconditioners [14, 15].

Consider now LNS3937. If no pivoting is used, the problem is small pivots. We thresholded the pivots for ILU(30). This decreases condest and helps the residual be reduced further, but there is still no convergence. By increasing amount of fill-in with the thresholding also does not help convergence.

Our experience with NNC1374 is somewhat different. If ILUT with or without pivoting is used, the solves are unstable, probably due to the problems discussed in section 2.2. Only very large values of the *thresh* stabilization parameter can reduce condest a significant amount. Thus we go back to ILU(0), which had failed due to small pivots. We try thresholding the pivots in this case, but this did not help. By increasing the amount of fill-in at the same time, the solves became unstable.

There are several problems, such as the above two, that are very difficult to solve by using ILU preconditioners. A third problem, 'shyy41,' is even difficult to solve with *direct* solvers: the factorization is stable, but the triangular solves are very unstable. This matrix contains an independent diagonal block that is a 5-point matrix with a zero diagonal. Solves with the factors of this block are very unstable.

6.7 Block ILU preconditioners

Many linear systems from engineering applications arise from the discretization of coupled partial differential equations. A blocking in these systems may be imposed by ordering together the equations and unknowns at a single grid point. Experimental tests suggest it is very advantageous for preconditionings to exploit this block structure in a matrix. In *block* incomplete factorizations (BILU), each entry in Algorithm 2.1 is actually a small dense block. Dropping of the blocks can be based on the block level (BILUK) or the Frobenius norm of the block (BILUT).

PULLIAM1 and BBMAT are two matrices with block structure. We will briefly compare BILUK and BILUT, and show the effect of increasing the block size. Figure 10 shows 'condest' for BILUT(lfil) applied to PULLIAM1, with blocksize 8 (*lfil* now refers to the number of blocks in a block row). The shape of this graph is typical: as fill-in is increased, the triangular solves become more unstable, until the factorization approaches that of a direct solve. For low amounts of fill-in, there are not enough nonzeros to make very large values of 'condest.' BILUK(0) corresponds to *lfil* of approximately 4. BILUT is successful in this case for *lfil* approximately 23. Van der Vorst [38] briefly investigated the effect that increasing fill-in has on stability in the nonsymmetric case. His conclusion also was that increasing the accuracy does not seem to help, unless of course, the accuracy approaches that of a direct solve.

Table 12(a) shows 'condest' for BILUK applied to the PULLIAM1 problem. The values are much smaller. This suggests that threshold-based incomplete factorizations are much more prone to unstable triangular solves than level-based ones. An alternative when an threshold-based ILU is unstable is to use a level-based factorization. Table 12(b) shows the number of GMRES steps required to solve the PULLIAM1 problem. Table 13(a) shows the number of GMRES steps required to solve the BBMAT problem, along with timings on a Cray-C90 computer. Note that block size 16 is fastest, even though the factorization requires 50 percent more storage than block size 8 (due to some explicit storage of zeros; see Table 13(b)), partially due to better vectorization. The storage for a direct solver is approximately 36.0 million nonzero entries.

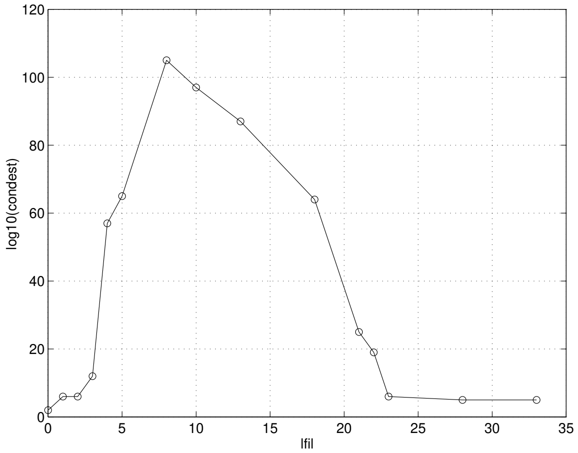


Figure 10: Plot of condest for BILUT(lfil) for the PULLIAM1 problem. The blocksize used was 8.

7 Conclusions

It is clear that the *blind* application of incomplete factorizations will be unsuccessful for many problems. However, by being attentive to the characteristics of a problem and the difficulties it encounters, incomplete factorizations can be made effective. For example, problem 'lhr01' encountered structurally zero pivots even when column pivoting was used. After looking at the structure of the matrix, we were able to solve the problem by using row pivoting. Poor orderings and scalings are other characteristics of matrices that can cause difficulties.

We presented several statistics that can help determine the causes of failure of incomplete factorizations. These statistics measure obvious quantities: the stability of the triangular solves, the smallest pivots, and the stability of the factorization.

The occurrence of zero pivots is very common in very unstructured problems. Partial pivoting is very effective in remedying this problem. Thresholding small and zero pivots is a less effective solution when the matrix is very unstructured. However, large values of the threshold (i.e., of the stabilization, or perturbation to the pivot) has another, more important effect: it makes the L and U factors better conditioned. This is an important effect, even if the problem can be solved with ILU(0).

The most difficult problems to solve were those with unstable triangular solves that were not caused by very small pivots. The $(LU)^{-1}$ condition estimate can be reduced by thresholding the pivots, but very large thresholds are required. This destroys the accuracy of the factorization, and usually, increasing the fill-in is done in vain. For these problems, the last resort seems to be to use very large amounts of fill-in [10], for example, as we did for the PULLIAM1 and BBMAT problems.

General-purpose software for incomplete factorizations should include options for pivoting and perturbing pivots. The latter should be a particularly simple addition to any incomplete factorization code. The difficulty is determining when these options should be used, and the values for their parameters. The statistics introduced in this paper can be used to determine what difficulties are occurring, and to guide the selection of parameters or variants of incomplete factorizations. The hope is that for a class of problems, one can find a strategy or set of parameters that is effective for all problems in that class.

Acknowledgements The authors wish to thank the people who have contributed test matrices, directed us to papers, and made very useful comments during this work, particularly Andrew Chapman, Laura C. Dutto, Vahé Haroutunian, Isaac Hasbani, Michael A. Heroux, Xiaoye S. Li, Thomas H. Pulliam, and Barry Rackner. Computer facilities for this research were provided by both the Minnesota Supercomputer Institute and SGI/Cray Research.

Selected matrices from the Harwell-Boeing collection			
matrix	n	nnz	description
BP0	822	3276	Basis matrix from the simplex method
BP1000	822	4841	Basis matrix from the simplex method
FS7603	760	5976	Chemical kinetics, 38 species
GEMAT11	4929	33185	Optimal power flow problem
GRE1107	1107	5664	Simulation of computer systems
IMPCOLD	425	1339	Chemical engineering model
LNS3937	3937	25407	Compressible Navier-Stokes
NNC1374	1374	8606	Nuclear reactor core model
ORANI678	2529	90158	Economic model of Australia
PSMIGR1	3140	543162	Demography application
SHL400	663	1712	Basis matrix from the simplex method
STR600	363	3279	Basis matrix from the simplex method
WEST0381	381	2157	Chemical engineering plant model
WEST2021	2021	7353	Chemical engineering plant model

Selected matrices from the UMFPACK collection			
matrix	n	nnz	description
goodwin	7320	324784	CFD finite element matrix (Goodwin)
lhr01	1477	18592	Chemical process simulation (Mallya)
radfr1	1048	13299	Chemical process separation (Zitney)
shyy41	4720	20042	Fully-coupled Navier-Stokes (Shyy)
BBMAT	38744	1771722	N-S model of airfoil, ARC2D (Simon)

Selected matrices from the SPARSKIT collection			
matrix	n	nnz	description
PULLIAM1	17028	400896	Euler model of airfoil, ARC2D, $M = 0.8$ (Pulliam)
UTM5940	5940	83842	Tokamak simulation (Brown)
WATSON5	1853	10847	Circuit simulation (Watson)
WIGTO966	3864	238252	Finite volume model of fluid flow (Wigton)
FIDAP006	1651	49533	Die-swell problem
FIDAP014	3251	66775	Isothermal seepage flow
FIDAP024	2283	48737	Unsymmetric forward roll coating
FIDAP032	1159	11343	Radiation heat transfer, open channel
FIDAPM02	537	19241	3-D steady Couette flow
FIDAPM03	2532	50380	Flow past a cylinder in freestream, Re = 40
FIDAPM07	2065	53533	Natural convection in a square enclosure
FIDAPM08	3876	103076	Developing flow, vertical channel
FIDAPM09	4683	95053	Jet impingment cooling, Re = 100
FIDAPM10	3046	53842	2-D flow over multiple steps in a channel
FIDAPM13	3549	71975	Axisymmetric poppet valve
FIDAPM15	9287	98519	Spin up of a liquid in an annulus
FIDAPM33	2353	23765	Radiation heat transfer in a square cavity

Table 2: Test matrices. These problems could not be solved using ILU(0) as a preconditioner.

matrix	max(L+U)	1/pivot	condest	reason for failure
BP0	Inf	Inf	Inf	
BP1000	Inf	Inf	Inf	
FS7603	3.19e+02	9.53e+02	9.07e+03	inaccuracy
GEMAT11	Inf	Inf	Inf	
GRE1107	2.20e+06	2.81e+06	1.85e+96	unstable solve
IMPCOLD	Inf	Inf	Inf	
LNS3937	4.17e+11	6.36e+11	3.82e+13	small pivot
NNC1374	4.58e+08	5.27e+08	2.38e+10	small pivot
ORANI678	Inf	Inf	Inf	
PSMIGR1	Inf	Inf	Inf	
SHL400	Inf	Inf	Inf	
STR600	Inf	Inf	Inf	
WEST0381	Inf	Inf	Inf	
WEST2021	Inf	Inf	Inf	
goodwin	5.82e+05	3.63e+04	1.47e+06	inaccuracy
lhr01	Inf	Inf	Inf	
radfr1	Inf	Inf	Inf	
shyy41	Inf	Inf	Inf	
BBMAT	2.39e+06	2.00e+06	6.32e+52	unstable solve
PULLIAM1	Inf	Inf	Inf	
UTM5940	1.01e+03	2.21e+03	1.68e+04	inaccuracy
WATSON5	1.89e+00	5.63e+14	6.63e+15	small pivot
WIGTO966	3.42e+04	1.20e+04	2.11e+12	unstable solve
FIDAP006	1.46e+01	1.61e+01	4.91e+04	inaccuracy
FIDAP014	4.02e+03	9.98e+03	2.26e+20	unstable solve
FIDAP024	Inf	Inf	Inf	(numerically zero pivot)
FIDAP032	Inf	Inf	Inf	
FIDAPM02	1.36e+02	3.80e+02	2.34e+04	inaccuracy
FIDAPM03	Inf	Inf	Inf	
FIDAPM07	2.81e+03	7.29e+03	5.60e+13	unstable solve
FIDAPM08	1.52e+01	2.81e+01	1.03e+03	inaccuracy
FIDAPM09	1.21e+05	2.91e+05	1.38e+22	unstable solve
FIDAPM10	4.68e+27	7.05e+27	2.84e+31	small pivot
FIDAPM13	2.42e+27	3.47e+27	2.96e+27	small pivot
FIDAPM15	Inf	Inf	Inf	
FIDAPM33	Inf	Inf	Inf	

Table 3: Problems that could not be solved with ILU(0), and corresponding statistics and possible reasons for failure.

matrix	method	max(L+U)	1/pivot	condest	steps
FS7603	ILU(1)	2.36e+03	1.56e+03	1.59e+06	84
goodwin	ILU(2)	1.26e+05	9.63e+04	2.91e+06	417
UTM5940	ILU(2)	3.63e+02	7.12e+02	8.61e+04	†
FIDAP006	ILU(1)	2.22e+01	2.81e+14	9.52e+14	49
FIDAPM02	ILU(1)	3.85e+01	1.04e+03	5.42e+02	19
FIDAPM08	ILU(1)	1.91e+01	1.67e+01	3.99e+02	178

Table 4: Increasing the level-of-fill for problems classified as failed due to inaccuracy from dropping. The Table shows the statistics and the number of GMRES steps for convergence.

matrix	max(L+U)	1/pivot	condest	steps	reason for failure
BP0	1.45e+02	2.44e+02	1.13e+04	3	
BP1000	2.26e+01	1.30e+03	5.31e+03	13	
FS7603	7.89e+02	8.63e+02	1.07e+10	†	inaccuracy
GEMAT11	4.99e+02	1.09e+03	8.20e+04	25	
GRE1107	9.63e+00	2.97e+01	1.18e+04	†	inaccuracy
IMPCOLD	2.25e+00	7.81e+00	3.16e+02	2	
LNS3937	1.72e+09	1.14e+09	2.55e+19	†	unstable solve
NNC1374	1.49e+09	1.67e+10	5.19e+172	†	unstable solve
ORANI678	6.37e+00	1.70e+01	7.66e+01	8	
PSMIGR1	7.58e+00	2.81e+01	3.98e+03	9	
SHL400	3.28e+01	2.56e+03	5.83e+05	3	
STR600	3.60e+01	4.95e+01	4.96e+03	4	
WEST0381	2.26e+01	3.41e+01	2.04e+02	11	
WEST2021	1.27e+05	2.19e+05	1.17e+07	8	
goodwin	1.55e+02	2.09e+07	3.37e+70	†	unstable solve
lhr01	Inf	Inf	Inf	†	zero pivot
radfr1	1.83e+01	1.90e+01	2.29e+04	25	
shyy41	2.07e+01	1.06e+37	1.35e+37	†	small pivot
BBMAT	6.25e+16	1.49e+09	1.83e+177	†	unstable solve
PULLIAM1	1.02e+03	2.26e+16	1.16e+209	†	unstable solve
UTM5940	7.63e+01	1.80e+05	3.81e+32	†	unstable solve
WATSON5	6.98e+02	1.69e+05	7.21e+04	8	
WIGTO966	4.84e+00	3.91e+00	9.98e+03	247	
FIDAP006	8.27e+00	3.70e+02	8.83e+03	26	
FIDAP014	9.10e+02	2.77e+04	2.05e+64	†	unstable solve
FIDAP024	4.29e+00	4.84e+00	1.06e+03	20	
FIDAP032	2.27e+00	7.43e+01	3.21e+03	6	
FIDAPM02	3.87e+01	5.34e+01	6.04e+04	141	
FIDAPM03	8.21e+00	8.97e+00	4.32e+02	30	
FIDAPM07	1.14e+03	5.76e+04	3.24e+28	†	unstable solve
FIDAPM08	3.27e+00	1.58e+01	8.39e+05	24	
FIDAPM09	Inf	Inf	Inf	†	zero pivot
FIDAPM10	4.23e+00	1.78e+02	3.63e+04	25	
FIDAPM13	9.87e+01	8.22e+02	3.37e+06	†	inaccuracy
FIDAPM15	4.93e+00	9.53e+00	1.06e+07	60	
FIDAPM33	8.83e+00	1.22e+01	1.35e+04	4	

Table 5: Results for ILUTP (*lfil*=30, *permtol*=1.00) for problems that failed with ILU(0). The statistics are shown along with the number of GMRES steps required for convergence, or the possible reason for failure.

matrix	max(L+U)	1/pivot	condest	steps	reason for failure
BP0	1.45e+02	2.44e+02	1.13e+04	3	
BP1000	6.12e+02	1.40e+03	2.29e+05	32	
FS7603	1.76e+03	1.87e+03	2.32e+07	195	
GEMAT11	9.15e+03	2.85e+04	7.49e+14	†	unstable solve
GRE1107	1.17e+03	1.88e+06	1.78e+34	†	unstable solve
IMPCOLD	5.92e+02	1.08e+02	4.53e+04	9	
LNS3937	Inf	Inf	Inf	†	zero pivot
NNC1374	5.71e+10	9.59e+09	5.75e+250	†	unstable solve
ORANI678	Inf	Inf	Inf	†	zero pivot
PSMIGR1	6.11e+00	4.71e+02	3.95e+03	9	
SHL400	3.28e+01	2.56e+03	5.83e+05	3	
STR600	2.54e+02	9.99e+01	5.11e+03	5	
WEST0381	2.55e+03	2.49e+02	2.48e+05	32	
WEST2021	1.93e+07	1.93e+07	3.86e+06	15	
goodwin	1.64e+04	1.47e+04	3.88e+42	†	unstable solve
lhr01	Inf	Inf	Inf	†	zero pivot
radfr1	9.10e+04	2.56e+02	2.44e+06	80	
shyy41	9.98e+01	3.55e+36	4.94e+42	†	small pivot
BBMAT	7.60e+19	2.60e+10	1.37e+474	†	unstable solve
PULLIAM1	6.39e+05	1.04e+12	3.80e+374	†	unstable solve
UTM5940	4.14e+02	1.87e+03	2.02e+07	†	inaccuracy
WATSON5	1.38e+03	5.65e+04	3.32e+04	8	
WIGTO966	2.25e+03	2.58e+02	2.23e+12	†	unstable solve
FIDAP006	1.26e+02	5.11e+13	1.69e+14	30	
FIDAP014	Inf	Inf	Inf	†	zero pivot
FIDAP024	8.97e+01	1.73e+02	4.34e+05	98	
FIDAP032	2.72e+00	1.18e+01	3.21e+03	6	
FIDAPM02	3.85e+01	1.04e+03	2.97e+02	13	
FIDAPM03	1.83e+02	1.52e+02	4.82e+17	†	unstable solve
FIDAPM07	4.49e+03	2.99e+05	5.14e+35	†	unstable solve
FIDAPM08	2.84e+01	1.62e+01	2.27e+06	28	
FIDAPM09	1.04e+03	1.27e+22	1.51e+314	†	unstable solve
FIDAPM10	2.29e+01	4.27e+01	1.82e+04	17	
FIDAPM13	1.39e+02	4.09e+02	2.81e+25	†	unstable solve
FIDAPM15	Inf	Inf	Inf	†	zero pivot
FIDAPM33	6.17e+02	1.33e+02	3.73e+18	3	

Table 6: Results for ILUTP (*lfil*=30, *permtol*=0.01) for problems that failed with ILU(0). The statistics are shown along with the number of GMRES steps required for convergence, or the possible reason for failure. The results are slightly worse than when *permtol*=1.00.

matrix	max(L+U)	1/pivot	condest	steps
FIDAP006	4.19e+01	6.31e+01	0.56e+04	251
FIDAP014	2.40e+03	2.27e+06	0.30e+09	†
FIDAP024	4.25e+00	9.74e+00	0.17e+03	168
FIDAP032	2.95e+00	6.76e+00	0.47e+02	†
FIDAPM02	1.10e+01	5.32e+02	0.93e+03	102
FIDAPM03	1.79e+01	2.47e+01	0.47e+03	57
FIDAPM07	2.34e+04	7.17e+05	0.16e+06	†
FIDAPM08	4.94e+00	9.43e+00	0.95e+03	262
FIDAPM09	2.48e+02	5.99e+02	0.53e+04	†
FIDAPM10	1.08e+01	1.57e+01	0.33e+03	140
FIDAPM13	3.76e+02	5.63e+02	0.29e+05	79
FIDAPM15	1.34e+01	3.25e+01	0.34e+05	†
FIDAPM33	9.46e+00	2.50e+01	0.83e+03	24

Table 7: ILUT in bordered form with fill-in comparable to ILU(0). The matrices were reordered with continuity equations last (block reordering).

matrix	ILU(0) reord	ILUT orig	ILUT reord	ILUTS reord
FIDAP006	†	345	†	251
FIDAP014	†	†	†	†
FIDAP024	†	†	†	168
FIDAP032	245	†	274	†
FIDAPM02	219	49	†	102
FIDAPM03	119	†	153	57
FIDAPM07	†	†	†	†
FIDAPM08	†	189	†	262
FIDAPM09	†	†	†	†
FIDAPM10	236	†	225	140
FIDAPM13	150	†	448	79
FIDAPM15	†	†	†	†
FIDAPM33	25	†	38	24
total successes	6	3	5	8

Table 8: Number of steps for convergence with the use of various preconditionings, all with comparable fill-in; original ordering (orig) and block reordering (reord).

matrix	max(L+U)	condest	steps
FIDAP006	2.30e+00	4.47e+02	46
FIDAP014	1.15e+00	1.74e+01	†
FIDAP024	2.85e+00	1.14e+02	33
FIDAP032	1.98e+00	1.98e+02	30
FIDAPM02	1.02e+00	5.60e+01	85
FIDAPM03	1.06e+01	2.35e+02	88
FIDAPM07	1.50e+00	2.12e+02	474
FIDAPM08	2.96e+00	3.88e+02	247
FIDAPM09	1.47e+00	9.26e+17	†
FIDAPM10	1.52e+01	1.14e+03	50
FIDAPM13	2.36e+00	4.57e+02	486
FIDAPM15	9.06e+00	6.81e+02	†
FIDAPM33	1.35e+01	5.20e+03	24

Table 9: Results for stabilized ILUT, *lfil* = 30, *thresh* = 0.5

thresh	condest	steps
0.	2.19e+17	†
0.001	4.63e+17	†
0.002	1.75e+09	†
0.003	1.04e+06	73
0.004	9.84e+03	90
0.005	7.42e+03	84
0.01	1.85e+03	90
0.1	2.64e+02	†

(a) Pointwise ILU(0)

thresh	condest	steps
0.	1.51e+08	†
0.001	7.22e+09	†
0.01	5.30e+05	72
0.1	7.24e+04	43
0.5	1.16e+03	177
1.0	4.25e+02	†

(b) Block ILU(0)

Table 10: Stablized ILU(0), (a) pointwise, and (b) block versions, for the WIGTO966 problem.

matrix	method	max(L+U)	1/pivot	condest	steps
lhr01(t)	ILUTP(30)	4.58e+03	9.68e+09	3.44e+11	134
GRE1107(f)	ILUT(50)	1.57e+01	3.40e+01	1.43e+04	94
UTM5940	ILUT(50)	2.23e+03	7.20e+02	5.24e+06	399
UTM5940(r)	ILUT(50)	1.37e+02	7.06e+02	5.56e+06	37

Table 11: Solution of some harder problems. Notes: (t) transposed data structure and ILUTP with *permtol*=1; (f) full GMRES was used; (r) reordered with RCM.

block size	BILUK level 0	1	2
4	9.74e+16	4.89e+11	4.94e+13
8	2.64e+10	4.25e+09	1.03e+09
16	4.51e+09	2.18e+09	1.90e+09

(a) condest

block size	BILUK level 0	1	2
4	†	†	†
8	148	61	42
16	147	39	40

(b) GMRES steps

Table 12: BILUK preconditioning for the PULLIAM1 problem. GMRES(100) was used to reduce the residual norm by 10^{-6}.

block size	GMRES steps	CPU time (s) precon	solve	total
4	†	291.1	†	†
8	56	61.1	162.6	223.8
16	51	28.1	70.9	99.0

(a) Timings on one processor of a Cray-C90

block size	BILUK level 0	1	2
1	1.8	5.7	11.4
4	3.3	8.5	13.0
8	4.2	9.7	14.2
16	7.6	14.2	21.9

(b) Number of scalar nonzeros for BILUK, in millions

Table 13: BILUK preconditioning for the BBMAT problem.

References

[1] O. Axelsson and L. Yu. Kolotilina. Diagonally compensated reduction and related preconditioning methods. *Num. Lin. Alg. Appl.*, 1, 1995.

[2] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Num. Math.*, 48:479–498, 1986.

[3] O. Axelsson and G. Lindskog. On the rate of convergence of the preconditioned conjugate gradient method. *Num. Math.*, 48:499–523, 1986.

[4] Owe Axelsson. *Iterative Solution Methods.* Cambridge University Press, Cambridge, 1994.

[5] E. F. F. Botta, A. van der Ploeg, and F. W. Wubs. Nested grids ilu-decomposition (ngilu). *J. Comp. Appl. Math.*, 66:515–526, 1996.

[6] A. M. Bruaset, A. Tveito, and R. Winther. On the stability of relaxed incomplete LU factorizations. *Math. Comp.*, 54:701–719, 1990.

[7] N. I. Buleev. A numerical method for the solution of two-dimensional and three-dimensional equations of diffusion. *Math. Sb.*, 51:227–238, 1960. English transl.: Rep. BNL-TR-551, Brookhaven National Laboratory, Upton, New York, 1973.

[8] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comp.*, 31:162–179, 1977.

[9] T. F. Chan and H. A. Van der Vorst. Approximate and incomplete factorizations. Technical Report 871, Department of Mathematics, University of Utrecht, 1994.

[10] A. Chapman, Y. Saad, and L. Wigton. High-order ILU preconditioners for CFD problems. Technical Report UMSI 96/14, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, Minnesota, 1996.

[11] E. Chow and Y. Saad. ILUS: an incomplete LU factorization for matrices in sparse skyline format. *Intl. J. Num. Meth. Fluids*, to appear.

[12] E. F. D'Azevdo, P. A. Forsyth, and W.-P. Tang. Towards a cost-effective ILU preconditioner with high level fill. *BIT*, 32:442–463, 1992.

[13] I. S. Duff, N. I. M. Gould, J. K. Reid, and J. A. Scott. The factorization of sparse indefinite matrices. *IMA J. Num. Anal.*, 11:181–204, 1991.

[14] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.

[15] L. C. Dutto. The effect of ordering on preconditioned GMRES algorithms, for solving the compressible Navier-Stokes equations. *Int. J. Numer. Methods Engrg.*, 36:457–497, 1993.

[16] H. C. Elman. A stability analysis of incomplete LU factorizations. *Math. Comp.*, 47:191–217, 1986.

[17] H. C. Elman. Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems. *BIT*, 29:890–915, 1989.

[18] M. Engelman. *FIDAP: Examples Manual, Revision 6.0.* Fluid Dynamics International, Evanston, IL, 1991.

[19] M. S. Engelman and I. Hasbani. Matrix-free solution algorithms in a finite element context. Technical Report 88-1, Fluid Dynamics International, Evanston, Illinois, 1988.

[20] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall, Englewood Cliffs, NJ, 1981.

[21] I. Gustafsson. A class of first-order factorization methods. *BIT*, 18:142–156, 1978.

[22] A. Jennings and G. M. Malik. Partial elimination. *J. Inst. Maths Applics*, 20:307–316, 1977.

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

[24] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34:473–497, 1980.

[25] J. A. Meijerink and H. A. Van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31(137):148–162, 1977.

[26] J. A. Meijerink and H. A. Van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J. Computational Physics*, 44(1):134–155, 1981.

[27] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients. *ACM Trans. Math. Softw.*, 6:206–219, 1980.

[28] T. A. Oliphant. An implicit numerical method for solving two-dimensional time-dependent diffusion problems. *Quart. Appl. Math.*, 19:221–229, 1961.

[29] T. A. Oliphant. An extrapolation process for solving linear systems. *Quart. Appl. Math.*, 20:257–267, 1962.

[30] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[31] S. V. Parter. The use of linear graphs in Gauss elimination. *SIAM Rev.*, 3:119–130, 1961.

[32] Y. Robert. Regular incomplete factorizations of real positive definite matrices. *Lin. Alg. Appl.*, 48:105–117, 1982.

[33] Y. Saad. Preconditioning techniques for indefinite and nonsymmetric linear systems. *J. Comp. Appl. Math.*, 24:89–105, 1988.

[34] Y. Saad. ILUT: A dual threshold incomplete ILU factorization. *Num. Lin. Alg. Appl.*, 1:387–402, 1994.

[35] Y. Saad. Preconditioned Krylov subspace methods for CFD applications. In W. G. Habashi, editor, *Proceedings of the International Workshop on Solution Techniques for Large-Scale CFD Problems*, pages 179–195, Montréal, Québec, 1994.

[36] O. Østerby and Z. Zlatev. *Direct Methods for Sparse Matrices*, volume 157 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1983.

[37] H. A. Van der Vorst. Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems. *J. Computational Physics*, 44(1):1–19, 1981.

[38] H. A. Van der Vorst. Stabilized incomplete LU-decompositions as preconditionings for the Tchebycheff iteration. In D. J. Evans, editor, *Preconditioning methods: Analysis and Applications*, New York, 1983. Gordon and Breach.

[39] R. S. Varga. Factorization and normalized iterative methods. In R. E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121–142, Santa Barbara, California, 1960. University of Wisconsin Press.

[40] R. S. Varga, E. B. Saff, and V. Mehrman. Incomplete factorizations of matrices and connections with h-matrices. *SIAM J. Numer. Anal.*, 17:787–793, 1980.

[41] J. W. Watts-III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Soc. Pet. Eng. J.*, 21:345–353, 1981.

[42] G. Wittum. On the robustness of ILU-smoothing. *SIAM J. Sci. Stat. Comput.*, 10:699–717, 1989.

[43] A. Yu. Yeremin. Private communication, 1995.

[44] Z. Zlatev, V. A. Barker, and P. G. Thomsen. SLEST: A Fortran IV subroutine for solving sparse systems of linear equations. User's guide. Technical Report NI-78-01, Numerisk Institut, Lyngby, Denmark, 1978.