A New Preconditioning Technique for Solving Large Sparse Linear Systems

M. Tismenetsky
IBM Scientific Research Center
Haifa 32000, Israel

Dedicated to Gene Golub, Richard Varga, and David Young

Submitted by Wilhelm Niethammer

ABSTRACT

A new sparse approximate triangular factorization technique for solving large sparse linear systems by iterative methods is proposed. The method is based on the description of the triangular factorization of a matrix as a product of elementary matrices and provides a general scheme for constructing incomplete preconditioners for the given matrix. In particular, the familiar incomplete Choleski decomposition can be incorporated into this scheme. The algorithm, based on choice by value, compares favorably with the incomplete Choleski preconditioner and, equipped with a user-controlled parameter, is able to tackle extremely ill-conditioned problems arising in structural analysis, semiconductor simulation, oil-reservoir modelling, and other applications. When applied to a positive definite symmetric matrix, the algorithm produces a preconditioning matrix preserving that property.

0. INTRODUCTION

Iterative solution of sparse linear systems

$$Au = f$$

is attractive, since full advantage of the matrix sparsity can be taken. To have any chance of beating the direct methods, the iterative methods must converge rapidly, and this naturally leads to the search for good preconditioners for A. The latter refers to finding a nonsingular matrix $\tilde{A}$ which is easily invertible and such that the matrix $\tilde{A}^{-1}A$ has improved condition.

© Elsevier Science Publishing Co., Inc., 1991
655 Avenue of the Americas, New York, NY 10010 0024-3795/91/$3.50

A powerful and popular class of preconditioners is based on an incomplete LDU factorization of the matrix. The factorization is incomplete in the sense that some elements of the triangular factors are thrown away during the elimination process. Among a wide variety of such preconditioners developed and tested [1–12, 14, 15, 17, 18], one can discern two main approaches in choosing the elements to be kept (see [2]):

(1) Incomplete factorization by position.
(2) Incomplete factorization by value.

This paper presents a new way to obtain an incomplete factorization of a sparse matrix for the purpose of creating a preconditioner. The factorization is constructed in the form LDU, where the matrices L and U are lower and upper triangular, respectively, and are considered as products of elementary matrices L_j and U_j:

$$L = L_1 L_2 \cdots L_n, \qquad U = U_n U_{n-1} \cdots U_1.$$

The factorization is then viewed as the process of sequentially choosing L_j and U_j in the recursion

$$A_0 = A, \qquad A_j = L_j^{-1} A_{j-1} U_j^{-1}.$$

The diagonal of A_n is taken as D, while its off-diagonal elements define the error matrix

$$A - LDU = L(A_n - D)U. \tag{0.1}$$

This framework allows a variety of strategies in choosing L and U. In particular, all previously known incomplete LDU preconditioners can be incorporated into this scheme.

The preconditioning algorithm presented in this work rests on the idea of keeping a few largest elements during the elimination. Several ways of discarding small elements are proposed, all different from the method described in [1, 17].

In the known incomplete triangular factorizations the elements of the active row (or column) discarded do not participate further in the elimination process. The algorithm proposed makes use of these elements in a way which, in particular, guarantees preservation of positive definiteness of the original matrix.

Another idea exploited in the algorithm is closely related to the minimal-degree ordering (see [16] and references therein). To reduce fill-in in the resulting matrix, the rows and columns of the active (symmetric) submatrix are permuted in a way which assures the choice of the pivot row (or column) with maximal sparsity. When the (incomplete) elimination is performed in the way described in this work and the choice of elements is based on the size of the discarded fill-in rather than the position, this ordering results in a significant decrease in complexity without influencing much the condition number of the preconditioned system. Note that for the usual incomplete Choleski decomposition, just the opposite has been experimentally shown in [5].

Results of some experiments with extremely ill-conditioned matrices arising in several fields of application show attractive performance of the proposed method as compared with the existing ones.

I. INCOMPLETE *LDU* DECOMPOSITION: A MATRIX APPROACH

We start with a detailed matrix form exposition of the *complete* triangular factorization of an $n \times n$ matrix A.

Provided the leading principal minors of A are nonsingular, there exists (e.g., [13, p. 62]) a decomposition

$$A = LDU, \tag{1.1}$$

where L (respectively, U) is a lower (respectively, upper) triangular matrix with ones on its main diagonal, and D stands for the diagonal matrix

$$D = \operatorname{diag}[d_1, d_2, \dots, d_n]$$

Let L_j (respectively, U_j) denote the matrix obtained from L (respectively, U) by replacing all its off-diagonal elements, except those in the jth column (respectively, jth row), with zeros. Obviously,

$$L = L_1 L_2 \cdots L_n, \qquad U = U_n U_{n-1} \cdots U_1,$$

and therefore the computation of the LDU decomposition (1.1) can be viewed as the recursion

$$A_0 = A, \qquad A_j = L_j^{-1} A_{j-1} U_j^{-1} \quad (j = 1, 2, \dots, n), \tag{1.2}$$

giving the diagonal matrix D at the nth step.

Now observe that the inverses of the matrices

$$L_j = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & L_{j+1,j} & \cdots & 0 & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & \cdot & \cdots & 1 & 0 \\ 0 & 0 & \cdots & L_{n,j} & \cdots & 0 & 1 \end{bmatrix},$$

$$U_j = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & 1 & U_{j,j+1} & \cdots & U_{j,n} \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}$$

appearing in (1.2) are

$$L_j^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & \cdot & \cdots & \cdot & \cdot \\ \hdotsfor{7} \\ 0 & 0 & \cdots & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & -L_{j+1,j} & \cdots & 0 & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & \cdot & \cdots & 1 & 0 \\ 0 & 0 & \cdots & -L_{n,j} & \cdots & 0 & 1 \end{bmatrix},$$

$$U_j^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & 1 & -U_{j,j+1} & \cdots & -U_{j,n} \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \hdotsfor{7} \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Representing the latter in the form

$$L_j^{-1} = I - \hat{L}_j, \qquad U_j^{-1} = I - \hat{U}_j,$$

where I stands for the $n \times n$ identity matrix, and substituting into the recursion (1.2), we have

$$A_j = A_{j-1} - \hat{L}_j A_{j-1} - A_{j-1}\hat{U}_j + \hat{L}_j A_{j-1}\hat{U}_j. \tag{1.3}$$

Let $\mathbf{l}_j$ ($\mathbf{u}_j^T$) denote the jth column (respectively, row) of the matrix $\hat{L}_j$ ($\hat{U}_j$). Furthermore, for an arbitrary matrix B, let $B[j;]$ ($B[;j]$) stand for the jth row (respectively, column) of B. Since

$$\hat{L}_j A_{j-1} = \hat{L}[;j]A_{j-1}[j;] = \mathbf{l}_j A_{j-1}[j;],$$

$$A_{j-1}\hat{U}_j = A_{j-1}[;j]\hat{U}[j;] = A_{j-1}[;j]\mathbf{u}_j^T,$$

the equation (1.3) can be rewritten as follows:

$$A_j = A_{j-1} - \mathbf{l}_j A_{j-1}[j;] - A_{j-1}[;j]\mathbf{u}_j^T + A_{j-1}[j;j]\mathbf{l}_j\mathbf{u}_j^T, \tag{1.4}$$

where $A_{j-1}[j;j]$ stands for the element of A_{j-1} in position (i, j).

To find the relationship between $\mathbf{l}_j$, $\mathbf{u}_j^T$, and A_{j-1}, denote by the $\mathbf{e}_j$ the jth unit vector and write

$$\begin{aligned} A_{j-1}[;j] &= A_{j-1}\mathbf{e}_j = L_j L_{j+1}, \dots, L_n D U_n \cdots U_j \mathbf{e}_j \\ &= L_j L_{j+1} \cdots L_n D \mathbf{e}_j = d_j L_j \mathbf{e}_j = d_j L_j[;j]; \end{aligned}$$

$$\begin{aligned} A_{j-1}[j;] &= \mathbf{e}_j^T A_{j-1} = \mathbf{e}_j^T L_j L_{j+1}, \dots, L_n D U_n \cdots U_j \\ &= \mathbf{e}_j^T D U_n U_{n-1} \cdots U_j = d_j \mathbf{e}_j^T U_j = d_j U_j[j;]. \end{aligned}$$

Since the diagonal elements of L_j and U_j equal 1, it thus follows that $d_j = A_{j-1}[j;j]$ and

$$\mathbf{e}_j + \mathbf{l}_j = \frac{1}{d_j} A_{j-1}[;j], \qquad \mathbf{e}_j^T + \mathbf{u}_j^T = \frac{1}{d_j} A_{j-1}[j;]. \tag{1.5}$$

Substitution of the above expressions into (1.4) leads to the familiar recursion

$$A_0 = A, \qquad A_j = A_{j-1} - \frac{1}{A_{j-1}[j;j]} A_{j-1}[;j] A_{j-1}[j;] + A_{j-1}[j;j]\mathbf{e}_j\mathbf{e}_j^T$$

$$(j = 1,2,\dots,n),$$

which results in the complete *LDU* factorization of the matrix A.

Incomplete factorizations of the matrix A, proposed in this work, also rely (as in the complete case) on the recursion (1.2) or, what is equivalent, (1.4). However, to reduce fill-in, the vectors $\mathbf{l}_j$ and $\mathbf{u}_j^T$ in (1.4) are now replaced by the vectors [compare with (1.5)]

$$\mathbf{l}_j = \frac{1}{d_j}\mathbf{m}_j, \qquad \mathbf{u}_j^T = \frac{1}{d_j}\mathbf{v}_j^T, \tag{1.6}$$

respectively, where $\mathbf{m}_j$ and $\mathbf{v}_j^T$ are obtained from the corresponding vectors $A_{j-1}[;j]$ and $A_{j-1}[j;]$ by deleting the elements in positions $i < j$ (during an incomplete factorization process the elimination is not fully accomplished, and hence these elements are generally not zero any more) as well as some off-diagonal elements in positions $i > j$.

In more detail, let the vectors $\mathbf{f}_j$, $\mathbf{g}_j^T$ consist of the discarded elements in the jth column and row of A_{j-1}, respectively. That is,

$$A_{j-1}[;j] = \mathbf{m}_j + \mathbf{f}_j, \qquad A_{j-1}[j;] = \mathbf{v}_j^T + \mathbf{g}_j^T. \tag{1.7}$$

Substituting (1.6) and (1.7) in (1.4), we obtain the following *working procedure*:

$$A_j = A_{j-1} - \frac{1}{d_j}\mathbf{m}_j\mathbf{v}_j^T - \frac{1}{d_j}\left(\mathbf{m}_j\mathbf{g}_j^T + \mathbf{f}_j\mathbf{v}_j^T\right) \qquad \left(d_j = A_{j-1}[j;j]\right). \tag{1.8}$$

The vectors $\mathbf{m}_j$ and $\mathbf{v}_j^T$ are set in the jth column and row of the triangular factors L and U, respectively; d_j is assigned to the jth diagonal element of the matrix D, thus producing an incomplete factorization *LDU* of the given matrix A.

The incomplete factorization described above possesses the following important property:

PROPOSITION. *Let the matrix A be real symmetric or hermitian. Then for any choice of $\mathbf{m}_j$ and $\mathbf{v}_j^T$ (obtaining by deleting some off-diagonal elements of* *the corresponding row and column of the active submatrix), the inertia of the resulting matrix LDU equals the inertia of A.*

In particular, if A is positive definite, so is LDU.

Proof. The recursion in (1.8) can be rewritten in the form (1.4) and subsequently as

$$A_0 = A, \qquad A_j = \underline{L}_j^{-1} A_{j-1} \underline{U}_j^{-1} \quad (j = 1, 2, \ldots, n),$$

where $\underline{L}_j = I + l_j$, $\underline{U}_j = I + u_j$, and the matrices l_j (u_j) has only one—namely, the jth—nonzero column (respectively, row) defined by (1.6).

For a real symmetric matrix A, we set $\underline{U}_j = \underline{L}_j^T$ to obtain

$$A_0 = A, \qquad A_j = \underline{L}_j^{-1} A_{j-1} \left(\underline{L}_j^{-1}\right)^T \quad (j = 1, 2, \ldots, n).$$

In the hermitian case the transposition is accompanied by complex conjugation. The statement of the proposition now follows on applying the well-known inertia theorem (see e.g. [13]). ∎

Note that in the previously reported incomplete factorization schemes, as in the usual Choleski incomplete decomposition, the last summand in (1.8) does not appear. This fact eliminates the possibility of viewing the decomposition process as a recursion of the above form, and consequently the inertia of the resulting matrix LDU may differ from that of A.

Consider the jth stage of the algorithm more closely. In practice there is no need to keep the first j rows and columns of the matrix A_j appearing at the jth stage of the algorithm, and they may be thus deleted.

Hence, we confine our attention to the active $(n-j+1)\times(n-j+1)$ submatrix $\tilde{A}_{j-1}$ of A_{j-1} lying in its right lower corner. This matrix is derived at the $j-1$th stage of the algorithm. In view of (1.8), the similarly defined $(n-j)\times(n-j)$ submatrix $\tilde{A}_j$ of A_j is then obtained in the following way:

$$\begin{bmatrix} d_j & * & * & * \\ * & \cdot & \cdot & \cdot \\ * & \cdot & \tilde{A}_j & \cdot \\ * & \cdot & \cdot & \cdot \end{bmatrix} = \tilde{A}_{j-1} - \frac{1}{d_j}\left(\tilde{\mathbf{m}}_j \tilde{\mathbf{v}}_j^T + \tilde{\mathbf{f}}_j \tilde{\mathbf{v}}_j^T + \tilde{\mathbf{m}}_j \tilde{\mathbf{g}}_j^T\right), \qquad (1.9)$$

where the vector $\tilde{\mathbf{a}}$ stands for the restriction of the vector $\mathbf{a}$ of order n, obtained by deleting its first $j-1$ elements.

Note that, during the familiar incomplete Choleski factorization process, the matrix $\tilde{A}_j$ is obtained by deleting the first column and row of the matrix

$$\tilde{A}_{j-1} - \frac{1}{d_j}\tilde{\mathbf{m}}_j\tilde{\mathbf{v}}_j^T,$$

where the choice of $\tilde{\mathbf{m}}_j$ and $\tilde{\mathbf{v}}_j$ is made by position. Consequently, the elements of $\tilde{\mathbf{f}}_j$ and $\tilde{\mathbf{g}}_j$ do not participate in constructing the matrix $\tilde{A}_j$ and hence play no role at later stages of the algorithm.

The possible fill-in at the jth stage of the algorithm is schematically shown below. The vectors $\tilde{\mathbf{m}}_j$ and $\tilde{\mathbf{v}}_j$ consist of largest off-diagonal elements chosen in the active column and row, respectively. The diagonal element d_j is shown separately. The vectors $\tilde{\mathbf{f}}_j$ and $\tilde{\mathbf{g}}_j^T$ contained the remaining nonzero elements of the corresponding column or row, which do not contribute to the jth column of L but participate in constructing $\tilde{A}_j$ [see (1.9)]. For convenience, it is assumed that the elements chosen are the first elements of the corresponding column and row:

$$\begin{bmatrix} d_j & \cdots & \tilde{\mathbf{v}}_j^T & \cdots & \tilde{\mathbf{g}}_j^T & \cdots \\ \vdots & & \vdots & & \vdots & \\ \tilde{\mathbf{m}}_j & \cdots & \mathrm{I} & \cdots & \mathrm{III} & \cdots \\ \vdots & & \vdots & & \cdot & \\ \tilde{\mathbf{f}}_j & \cdots & \mathrm{II} & \cdot & \cdot & \cdot \\ \vdots & & \vdots & & \cdot & \end{bmatrix}.$$

As easily seen fill-in at this stage of the algorithm occurs in areas I–III and is caused by the Kronecker products

$$\tilde{\mathbf{m}}_j\tilde{\mathbf{v}}_j^T, \qquad \tilde{\mathbf{f}}_j\tilde{\mathbf{v}}_j^T, \qquad \tilde{\mathbf{m}}_j\tilde{\mathbf{g}}_j^T, \tag{1.10}$$

respectively, while during the incomplete Choleski process it may appear only in area *I*.

Provided the number of elements chosen equals the number of elements on the support of the original matrix, a reduction to that level of fill-in may be achieved by updating the nonzero elements of $\tilde{A}_{j-1}$ in areas II and III and discarding the elements of $\tilde{\mathbf{f}}_j\tilde{\mathbf{v}}_j^T$ and $\tilde{\mathbf{m}}_j\tilde{\mathbf{g}}_j^T$ causing fill-in. Clearly, such a relaxation of the process in the case of a positive definite matrix A should be accompanied by a procedure which assures that the resulting preconditioner preserves this property. This, as well as the choice of the elements in $\tilde{\mathbf{m}}_j$ and $\tilde{\mathbf{v}}_j^T$, is discussed below.

II. CHOOSING THE ELEMENTS OF THE TRIANGULAR FACTORS

We start with a motivation for choosing the elements by value in the approximate factorization process (1.9) or, what is equivalent, (1.8).

A comparison with the exact factorization procedure

$$A_j = A_{j-1} - \frac{1}{d_j}(\mathbf{m}_j + \mathbf{f}_j)(\mathbf{v}_j^T + \mathbf{g}_j^T) \tag{2.1}$$

shows that the difference between them amounts to the matrix term

$$\frac{1}{d_j}\mathbf{f}_j\mathbf{g}_j^T \tag{2.2}$$

added to the right-hand expression in (2.1) to obtain (1.8). Hence it is reasonable to assume that the approximate factorization becomes better when the rank-one error matrix in (2.2) is "smaller."

Consider for simplicity the symmetric case $\mathbf{v}_j = \mathbf{m}_j$, $\mathbf{g}_j = \mathbf{f}_j$. The unique nonzero eigenvalue of the matrix $d_j^{-1}\mathbf{f}_j\mathbf{f}_j^T$ is $d_j^{-1}\mathbf{f}_j^T\mathbf{f}_j = d_j^{-1}\|\mathbf{f}_j\|_2^2$, corresponding to the eigenvector $\mathbf{f}_j$. Hence we improve the preconditioner when relatively small elements of the corresponding row (or column) are discarded. Moreover, the appropriate deletion criteria should be related to the norm of the discarded part.

In practice the procedure of choosing elements of the pivot column $A_{j-1}[;j]$ and the pivot row $A_{j-1}[j;]$ may start with ordering their elements in decreasing order (by absolute values). Now several ways for determining the numbers nrow and ncol of the largest elements to be kept at the jth stage of the algorithm can be suggested:

RULE 1. Let r and s denote, here and below, the numbers of nonzero elements of the original matrix lying in the pivot row and column j (below and to the right of the diagonal), respectively. Set

$$\text{nrow} = \alpha r, \qquad \text{ncol} = \alpha s, \tag{2.3}$$

where the parameter α is provided by the user. Note that with the choice $\alpha = 1$ the algorithm produces triangular factors of the same sparsity as the standard incomplete Choleski ICCG(0) algorithm. Clearly, we improve the preconditioner (but increase the complexity) when $\alpha > 1$ is taken.

RULE 2. Preserving the notation above, set

$$\text{nrow} = \left[\alpha p\left(1-\sqrt{1-\frac{rs}{pq}}\right)\right], \qquad \text{ncol} = \left[\alpha q\left(1-\sqrt{1-\frac{rs}{pq}}\right)\right]. \tag{2.4}$$

Here and below p and q stand for the numbers of nonzero elements in the pivot row and column of the active submatrix, respectively, and $[c]$ denotes the integer part of the number c.

This rule is motivated by the desire to produce a preconditioner at approximately the same cost as the usual incomplete Choleski algorithm. Indeed, the numbers of arithmetic operations required to compute the Kronecker products in (1.10) are

$$\text{ncol} \times \text{nrow}, \qquad (q - \text{ncol}) \times \text{nrow}, \qquad \text{ncol} \times (p - \text{nrow}),$$

respectively. Therefore, assuming, for simplicity, that the numbers in the brackets in (2.4) are integers and setting $\alpha = 1$, we easily deduce that the total number of operations required to compute the Kronecker products above is equal to rs, as in the standard Choleski ICCG(0) algorithm.

In practice we replace the square root in (2.4) by $1 - rs/2pq$ and use the following simplified rule:

$$\text{nrow} = \left[\alpha\frac{rs}{2q}\right], \qquad \text{ncol} = \left[\alpha\frac{rs}{2p}\right]. \tag{2.5}$$

Note that in the symmetric case $p = q$, $r = s$, we have nrow = ncol and the rule above reduces to

$$\text{nrow} = \text{ncol} = \left[\alpha\frac{s^2}{2q}\right].$$

REMARK. If permutations reducing fill-in are used (see Section IV), the numbers r and s above are replaced by the numbers

$$r = \left[\tfrac{1}{2}\alpha\tilde{r}\right], \qquad s = \left[\tfrac{1}{2}\alpha\tilde{s}\right], \tag{2.6}$$

where $\bar{r}$ (respectively, $\bar{s}$) stands for the average number of nonzero elements of the original matrix in a row (respectively, column). In this case it should be assured that the integers ncol and nrow are not too small. This is done by imposing the conditions

$$p_0 \leqslant \text{nrow} \leqslant p, \qquad q_0 \leqslant \text{ncol} \leqslant q$$

for some preassigned integers p_0 and q_0.

Rule 3. With the notation introduced above, let $\hat{r} = \max(r, \alpha\bar{r})$, $\hat{s} = \max(s, \alpha\bar{s})$. Set

$$\text{nrow} = \left[\alpha\frac{\hat{r}\hat{s}}{2q}\right], \qquad \text{ncol} = \left[\alpha\frac{\hat{r}\hat{s}}{2p}\right]. \tag{2.7}$$

Numerical experiments show that, for extremely ill-conditioned matrices, the rules (2.7) and (2.5) with the numbers r, s defined by (2.6) are the best so far. For the purpose of comparison with the incomplete Choleski factorization, it is convenient to use the latter, which, as shown above, can be computed with roughly the same arithmetic complexity as the ICCG(0) preconditioner.

Some additional rules for determining the number of elements to be kept are presented below:

Rule 4. Given a preassigned $\varepsilon > 0$, set nrow $= k_1$ and ncol $= k_2$, where k_1 and k_2 denote the largest integers satisfying the inequalities

$$\sum_{s=k_1+1}^{p} |a_s| < \varepsilon \sum_{s=1}^{p} |a_s|, \qquad \sum_{s=k_2+1}^{q} |b_s| < \varepsilon \sum_{s=1}^{q} |b_s|. \tag{2.8}$$

Here a_s and b_s stand for the nonzero elements of the active row and column, respectively. Note that, using this rule, we avoid the necessity of ordering the nonzero elements at each stage of the algorithm.

Rule 5. The choice elements here is similar to that in (2.8), but the absolute row sums on the right are replaced by the corresponding diagonal element.

Obviously, the rules above can be combined with the choice of elements by position in the following way: pick up the elements of the active row (column) belonging to the original support, or a part of it, and add to them a few largest elements outside the support of the initial matrix in this row (column).

III. REDUCING FILL-IN BY DELETING

The computation of the Kronecker products in (1.10) usually leads to fill-in the matrix $\tilde{A}_j$ in (1.9). In this section we describe a method which avoids fill-in occurring in areas II and III depicted schematically above and caused by the products

$$\frac{1}{d_j}\tilde{\mathbf{f}}_j\tilde{\mathbf{v}}_j^T, \qquad \frac{1}{d_j}\tilde{\mathbf{m}}_j\tilde{\mathbf{g}}_j^T. \tag{3.1}$$

Consider, for brevity, the symmetric positive definite case in which

$$\tilde{\mathbf{g}}_j = \tilde{\mathbf{f}}_j, \qquad \tilde{\mathbf{m}}_j = \tilde{\mathbf{v}}_j$$

To avoid the fill-in mentioned above, we propose to compute the product

$$\frac{1}{d_j}\tilde{\mathbf{f}}_j\tilde{\mathbf{v}}_j^T$$

(with elements in area II) only on the existing support of the matrix $\tilde{A}_{j-1}$, and compensate the possible loss of positive definiteness by applying the following standard procedure (see, for instance, [1]): Let c denote a nonzero element of $d_j^{-1}\tilde{\mathbf{f}}_j\tilde{\mathbf{v}}_j^T$ in position (k,r) $(k,r>1)$ not belonging to the support of $\tilde{A}_{j-1}$. Recalling that the matrices $\tilde{A}_{j-1}$ and $\tilde{A}_j$ are $(n-j+1)\times(n-j+1)$ and $(n-j)\times(n-j)$, respectively, denote by $S_{(k,r)}$ the matrix of the size of $\tilde{A}_j$, the only nonzero element of which is c in position $(k-1,r-1)$ and $(r-1,k-1)$. The effect of deleting c can be then seen as replacing the matrix $\tilde{A}_j$ in (1.9) by the matrix $\tilde{A}_j - S_{(k,r)}$. To avoid the possible loss of positive definiteness by the latter, add to it the matrix $I_{(k,r)}$, which differs from the $(n-j)\times(n-j)$ zero matrix by the element $|c|$ in positions

$(k-1,k-1)$ and $(r-1,r-1)$. Clearly, the matrix $I_{(k,r)}-S_{(k,r)}$ is positive semidefinite, and therefore

$$\left[\tilde{A}_j - S_{(k,r)}\right] + I_{(k,r)} \geqslant \tilde{A}_j > 0,$$

where the inequality $A \geqslant B$ (respectively, $A > B$) means that the matrix $A-B$ is positive semidefinite (respectively, positive definite).

Thus, if the deletion process is accompanied by adding absolute values of the elements deleted to the corresponding diagonal elements, then the resulting matrix preserves positive definiteness.

Remark. Due to the choice by value, the elements of $d_j^{-1}\tilde{\mathbf{f}}_j\tilde{\mathbf{v}}_j^T$ are small compared to the corresponding diagonal elements. Therefore, the deviation of the resulting matrix $\tilde{A}_j - S_{(k,r)} + I_{(k,r)}$ from $\tilde{A}_j$ does not much deteriorate the properties of the preconditioner.

Note that the product $d_j^{-1}\tilde{\mathbf{m}}_j\tilde{\mathbf{v}}_j^T$ is fully computed and it is the only one (assuming that the procedure above is implemented) causing fill-in (namely, in area I). We also remark that the procedure described above differs from the approach adopted in the usual incomplete Choleski method, according to which the products in (3.1) are not computed at all and hence there are no element updates in areas II and III.

IV. REDUCING FILL-IN BY PERMUTATIONS

An additional tool for reducing fill-in makes use of the *minimum-degree ordering* (see, for instance, [16]).

In the symmetric case, a local minimization of fill-in is achieved by selecting at each stage of the algorithm the most sparse row (column) of the active submatrix and interchanging them with its first row and column. In the nonsymmetric case the Markowitz counts (e.g., [16]) are used.

The interchanges are accomplished before sorting elements of the first row and column and performing an incomplete elimination.

Usually there are several rows and columns with the same minimal number of nonzero elements. In the case of a symmetric positive definite matrix, it is recommended to select the row (or column) with the least ratio of the sum of absolute values of the elements to the corresponding diagonal element. Hence an incomplete elimination of "worse" rows and columns is performed at later stages of the algorithm.

Note that at the first stages of the algorithm the numbers nrow and ncol computed by one of the rules of Section II are, due to permutations, usually equal to the actual numbers of nonzero elements in the corresponding row and column. Hence one performs in fact a complete *LDU* factorization at the first stages of the algorithm.

V. A DESCRIPTION OF THE ALGORITHM

The algorithm is presented below in a compact verbal-matrix form to emphasize, in particular, its relation to the previous exposition. For brevity, the given $n \times n$ nonsingular matrix A is assumed to be real and symmetric: $A = A^T$. Also, the rule (2.5) for determining the number of largest elements to be kept is chosen. Since permutations are used, this rule is combined with the average number of nonzero elements in a row [see (2.6)].

In presenting the algorithm we use the previous notation and, for simplicity, describe it in terms of $n \times n$ matrices and vectors of order n.

THE ALGORITHM.

0. Construct an integer array N of the number of nonzero off-diagonal elements in each column.
   Compute the average number s of nonzeros in a column.
   Choose the parameters α and q_0.
   Set $A_0 = A$.

For $j = 1,2,\dots,n-1$ **do**

1. Permutation. Select column of A_{j-1} of minimal sparsity, say, column j, having q nonzero elements.
   Interchange row j and column j with the first row and column of A_{j-1}.
   Denote the resulting matrix by $\hat{A}_{j-1}$.
2. *Determining the number of largest elements.* Compute

$$\text{ncol} = \text{nrow} = \left[\alpha \frac{s^2}{2q}\right].$$

Set

$$\text{ncol} = \max(q_0, \text{ncol}), \qquad \text{ncol} = \min(\text{ncol}, q_0).$$

3. *Selecting elements and updating the triangular factors.* Select the largest (by absolute values) ncol off-diagonal elements of column j, and denote the resulting vector by $\mathbf{m}_j$. Set

$$d_j = \hat{A}_{j-1}[j;j] \quad \text{and} \quad \mathbf{l}_j = \frac{1}{d_j}\mathbf{m}_j.$$

4. Elimination. Compute

$$\hat{A}_j = \hat{A}_{j-1} - \frac{1}{d_j}\mathbf{m}_j\mathbf{m}_j^T - \left(\mathbf{m}_j\mathbf{f}_j^T + \mathbf{f}_j\mathbf{m}_j^T\right), \tag{5.1}$$

where $\mathbf{f}_j = \hat{A}_{j-1}[;j] - \mathbf{m}_j$. [Recall that a reduction of fill-in can be achieved by discarding those elements of the last summand in (5.1) which lie outside the existing support and updating the corresponding diagonal elements—see Section III.]

5. Update the integer array N, and replace A_{j-1} by $\hat{A}_j$.

The algorithm described produces a left triangular matrix L and a diagonal matrix D defined by

$$L = [\mathbf{m}_1 \quad \mathbf{m}_2 \quad \cdots \quad \mathbf{m}_n], \qquad D = \operatorname{diag}[d_1, d_2, \ldots, d_n]$$

and gives an incomplete factorization of a permuted version of the matrix A.

Note that an implementation of the algorithm requires a different approach from familiar incomplete factorization algorithms. Details of implementation, as well as numerical experiments with very large and extremely ill-conditioned symmetric and nonsymmetric linear systems, will be reported in a forthcoming paper.

In the following section we present the results of some experiments with relatively small but very ill-conditioned symmetric positive definite matrices arising in several fields of applications.

VI. NUMERICAL EXPERIMENTS

Several versions of the algorithm proposed are compared in this section with the standard incomplete Choleski preconditioner. In all of them the number of elements to be kept is determined by the simplified rule 2 [see (2.5) and (2.6)].

In the first version, called Rob0, the incomplete elimination is performed according to (1.9). In the second and the third versions, named Rob10 and Rob20, respectively, we delete the elements in areas II and III causing fill-in (see Section III). In the version Rob20 the absolute values of the deleted elements are added to the diagonal (to preserve the positive definiteness).

The algorithms Rob, Rob1, and Rob2 are permuted variants of the above algorithms, respectively (see Section IV). Note that there is no guarantee that the algorithms Rob1 and Rob10 preserve the positive definiteness of the original matrix. However, in some cases these algorithms may perform better than the others (see Tables 2 and 5 below).

The algorithm are applied to two groups of problems:

(1) Linear systems occurring in structural analysis.
(2) Linear systems arising in semiconductor device simulation.

The numbers N_{prec} of arithmetic operations required to produce the preconditioner and N_{total} to obtain the solution are given in thousands of flops. The user-controlled parameter α is always chosen to be an integer.

The stopping rule for the conjugate-gradient iterations is

$$\frac{\|\mathbf{r}_k\|}{\|\mathbf{r}_0\|} < 10^{-10},$$

and it has been always verified that the convergence (if any) is to the correct solution. The zero vector is used as a starting vector.

We start with demonstrating the well-known fact of loss of positive definiteness by the incomplete Choleski preconditioner. To this end, we use

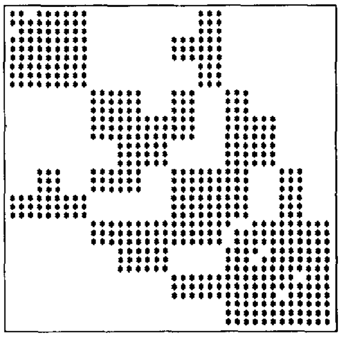

FIG. 1. Nonzero patterns of A_0.

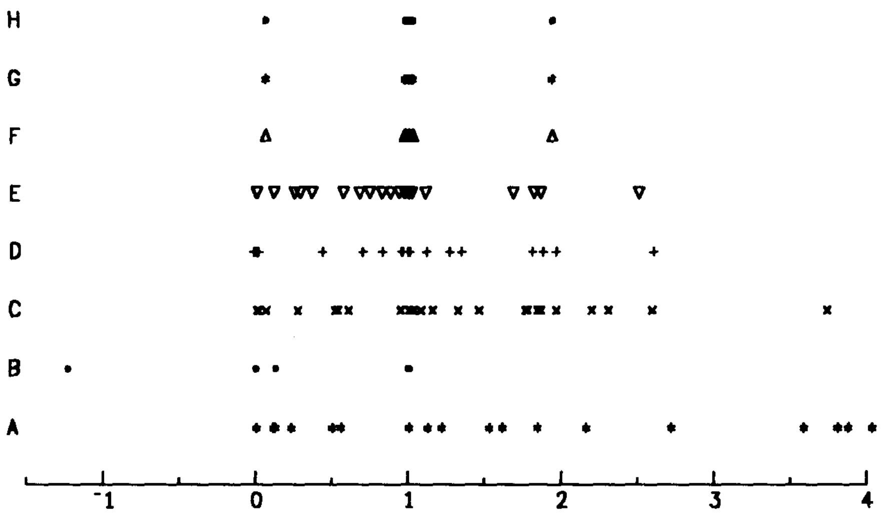

FIG. 2. Eigenvalue distribution of A_0 and the preconditioned matrices.

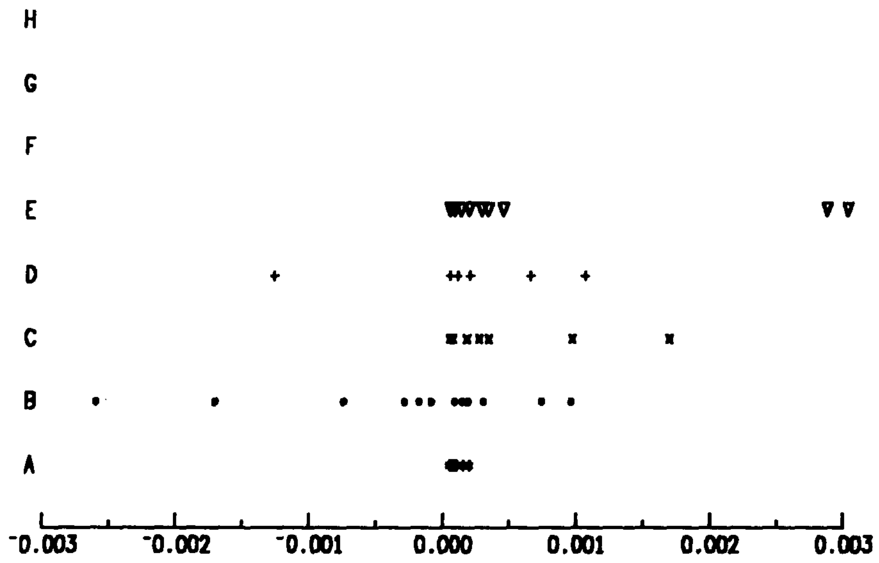

FIG. 3. Eigenvalue distribution near zero.

TABLE 1
THE 36×36 MATRIX A_0

Algorithm	α	Density AA	Density L	N_{prec}	Condition
Cholesky		792	310	3.2	
Rob0	1	890	159	2.9	7×10^4
Rob10	2	660	246	2.5	
Rob20	2	660	296	3.1	5×10^4
Rob	2	612	297	2.8	32
Rob1	2	612	296	2.8	32
Rob2	2	612	296	2.8	32

the 36×36 leading principal submatrix A_0 of the symmetric positive definite matrix A_2 described later. The spectral condition number of the matrix A_0 is approximately 9×10^4. The support of the matrix is shown in Figure 1.

In Figure 2 the eigenvalue distribution of the original and preconditioned matrices is presented. The spectra of the matrix A_0 and the Choleski preconditioned matrix are shown in lines A and B, respectively. The spectra of preconditioned matrices produced by the algorithms Rob0, Rob10, Rob20, Rob, Rob1, Rob2 are depicted in that order in lines C–H.

The eigenvalue distribution of the matrices mentioned above in a neighborhood of the origin is shown in Figure 3.

Observe that the Choleski and the Rob10 preconditioners do not preserve the positive definiteness of the original matrix A_0.

In Table 1 the effect of preconditioning for the matrix A_0 above is shown. Note that the density of the matrix AA roughly reflects the fill-in of the matrix A_n in (0.1) replaced during incomplete factorization by its diagonal. Since (1.9) is actually used instead of (1.8), the equality $AA = A_n$ does not hold true any more.

The next matrix tested is generated by the structural analysis simulator NASTRAN. It is called the BCLL4 matrix and represents a finite-element structural-analysis problem over a $4 \times 4 \times 4$ cube. There are 121 nodes, six variables in each, which amounts to a system of order 726. The apparent discrepancy between the cubic geometry and the number of nodes is due to four missing corners corresponding to fixed supports. The matrix A_1 used for Table 2 is the leading principal submatrix of order 144 of the matrix BCLL4. Note that A_1 is a symmetric positive definite matrix with 3270 nonzero elements. The spectral condition number of this matrix is 3.7×10^8. Nevertheless, this is an example where the incomplete Choleski preconditioner exhibits attractive performance.

Numerical experiments with two symmetric positive definite matrices called A_2 and A_3 arising in metal forming are presented in Tables 3 and 4.

TABLE 2
THE 144×144 MATRIX A_1

Algorithm	α	Density L	N_{prec}	Iterations	N_{total}
Cholesky		1713	23.1	11	**192.8**
Rob0	1	574	23.8	22	253.7
	2	907	42.8	13	197.8
Rob10	1	894	12.2	15	189.7
	2	1246	21.6	11	167.2
Rob20	1	894	14.5	16	203.7
	2	1246	21.6	12	183.3
Rob	1	689	20.4	18	217.6
	2	996	34.3	14	206.4
Rob1	1	899	13.6	15	179.8
	2	1257	26.9	10	**157.3**
Rob2	1	896	15.4	15	193.9
	2	1259	26.9	11	173.1

Both matrices are 102×102 and provide a finite-element description of the deformation of a cylindrical piece of metal heated up to a certain temperature and then pressed by a die. The spectral condition number of the first matrix, having 2140 nonzero elements, is roughly 1.8×10^5. The second matrix has 2092 nonzeros and a condition number of order 10^{10}. Note that the nonpermuted versions of the algorithm result in a very slow convergence or even divergence of conjugate-gradient iterations. The values of α shown are the least integers for which the convergence rate is satisfactory.

TABLE 3
THE 102×102 MATRIX A_2

Algorithm	α	Density L	N_{prec}	Iterations	N_{total}
Cholesky		1121	15.3	93	**929.3**
Rob	3	1232	19.9	15	161.1
	4	1409	21.2	6	**89.7**
Rob1	3	1245	18.8	43	465.1
	4	1409	20.9	7	102.2
Rob2	3	1247	19.3	29	324.4
	4	1409	20.9	8	113.3

TABLE 4
THE 102×102 MATRIX A_3

Algorithm	α	Density L	N_{prec}	Iterations	N_{total}
Cholesky		1097	14.9	86	**846.9**
Rob	3	1209	19.6	16	184.3
	4	1385	20.8	8	**111.3**
Robl	3	1226	18.3	—	—
	4	1383	20.7	10	132.6
Rob2	3	1220	19.0	49	517.3
	4	1385	20.7	11	143.4

The incomplete Choleski and Robl with $\alpha = 3$ preconditioners produce indefinite matrices here. Hence the slow convergence of the iterations. Note that both preconditioners cause divergence of the conjugate gradients when applied to the whole matrix.

As in the previous case, the Choleski preconditioner and the Robl algorithm with $\alpha = 3$ produce here matrices with negative elements in the diagonal factor.

TABLE 5
THE 100×100 MATRIX A_4

Algorithm	α	Density L	N_{prec}	Iterations	N_{total}
Cholesky		280	0.52	19	**59.5**
Rob0	1	199	0.97	34	94.6
	2	216	1.6	32	96.8
Robl0	1	289	0.77	17	54.4
	2	347	1.8	14	48.8
Rob20	1	289	9.0	17	54.5
	2	347	1.2	14	48.9
Rob	1	270	1.3	28	54.5
	2	345	1.8	21	72.7
Robl	1	294	0.78	17	54.7
	2	384	1.3	11	**40.8**
Rob2	1	294	0.78	16	51.7
	2	384	1.44	12	44.4

The last matrix tested is derived from a semiconductor simulator for the electron continuity equation on a 10×10 diode. It is a symmetric negative definite matrix of order 100 having 460 nonzeros and is extremely ill conditioned. Scaling reduces the condition number of the matrix drastically (to approximately 464). The symmetrically scaled positive definite version of this matrix is denoted by A_4 and tested in Table 5.

VII. CONCLUDING REMARKS

In all the reported runs most versions of the algorithm perform better than the incomplete Choleski preconditioner. When the matrices are extremely ill conditioned, all the variants of the algorithm with permutations and are sufficiently large value of the parameter are superior to the ICCG(0) algorithm.

The algorithm is capable of achieving impressive speedup factors for some groups of problems. Many of the tests which led to failure of the incomplete Choleski preconditioner were successfully tackled by the new algorithm suggested here.

A FORTRAN code of the algorithm, now under development in the IBM Heidelberg Scientific Center, has been applied to several midsize symmetric positive definite matrices arising in applications and has been compared with the incomplete Choleski preconditioned conjugate gradients and the direct method as implemented in the Engineering and Scientific Subroutine Library package (ESSL). Tested on the 1919×1919 matrix of Platzman with 32,399 nonzero entries derived from a finite-difference discretization of a three-dimensional ocean model, a preliminary draft of the Rob10 algorithm with $\alpha = 1$ produces the solution in 9 iterations (CPU time on the IBM 3090 machine: 6.07 seconds). It converges in 11 iterations and requires only 3.41 seconds when $\alpha = 0.2$. Experiments with a 957×957 matrix of Simon (4137 nonzeros) describing a finite-element approximation to the biharmonic operator on a beam show, for instance, that the solution can be obtained by Rob1 in 0.84 seconds (14 iterations) when $\alpha = 1$. In both cases the incomplete Choleski subroutine of ESSL failed (since negative numbers appear on the main diagonal), while the cost of the direct method is 5–6 times higher.

It is difficult to choose between the variants of the algorithm. Examination of the tables above indicates the superiority of the algorithms with permutations. Also, the Rob algorithm is more reliable and seems to perform better for extremely ill-conditioned problems, whereas Rob1 seems to be better in other cases.

Our experience with nonsymmetric systems is much smaller. However, a few tests carried out show that the algorithm compares favorably with the usual incomplete *LDU* factorization technique for accelerating the biconjugate-gradient iterations. This, as well as details of implementation and a numerical study of other variants of the algorithm, will be reported in a subsequent work.

The author wishes to thank Dr. V. Amdursky for encouraging this work and fruitful discussions, and the referees for useful suggestions.

REFERENCES

1 M. A. Ajiz and A. Jennings, A robust incomplete Cholesky–conjugate gradient algorithm, *Internat. J. Numer. Methods Engrs.* 20:949–966 (1984).

2 O. Axelsson, A survey of preconditioned iterative methods for linear systems of algebraic equations, *BIT* 25:166–187 (1985).

3 O. Axelsson, Solution of linear systems of equations: Iterative methods, in *Sparse Matrix Techniques* (V. A. Barker, Ed.), Springer-Verlag, 1977, pp. 1–51.

4 P. Concus, G. H. Golub, and D. P. O'Leary, A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in *Sparse Matrix Computations* (J. R. Bunch and D. J. Rose, Eds.), Academic, New York, 1976, pp. 309–332.

5 I. S. Duff and G. Meurant, The Effect of Ordering an Preconditioned Conjugate Gradients, Report CSS 221, Harwell Lab. Sept. 1988.

6 S. C. Eisenstat, Efficient implementation of a class of preconditioned conjugate gradient methods, *SIAM J. Sci. Statist. Comput.* 2:1–4 (1981).

7 D. J. Evans (Ed., *Preconditioning Methods, Theory and Applications*, Gordon and Breach, New York, 1983.

8 Gambolati, Pini, and Zilly, Numerical comparison of preconditioning for large sparse finite elements problems, *Numer. Methods Partial Differential Equations* 4:139–157 (1988).

9 I. Gustafsson, A class of first order factorizations, *BIT* 18:142–156 (1978).

10 I. Gustafsson, On modified incomplete factorization methods, in *Lecture Notes in Math.* 96 (A. Dold and B. Eckmann, Eds.), Springer-Verlag, New York, 1980, pp. 334–351.

11 C. P. Jackson and P. C. Robinson, A numerical study of various algorithms related to the preconditioned conjugate gradient method, *Internat. J. Numer. Methods Engrs.* 21: 1315–1338 (1985).

12 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).

13 P. Lancaster and M. Tismenetsky, *The Theory of Matrices*, Academic, Orlando, Fla., 1985.

14 T. A. Manteuffel, An incomplete factorization technique for positive definite linear systems, *Math. Comp.* 24:473–480 (1980).

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

16 S. Pissanetzky, *Sparse Matrix Technology*, Academic, London, 1984.

17 A. D. Tuff and A. Jennings, An iterative method for large systems of linear structural equations, *Internat. J. Numer. Methods Engrs.* 7:175–183 (1973).

18 Y. S. Wong, Preconditioned conjugate gradient methods for large sparse matrix problems, in *Numerical Methods in Thermal Problems* (R. W. Lewis and K. Morgan, Eds.), Pineridge Press, 1979, pp. 967–979.

Received 30 April 1990; final manuscript accepted 18 September 1990