MATHEMATICS OF COMPUTATION, VOLUME 24, NUMBER 112, OCTOBER 1970

# Some Results on Sparse Matrices*

By Robert K. Brayton, Fred G. Gustavson and Ralph A. Willoughby

**Abstract.** A comparison in the context of sparse matrices is made between the Product Form of the Inverse PFI (a form of Gauss-Jordan elimination) and the Elimination Form of the Inverse EFI (a form of Gaussian elimination). The precise relation of the elements of these two forms of the inverse is given in terms of the nontrivial elements of the three matrices $L$, $U$, $U^{-1}$ associated with the triangular factorization of the coefficient matrix $A$; i.e., $A = L \cdot U$, where $L$ is lower triangular and $U$ is unit upper triangular. It is shown that the zero-nonzero structure of the PFI always has more nonzeros than the EFI. It is proved that Gaussian elimination is a minimal algorithm with respect to preserving sparseness if the diagonal elements of the matrix $A$ are nonzero. However, Gaussian elimination is not necessarily minimal if $A$ has some zero diagonal elements. The same statements hold for the PFI as well. A probabilistic study of fill-in and computing times for the PFI and EFI sparse matrix algorithms is presented. This study suggests quantitatively how rapidly sparse matrices fill up for increasing densities, and emphasizes the necessity for reordering to minimize fill-in.

**I. Introduction.** A sparse matrix is a matrix with very few nonzero elements. In many applications, a rough rule seems to be that there are $O(N)$ nonzero entries; typically, say 2 to 10 nonzero entries per row. If the dimension $N$ of the matrix is not large, then there is no compelling reason to treat sparse matrices differently from full matrices. It is when $N$ becomes large and one attempts computations with the sparse matrix that it becomes necessary to take advantage of the zeros. The reason for this is obvious: there is a storage requirement of the order of $N^2$ and arithmetic operations count of the order of $N^3$ for many matrix algorithms using the full matrix. On the other hand, by storing only nonzero quantities and using logical operations to decide when an arithmetic operation is necessary, the storage requirement and arithmetic operations count can be reduced by a factor of $N$ in many instances. Of course, this not only becomes a sizable savings of computer time, but also dictates whether or not some problems can be attempted.

Computations with sparse matrices are not new. Iterative techniques for these matrices, especially those related to the solution of partial differential equations have been extensively developed [1]. Sparse matrix methods for solving linear equations by direct methods have been used for a long time in linear programming and there is a large body of literature, computational experience, programs, and artfulness, which has been built up in this area [2]–[15]. In most linear programming codes, the product form of the inverse (PFI) is the method used to solve linear equations [16]–[19], although there are exceptions [20]–[21]. Methods for scaling, pivoting for accuracy and sparseness, structuring data, and handling input-output have been extensively developed [22]–[72]. However, there do not seem to exist rigorous results

Received February 3, 1969, revised March 16, 1970.

*AMS* 1969 *subject classifications.* Primary 6535, 9050; Secondary 1515.

*Key words and phrases.* Sparse matrices, elimination form of inverse, product form of inverse, fill-in, symbolic and numerical zeros, minimal algorithms.

* The results reported in this paper were obtained in the course of research jointly sponsored by the Air Force Office of Scientific Research (Contract AF 49(638)-1474) and IBM.

which compare one method against another, results which state that one method is best, or results which give lower bounds for the best method. Some of the results in this paper provide answers in these directions.

Recently, in a number of other areas of applications, increased interest has been shown in sparse matrix methods, namely, electrical networks, structural engineering, and power distribution systems [73]–[107]. The reason for this interest seems to be inspired by the attempt to do larger problems which, in turn, seems to be inspired by the availability of faster and larger computers. This interest has generated a number of general sparse matrix programs for solving linear equations, which are under development, and indeed has given rise to new ideas. The availability of sparse matrix codes [108]–[109] has stimulated interest in other application areas and, therefore, further increase the desirability of advancing research in this area.

In this paper we analyze certain direct methods of solving $Ax = b$, where $A$ is a sparse matrix. However, we do not want to imply that the investigation of other matrix algorithms in the context of sparsity (e.g., eigenvalue, eigenvector computations) is not important.

In the second section, we compare, in the context of sparsity, two well-known direct methods for solving $Ax = b$, namely, the elimination form of the inverse (EFI), a form of Gaussian elimination, and the product form of the inverse (PFI), a form of Gauss-Jordan elimination. It is proved that the upper triangular part of the PFI is exactly the negative of the inverse of the matrix $U$ involved in Gaussian elimination. The lower triangular part is exactly the $L$ of Gaussian elimination. The comparison of the two methods, therefore, requires a method of comparing the sparseness of $U$ with the sparseness of $U^{-1}$. This leads to the next section in the paper.

In the third section we define the concept of the Boolean form of an algorithm. For example, for Gaussian elimination, the matrix $A$ is factored into $L \cdot U$. By replacing arithmetic operations by logical operations in the natural way, the same algorithm, given $A_s$ (the Boolean matrix representing the zero-nonzero structure of $A$) will produce the Boolean matrices $L_s$ and $U_s$. The concept of minimal algorithms in the sense of sparsity is discussed, i.e., an algorithm which produces the minimal Boolean matrix and, therefore, requires the minimal amount of storage. It is shown by example that Gaussian elimination and the Householder method [59] for factoring $A = QU$, $Q'Q = I$ are *not* minimal algorithms. It is clearly seen in these examples how nonminimality produces unnecessary arithmetic operations as well as unnecessary storage requirements. It is then proved that Gaussian elimination is a minimal algorithm if the diagonal elements of the matrix $A$ are nonzero. It is shown that the PFI must produce a Boolean matrix $(U^{-1})_s$ which is fuller than $U_s$ produced by Gaussian elimination. In fact, if the diagonal elements of $A$ are nonzero, no method of solving linear equations requiring the computation of $U^{-1}$ can require less storage than Gaussian elimination. Also, the minimal algorithms for computing $U$ and $U^{-1}$ when the diagonal elements of $A$ may be zero are given precisely. These algorithms combine the Gaussian elimination or the Gauss–Jordan algorithms with a test for a property on submatrices of $A$ (called property $R$).

In Section IV, a probabilistic study of fill-in is discussed. The results of this study verify some of the conclusions of the previous section. They also indicate how rapidly the matrices can fill up and, therefore, emphasize quantitatively the importance of pivoting for reducing the fill-in.

**II. A Comparison Between EFI and PFI for Sparse Matrices.** The EFI is a form of Gaussian elimination where one transforms the matrix $A$ into a unit upper triangular matrix $U$. The PFI is a form of Gauss-Jordan total elimination where one transforms $A$ into the identity matrix $I$. In this discussion, we will not include pivoting (which introduces row and column permutations), but we will briefly discuss (at the end of this section) how the same pivoting strategy can be carried out for either scheme. The notation $(A)_{ij}$ will be used for the element of the matrix $A$ in the $i$th row and $j$th column.

Certain elementary matrices, which differ from the identity matrix in only one column, are the primary tool for both algorithms. Let $t_k = \mathrm{col}\,(t_{1k}, \cdots, t_{nk})$, $e_k = k$th column of the $n \times n$ identity matrix $I$, and $'$ denote the transpose operation. An *elementary matrix* is any matrix of the form

$$T_k = \begin{bmatrix} 1 & & & t_{1k} & & & \\ & \ddots & & \vdots & & & \\ & & 1 & & & & \\ & & & t_{kk} & & & \\ & & & & 1 & & \\ & & & \vdots & & \ddots & \\ & & & t_{nk} & & & 1 \end{bmatrix} = I + (t_k - e_k)e_k'.$$

If $t_{kk} \neq 0$, then $T_k^{-1}$ exists and $T_k^{-1} = I - t_{kk}^{-1}(t_k - e_k)e_k'$. Note that the zero-nonzero structure of an elementary matrix is the same as its inverse.

*PFI Algorithm.* Elementary matrices $T_k$ for $k = 1, \cdots, n$ are formed with the property that

$$T_k^{-1}(T_{k-1}^{-1} \cdots T_1^{-1}A)e_k = e_k$$

so that $T_n^{-1} \cdots T_1^{-1}A = I$. Thus,

$$T_k = I + (t_k - e_k)e_k',$$

where $t_k = T_{k-1}^{-1} \cdots T_1^{-1}Ae_k$. It is assumed that $t_{kk} \neq 0$ for $k = 1, \cdots, n$.

*EFI Algorithm.* Elementary *lower triangular* matrices $L_k$ are formed for $k = 1, \cdots, n$ with the property that

$$L_k^{-1}(L_{k-1}^{-1} \cdots L_1^{-1}A)e_k = b,$$

where

$$\begin{aligned} b_j &= (L_{k-1}^{-1} \cdots L_1^{-1}A)_{jk} && \text{for } j < k, \\ &= 1 && \text{for } j = k, \\ &= 0 && \text{for } j > k. \end{aligned}$$

Thus, $L_k = I + (c - e_k)e_k'$, where

$$\begin{aligned} c_j &= 0 && \text{for } j < k, \\ &= (L_{k-1}^{-1} \cdots L_1^{-1}A)_{jk} && \text{for } j \geqq k. \end{aligned}$$

The result is that

$$L_n^{-1} \cdots L_1^{-1} A = U,$$

where $U$ is unit upper triangular with

$$\begin{aligned} (U)_{ij} &= (L_{j-1}^{-1} \cdots L_1^{-1} A)_{ij} && \text{for} \quad i < j, \\ &= 1 && \text{for} \quad i = j, \\ &= 0 && \text{for} \quad i > j. \end{aligned}$$

Our aim is to relate the quantities $T_k$, $L_k$, and $U$ of the PFI and EFI algorithms. With $t_k$ as defined in the PFI algorithm, let

$$\begin{aligned} v_k &= \operatorname{col}(0, \cdots, 0, t_{kk}, \cdots, t_{nk}), \\ w_k &= \operatorname{col}(t_{1k}, \cdots, t_{k-1,k}, 0, \cdots, 0), \\ V_k &= I + (v_k - e_k)e_k', \\ W_k &= I + w_k e_k'. \end{aligned}$$

It is easily shown that

(a) $V_k W_k = T_k$ for $k = 1, \cdots, n$,

(b) $V_k W_j = W_j V_k$ for $1 \leqq j < k \leqq n$.

THEOREM 1. *$V_k = L_k$ for $1 \leqq k \leqq n$.*

*Proof.* It is clear that $L_1 = T_1 = V_1$. Assume that, for $j < k$, $V_j = L_j$. Then using (b) we have

$$T_{k-1}^{-1} \cdots T_1^{-1} A = (W_{k-1}^{-1} \cdots W_1^{-1})(L_{k-1}^{-1} \cdots L_1^{-1}) A.$$

Since $W_j^{-1}$ is an elementary unit upper triangular matrix it has the property that $b_i = (W_j^{-1} b)_i$ for $i \geqq j$. Thus, $(T_{k-1}^{-1} \cdots T_1^{-1} A)_{jk} = (L_{k-1}^{-1} \cdots L_1^{-1} A)_{jk}$ for $j \geqq k$. From the way that $T_k$, $V_k$ and $L_k$ are formed, it is clear that these are the nontrivial elements of $L_k$ and $V_k$. Thus, by induction $L_k = V_k$ for $k = 1, \cdots, n$. Q.E.D.

Let $L = L_1 \cdots L_n$.

LEMMA. *$L = I + \sum_{j=1}^{n} (v_j - e_j)e_j'$. I.e. $(L)_{ij} = (L_j)_{ij}$.*

*Proof.* Since $L_j = V_j = I + (v_j - e_j)e_j'$ and $e_i'(v_j - e_j) = 0$ for $i < j$, then

$$L = L_1 \cdots L_n = I + \sum_{j=1}^{n} (v_j - e_j)e_j'. \qquad \text{Q.E.D.}$$

This lemma states that multiplication of the $L_k$ is trivial and that $L$ is obtained by superposition.

LEMMA. *$U = W_1 \cdots W_n$.*

*Proof.* Since $T_n^{-1} \cdots T_1^{-1} A = I = W_n^{-1} \cdots W_1^{-1} L_n^{-1} \cdots L_1^{-1} A$ and $U = L_n^{-1} \cdots L_1^{-1} A$, then

$$W_n^{-1} \cdots W_1^{-1} U = I. \qquad \text{Q.E.D.}$$

Note that here the multiplication of $W_k$ is not trivial.

LEMMA. *$U^{-1} = I - \sum_{j=1}^{n} w_j e_j'$. I.e. $(U^{-1})_{ij} = -(W_j)_{ij}$ for $i < j$.*

*Proof.* By the previous lemma $U^{-1} = W_n^{-1} \cdots W_1^{-1}$. Since $W_j^{-1} = I - w_j e_j'$ and $e_i' w_j = 0$ for $i \geqq j$, then

$$U^{-1} = W_n^{-1} \cdots W_1^{-1} = I - \sum_{j=1}^{n} w_j e_j'. \qquad \text{Q.E.D.}$$

As in the first lemma the multiplication of the $W_k^{-1}$ is trivial.

THEOREM 2. $t_j = (L + I - U^{-1})e_j$ *for* $j = 1, \cdots, n$.

*Proof.* This is just a summation of the results of the three lemmas. Q.E.D.

In using these algorithms for computing solutions to linear equations where the matrix of coefficients is sparse, one stores only the nontrivial data, i.e., not the zeros and not the ones occurring along the diagonal. Furthermore, if one adopts column-wise storage, then it is more efficient to operate on the matrix column-wise. This is because it is more complicated to find, for example, the elements in the same row if the data is stored column-wise.

It is clear that the PFI algorithm can be implemented column-wise (or row-wise). For the EFI this is not so obvious. To do this we require a factored form of $U^{-1}$, which uses the data of $U$. Markowitz [20] obtained a factored form for $U^{-1}$ for the EFI in terms of the *rows* of $U$. Since the $L_j$ are obtained column-wise we require a form in terms of the *columns* of $U$. Let

$$u_j = (U - I)e_j, \qquad U_j = I + u_j e_j'.$$

Then

LEMMA. $U = U_n \cdots U_2$.

*Proof.* Since $U_j = I + u_j e_j'$ and $e_i' u_j = 0$ for $i \geqq j$, then

$$U_n \cdots U_2 = I + \sum_{j=2}^{n} u_j e_j' = U. \qquad \text{Q.E.D.}$$

Note that multiplication of the $U_k$ in reverse order is trivial.

Thus, for the EFI we have

$$U_2^{-1} \cdots U_n^{-1} L_n^{-1} \cdots L_1^{-1} A = I,$$

the elements requiring storage are the nontrivial elements of $L$ and $U$, and the data can be stored and operated on column-wise.

Note for comparison that the PFI yields

$$W_{n-1}^{-1} \cdots W_1^{-1} L_n^{-1} \cdots L_1^{-1} A = I$$

but $W_k \neq U_k$.

Since only the nontrivial elements are stored, the zero-nonzero structure of the data is important.

*Definition.* Let $M_s$ denote the matrix of 0's and 1's obtained from a matrix $M$ by replacing nonzero quantities by 1.

The comparison between the EFI and PFI in terms of storage is given in Theorem 2 and is restated.

*PFI Storage Requirements.* $(L + I - U^{-1})_s$.

*EFI Storage Requirements.* $(L - I + U)_s$.

The storage requirements are, of course, the number of 1's in each of the above matrices.

The comparison is, therefore, between the number of nonzero elements in $U$ and $U^{-1}$. In general, it is impossible to say that one is sparser than the other because the statement $U$ sparser than $U^{-1}$ can be applied to a matrix $V \equiv U^{-1}$. For purposes of computation it is important to distinguish between two types of zeros obtained by an algorithm:

(a) Zeros which are the result of multiplication and addition by zero. These can be detected by logical operations on the zero-nonzero structure of the matrix and, therefore, require no floating-point arithmetic operations. They are independent of the numerical values of the nonzero elements.

(b) Zeros which result by exact numerical cancellation given no round-off error. These cannot be detected by logical operations and in the presence of round-off error are difficult to detect by examining the numerical result. In any case, they require floating-point arithmetic operations and are dependent on the numerical values of the nonzero elements.

In the next section, the comparison between the number of zeros of type (a) of $U$ and $U^{-1}$ is obtained.

Pivoting strategies are easy to implement, but for the EFI this is not obvious. Following the usual methods, we can process the columns of $A$ in any order, and if the order is other than the natural one, we simply form a permutation $p(k)$ to indicate that at the $k$th stage we are processing the $p(k)$th column of $A$. Second, we can choose any pivot position in the $p(k)$th column corresponding to any row not previously chosen. This introduces a second permutation, $q(k)$, which indicates the pivot position for the $k$th stage. Thus, for the PFI with $t_{p(k)} = T_{k-1}^{-1} \cdots T_1^{-1}Ae_{p(k)}$ we want $T_k^{-1}a = e_{q(k)}$ so $a = T_k e_{q(k)}$ and hence, $T_k = I + (a - e_{q(k)})e'_{q(k)}$.

The situation for the EFI is only different in that in transforming the corresponding vector $a$ we leave invariant the positions corresponding to previous pivot positions. This means that $L_k$ is no longer lower triangular, but the $q(k)$th column of $L_k$ has zeros in chosen pivot rows and otherwise contains the corresponding element of $a$. The elements of $a$ not used in $L_k$ form the nontrivial part of the $q(k)$th column of $U$. As before, we have $T_k = L_k W_k$, where $W_k$ is related to $U^{-1}$, and for $j < k$, $L_k W_j = W_j L_k$.

**III. Minimality of Algorithms for Sparse Matrices.** The following notation and definitions are used in this section.

$(A)_{ij}$—the element of $A$ in the $i$th row and $j$th column.

$A_{ij}$ $(i \geqq j)$—the $j \times j$ matrix obtained from $A$ by deleting rows $\mu$ for $j \leqq \mu < i$, $i < \mu \leqq n$, and columns $\mu$ for $j < \mu \leqq n$.

$A_{ij}$ $(i < j)$—the $i \times i$ matrix obtained by deleting rows $\mu$ for $i < \mu \leqq n$ and columns $\mu$ for $i \leqq \mu < j$, $j < \mu \leqq n$.

$\tilde{A}_{ij}$ $(i < j)$—the $(j - 1) \times (j - 1)$ matrix obtained from $A_{j-1,j-1}$ by replacing its $i$th column with the first $j - 1$ components of $Ae_j$.

*Definition* 1. A *Booleanization* of a computational algorithm is obtained by replacing each nonzero operand by a 1; multiplication by logical and, addition and subtraction by logical or; and division by logical and if the divisor is one, otherwise stop. (See below for further discussion.)

*Definition* 2. $B_s \subseteq A_s$ denotes the fact that $A_s$ has a one wherever $B_s$ has a one.

*Definition* 3. The output of a Booleanized algorithm $\alpha$ is denoted by $B_s^\alpha$, symbolically $A_s \xrightarrow{\alpha} B_s^\alpha$. This is used to distinguish the fact that the output depends on the algorithm used.

*Definition* 4. By numerical data $A$ *representing* $A_s$, we mean that given a Boolean matrix $A_s$, then $(A_s)_{ij} = 0$ implies that $(A)_{ij} = 0$ but not necessarily the converse.

*Definition* 5. An algorithm $\alpha$ is *s-minimal* if given any Boolean matrix $A_s$ and integers $i$, $j$ such that $(B_s^\alpha)_{ij} = 1$ (where $A_s \xrightarrow{\alpha} B_s^\alpha$), there exists $A$ representing $A_s$ such that $(B)_{ij} \neq 0$ (where $A \xrightarrow{\alpha} B$).

*Definition* 6. An algorithm $\alpha$ is $o$-minimal if it requires the least number of arithmetic operations (counting addition and multiplication equally) of all algorithms which compute the same thing (neglecting round-off).

*Definition* 7. A matrix $M$ has *property R* if there exists a rearrangement of the rows and columns of $M$ which puts nonzero entries on the diagonal (i.e., the permanent of $M_s$ is nonzero).

The notion of "*Booleanizing*" a computational algorithm $\alpha$ needs to be discussed further. Consider an example. The Crout algorithm performs the factorization $A = LU$, where $L$ and $U$ are computed by the formulae

$$\begin{aligned} l_{ik} &= a_{ik} - \sum_{\mu=1}^{k-1} l_{i\mu}u_{\mu k}, && i \geqq k, \\ u_{ik} &= \left(a_{ik} - \sum_{\mu=1}^{i-1} l_{i\mu}u_{\mu k}\right) \Big/ l_{ii}, && i < k. \end{aligned} \tag{3.1}$$

This notion, as applied to the Crout algorithm, is motivated by the logical process of deciding when an arithmetic operation is necessary in the processing of a sparse matrix. It is clear that the Crout algorithm, when converted, will take a Boolean matrix $A_s$ and produce Boolean matrices (see Definition 3) $L_s^c$ and $U_s^c$, provided division by zero is not encountered.

It should be clear that algorithms which compute the same numerical results will not, in general, produce the same Boolean output. Obviously an algorithm which (a) computes $A^{-1}$ by PFI, (b) triangularizes $A^{-1}$ to obtain $L^{-1}$ and $U^{-1}$, (c) computes $L$ and $U$ from $L^{-1}$ and $U^{-1}$ by PFI will (neglecting round-off) produce the same matrices $L$ and $U$ as the Crout algorithm, but will generally give different $L_s$ and $U_s$. Specifically, the Boolean process is not reversible: i.e., although $A = LU$, in general, $A_s \neq L_s \odot U_s$, where $\odot$ represents the Booleanization of matrix multiply. The matrix $A_s$ below is an example.

$$A_s = \begin{bmatrix} 1&0&1 \\ 1&1&0 \\ 0&0&1 \end{bmatrix} \qquad \begin{bmatrix} 1&& \\ 1&1& \\ 0&0&1 \end{bmatrix} \odot \begin{bmatrix} 1&0&1 \\ &1&1 \\ &&1 \end{bmatrix} = \begin{bmatrix} 1&0&1 \\ 1&1&1 \\ 0&0&1 \end{bmatrix} = L_s \odot U_s.$$

Not all algorithms can be Booleanized. In particular, an algorithm which tests numerical data and makes a decision based on this test (e.g., Gaussian elimination *with pivoting*) will not qualify. However, we take the attitude that the pivoting can be replaced by an initial rearrangement of rows and columns. In general, the Booleanized algorithm is restricted to computations on the matrix $A_s$, but it is *not* legitimate to rearrange $A_s$. We emphasize in the rest of this paper, that we are given $A_s$ with the arrangement of rows and columns fixed thereafter.

The very difficult question of what reordering of the rows and columns for the $s$-minimal algorithm will produce the sparsest output will not be attacked in this paper, although it is felt that some of our results may have some bearing on this question.

A simple consequence of non $s$-minimality is that if an algorithm $\alpha$ is not $s$-minimal, then there is an $i, j$ location for which computation is performed to compute something which is always zero; i.e., we always end up with exact (neglecting round-off) numerical cancellation in location $i, j$.

Certain well-known algorithms which are thought to be good or best, in some sense, are not $s$-minimal, and therefore, do unnecessary fill-in. The Crout algorithm as specified in (3.1) is not $s$-minimal, but as we will see, a certain modification of this algorithm is $s$-minimal. A counterexample for the Crout algorithm is

$$A = \begin{bmatrix} a & b & c & 0 \\ d & 0 & 0 & 0 \\ e & f & g & 0 \\ h & 0 & 0 & i \end{bmatrix}.$$

We compute by the Crout algorithm

$$(L)_{43} = -h \cdot \frac{c}{a} + \frac{bh}{a} \cdot \frac{c}{b} \equiv 0,$$

whereas $(L_s^c)_{43} = 1$. In particular, besides having to allocate additional storage, there are two unnecessary multiplies and one add required.

It is obvious that the Gauss-Jordan method is also not $s$-minimal. Another non $s$-minimal algorithm is the Householder algorithm for factoring a matrix $A$ into $A = QR$, where $R$ is upper triangular and $Q'Q = I$. A counterexample is

$$A = \begin{bmatrix} 0 & a & 0 \\ b & 0 & 0 \\ c & 0 & d \end{bmatrix}.$$

It turns out that $(R)_{23} \equiv 0$, but following the Householder algorithm as given in [59] we find that $(R_s^H)_{23} = 1$.

The Gram-Schmidt and the modified Gram-Schmidt algorithms [110] do not fail on this example and it is not known whether these are $s$-minimal. The Givens method [59] does fail, and, therefore, is not $s$-minimal.

Let $L_s^m$ and $U_s^m$ denote the minimal sparseness structure for the factorization of $A_s$. Thus, $(L_s^m)_{ij} = 1$ if and only if there exists a numerical realization of $A_s$ such that $(L)_{ij} \neq 0$.

We will assume in what follows, that $A_{sii}$ has property $R$ for $1 \leqq i \leqq n$; otherwise, the factorization cannot exist.

THEOREM 3. *For* $i \geqq j$ $(i < j)$, $(L_s^m)_{ij} = 1$ $((U_s^m)_{ij} = 1)$ *if and only if* $A_{sij}$ *has property* $R$.

*Proof.* The proof depends on the following lemmas. Lemmas 1 and 2 can be found in Householder [44].

LEMMA 1. *For* $i \geqq j$, $(L)_{ij} = (\det A_{ij})/\det A_{j-1,j-1}$.

LEMMA 2. *For* $i < j$, $(U)_{ij} = (\det A_{ij})/\det A_{ii}$.

LEMMA 3. *Let $A_s$ be given. Then there exists a matrix $A$ representing $A_s$ such that for all square submatrices $B_s$ of $A_s$ with property $R$,* $\det B \neq 0$, *where $B$ is the corresponding submatrix of $A$.*

*Proof.* For each square submatrix $B$ of $A$, $\det B$ is a polynomial of the nonzero entries of $A$. Let these entries be denoted by $x = (x_1, x_2, \cdots, x_n)$, where $n$ is the number of nonzero entries in $A_s$. For each $B_s$ with property $R$ we have that

$$\det B = p_{B_s}(x)$$

is a nonconstant polynomial in $x$ since by property $R$, $\det B$ has at least one nonvanishing term. Since nonconstant polynomials vanish on a set of measure zero, and since there are only a finite number of square submatrices of $A$, then the set

$$S = \bigcup_{B_s} \{x;\ P_{B_s}(x) = 0\},$$

where the union is taken over each submatrix $B_s$ of $A_s$ with property $R$, is a set of measure zero. Hence, there must exist a point $x^*$ such that $p_{B_s}(x^*) \neq 0$ for all $B_s$, $B_s \subseteq A_s$ and $B_s$ has property $R$. The point $x^*$ gives us the matrix $A^*$ with the required property. Q.E.D.

We suppose that $i \geqq j$ in the remainder of the proof of Theorem 3. The case $i < j$ is very similar and will not be given.

Now suppose $A_{sij}$ has property $R$. Then by Lemma 3 there exists a numerical representation $A^*$ of $A_s$ such that $\det A_{ij}^* \neq 0$ and $\det A_{j-1,j-1}^* \neq 0$. Hence, by Lemma 1, $(L^*)_{ij} \neq 0$. Thus $(L_s^m)_{ij} = 1$.

Now assume that $(L_s^m)_{ij} = 1$. By definition, there must exist a numerical representation of $A_s$ such that $(L)_{ij} \neq 0$. This implies that $\det A_{j-1,j-1} \neq 0$ also. However, if $A_{ij}$ does not have property $R$, then $\det A_{ij} = 0$. Therefore, by Lemma 1, we have a contradiction and $A_{ij}$ must have property $R$. Q.E.D.

Theorem 3 immediately leads us to $s$-minimal algorithms for the triangularization of $A$ into $LU$. Namely, we take any algorithm $\alpha$ which computes $L$, $U$ and before we attempt to compute $(L)_{ij}$ or $(U)_{ij}$, we test if $A_{sij}$ has property $R$. Note that this is not a test on the numerical data, but only a test on $A_s$ and as such is an algorithm, which has a Boolean counterpart. The testing for property $R$ is the assignment problem for which there are known algorithms [111]–[114].

THEOREM 4. *Any algorithm $\alpha + R$ which computes $L$, $U$, and tests for property $R$ is s-minimal.*

*Proof.* In the Booleanized algorithm $\alpha + R$, we compute the matrices $L_s^\alpha$, $U_s^\alpha$ and for each "one" entry, say $(L_s^\alpha)_{ij} = 1$, as it is generated by the Booleanized $\alpha$ algorithm, we test if the matrix $A_{sij}$ has property $R$. If it does not, we change $(L_s^\alpha)_{ij}$ to zero and proceed. Therefore, $(L_s^{\alpha+R})_{ij} = 1$ only if $A_{sij}$ has property $R$. Conversely, if $A_{sij}$ has property $R$, then $(L_s^{\alpha+R})_{ij} = (L_s^\alpha)_{ij} = 1$. Using Theorem 3, we conclude that $L_s^{\alpha+R} = L_s^m$, $U_s^{\alpha+R} = U_s^m$. Q.E.D.

It turns out that if the diagonal entries of $A_s$ are 1, a case often met in practice, then it is not necessary to include the test for property $R$ with the Crout algorithm; i.e., the Crout algorithm is $s$-minimal in this case.

THEOREM 5. *Suppose $i \geqq j$ $(i < j)$ and $(A_s)_{\mu\mu} = 1$ for $1 \leqq \mu \leqq j - 1$ $(1 \leqq \mu \leqq i - 1)$. Then $A_{sij}$ has property $R$ if and only if $(L_s^c)_{ij} = 1$ $((U_s^c)_{ij} = 1)$.*

*Proof.* Suppose $A_{sij}$ has property $R$. Using Theorem 3 and the fact that $L_s^m \subseteq L_s^c$, it follows that $(L_s^c)_{ij}$ must be one. The proof for $U$ is similar and omitted.

Conversely, suppose that $(U_s^c)_{ij} = 1$. The proof for $L$ is similar and will be omitted. We will show that there exists a numerical representation of $A_s$ such that $(U)_{ij} \not\equiv 0$ and, therefore, by Theorem 3, $A_{sij}$ must have property $R$.

Consider the Crout formulae

$$l_{ij} = a_{ij} - \sum_{\mu=1}^{j-1} l_{i\mu} u_{\mu j}, \qquad i \geqq j,$$

$$u_{ij} = \left(a_{ij} - \sum_{\mu=1}^{i-1} l_{i\mu} u_{\mu j}\right) \Big/ l_{ii}, \qquad i < j.$$

*Case* 1. $(A_s)_{ij} = 1$. Then we can perturb the value of $a_{ij}$ so that $u_{ij}$ must be nonzero, since the other entries in the formula for $u_{ij}$ do not depend on $a_{ij}$.

*Case* 2. $(A_s)_{ij} = 0$. Choose the largest value of $\mu$, say $\mu^*$, in the formula for $u_{ij}$ such that $(L_s^c)_{i\mu^*} = (U_s^c)_{\mu^* j} = 1$. There must be such an index; otherwise, $(U_s^c)_{ij} = 0$. We proceed by induction. Assume that $u_{ij} \equiv 0$, but that for $1 \leqq \mu < i$ if $(U_s^c)_{\mu j} = 1$, then $A_{s\mu j}$ has property $R$, and if $(L_s^c)_{i\mu} = 1$, then $A_{si\mu}$ has property $R$. By Lemma 3, this means we must have a numerical representation for which $l_{i\mu^*}$ and $u_{\mu^* j}$ are both nonzero. We note that $l_{i\mu^*}$ and $\tilde{u}_{\mu^* j}$ (where $u_{\mu^* j} = \tilde{u}_{\mu^* j}/l_{\mu^*\mu^*}$) are independent of $a_{\mu^*\mu^*}$. Therefore, let $a_{\mu^*\mu^*}$ become large. (We use the fact that $(A_s)_{\mu^*\mu^*} = 1$.) We have by assumption

$$u_{ij} = l_{i\mu^*} u_{\mu^* j} + \Sigma \equiv 0.$$

But since $l_{i\mu^*} u_{\mu^* j}$ can be made arbitrarily small, by making $a_{\mu^*\mu^*}$ arbitrarily large and since $\Sigma$ is independent of $a_{\mu^*\mu^*}$, it must be that $\Sigma \equiv 0$ and, therefore, $l_{i\mu^*} u_{\mu^* j} \equiv 0$. This is a contradiction and, hence, $u_{ij} \not\equiv 0$. By Theorem 3 this implies that $A_{ij}$ has property $R$. Q.E.D.

COROLLARY. *If* $(A_s)_{ii} = 1$ *for* $1 \leqq i \leqq n - 1$, *then the Crout algorithm is s-minimal.*

It is not true in general that if $A_{ik}$ has property $R$, then $(U^{-1})_{ik} \not\equiv 0$, as the following counterexample shows. Consider

$$A = \begin{bmatrix} a & b & c & d \\ e & 0 & f & 0 \\ 0 & g & 0 & 0 \\ i & 0 & 0 & j \end{bmatrix}.$$

Then clearly $A_{24}$ has property $R$, and we compute $(U)_{24} = d/b$, but

$$(U^{-1})_{24} = -\frac{d}{b} - \left(\frac{f - ec/a}{-be/a}\right)\left(\frac{(gd/a)(-be/a)}{-g(f - ec/a)}\right) \equiv 0.$$

Since $U_s^m$ is full, this example shows that in special cases $(U^{-1})_s^m \subseteq U_s^m$. In fact, if $(A_s)_{23} = 0$, then $(U^{-1})_{14} \equiv 0$ also, but $U$ is still full.

However, we can characterize the fill-in of $U^{-1}$ directly in terms of determinants of the original matrix $A$ just as we did for the fill-in of $L$ and $U$. We do not know if the following lemma is known.

LEMMA 4. $(U^{-1})_{ik} = -\det \tilde{A}_{ik}/\det A_{k-1,k-1}$, *where* $i < k$.

*Proof.* From Section II we know the $k$th column of $T_{k-1}^{-1} \cdots T_1^{-1} A$ is the nontrivial column of $T_k$. Letting $w_k = \operatorname{col}(t_{1k}, \cdots, t_{k-1,k}, 0, \cdots, 0)$, we saw that $W_k \equiv$

$I + w_k e_k^t$ had the property that the nontrivial column of $W_k^{-1}$ was exactly the $k$th column of $U^{-1}$. Thus, $(U^{-1})_{ik} = -t_{ik}$ for $i < k$.

On the other hand,

$$T_{k-1}^{-1} \cdots T_1^{-1} = \begin{bmatrix} A_{k-1,k-1}^{-1} & 0 \\ B & C \end{bmatrix},$$

where $B$ and $C$ are some matrices of the appropriate size. Thus, the first $k - 1$ components of the $k$th column of $T_{k-1}^{-1} \cdots T_1^{-1} A$ is $A_{k-1,k-1}^{-1} \hat{a}_k$, where $\hat{a}_k = \operatorname{col}(a_{1k}, \cdots, a_{k-1,k})$. Using Cramer's rule, we have that $t_{ik} = \det A_{ik} / \det A_{k-1,k-1}$. Q.E.D.

Therefore, we have the following theorem.

THEOREM 6. $(U^{-1})_{ik} \not\equiv 0$ *if and only if* $\tilde{A}_{sik}$ *has property R.*

Since the proof follows directly from the lemma and is similar to the proof of Theorem 3, it will be omitted.

If the diagonal elements of $A$ are nonzero, we have a complete characterization of the fill-in of $U^{-1}$.

Denote by $\alpha^*$ the algorithm $A \xrightarrow{\text{Crout}} L\backslash U \rightarrow U^{-1}$, where $U^{-1}$ is computed from $U$ by the formula (3.4) below.

THEOREM 7. *If* $(A_s)_{ii} = 1$ for $1 \leqq i \leqq n$, *then* $\alpha^*$ *is s-minimal.*

*Proof.* For convenience, let $v_{ij}$ denote $(U^{-1})_{ij}$ and $V$ denote $U^{-1}$. Then we compute $V$ from $U$ by

$$v_{ik} = -u_{ik} - \sum_{\mu=i+1}^{k-1} v_{i\mu} u_{\mu k}. \tag{3.4}$$

Suppose that $(V_s^{\alpha^*})_{ik} = 1$. We need to show that there exists a numerical representation of $A_s$ such that $v_{ik} \neq 0$. Let $\mu^*$ denote the smallest row index of $u$ occurring in (3.4) for which $(U_s^c)_{\mu^* k} = (V_s^{\alpha^*})_{i\mu^*} = 1$. There must be at least one such $\mu$ value since $(V_s^{\alpha^*})_{ik} = 1$. It may be that $\mu^* = i$. In that case, $v_{i\mu^*} = 1$ in the following.

We proceed by induction. Assume that for arbitrary $l$ and $1 \leqq j < k$, $(V_s^{\alpha^*})_{lj} = 1$ implies $v_{lj} \not\equiv 0$. By Theorem 5 there must exist a numerical representation of $A_s$ such that $u_{\mu^* k} \neq 0$. If for this numerical representation, $v_{i\mu^*} = 0$, then using Lemmas 3 and 4, we can perturb the data so that both $v_{i\mu^*}$ and $u_{\mu^* k}$ are nonzero. We note that both $v_{i\mu^*}$ and $u_{\mu^* k}$ are independent of $a_{\mu\mu}$ for $\mu > \mu^*$.

If $v_{ik} \equiv 0$, then from (3.4)

$$u_{\mu^* k} v_{i\mu^*} + \Sigma \equiv 0$$

and we have $\Sigma = -u_{\mu^* k} v_{i\mu^*} \neq 0$. However, by making $a_{\mu\mu}$ large for $\mu > \mu^*$, each $v$ and $u$ term in $\Sigma$ must approach zero and since $u_{\mu^* k} v_{i\mu^*}$ remains constant, we have a contradiction. Therefore, $v_{ik} \not\equiv 0$. Q.E.D.

THEOREM 8. $U_s^{\text{EFI}} \subseteq (U^{-1})_s^{\text{PFI}}$.

*Proof.* From Section II we know that the PFI or Gauss-Jordan algorithm is the Boolean equivalent of $A \xrightarrow{\text{GE}} L\backslash U \xrightarrow{(3.4)} U^{-1}$, where GE denotes any form of Gaussian elimination. The only difference in these algorithms is the sequence in which operations are carried out. This, however, does not affect the Boolean output of the algorithm. Since from (3.4) we have $(U_s^{\text{GE}})_{ik} = 1$ implies $((U^{-1})_s^{\alpha^*})_{ik} = 1$, we conclude that $U_s^{EFI} \subseteq (U^{-1})_s^{PFI}$.

COROLLARY. *If* $(A_s)_{ii} = 1$ *for* $1 \leqq i \leqq n$, *then the EFI and PFI algorithms are s-minimal for computing* $L\backslash U$ *and* $L\backslash U^{-1}$, *respectively.*

This result can be rephrased, in that if the diagonal elements of $A$ are nonzero, then there is no method, known or unknown, of solving linear equations which computes $L$, $U^{-1}$ and which requires less storage than Gaussian elimination.

Theorems 3 and 6 give a partial explanation of the occurrence in the EFI and PFI of very small nonzero quantities obtained in computing the forms of the inverse. In some instances these are zeros identically, but because of round-off error, appear as nonzeros. We see that this can happen if the diagonal elements of $A$ are not all nonzero; i.e., this phenomena can occur if we choose some pivot positions corresponding to zero locations of the original matrix $A$. In some codes, a threshold test is applied to every nonzero quantity, and if it is too small in absolute value, it is set to zero. Such a threshold test is not so necessary if it is known that the pivot choices correspond to nonzero elements of $A$, since "logical" cancellations cannot occur.

We have thus far been concerned with sparseness only and not the arithmetic operations count which might be another quantity to be minimized by an algorithm. According to Definition 6 an algorithm is *o-minimal* if it requires the least number of arithmetic operations (counting both addition and multiplication equally) of all algorithms which compute the same thing. Clearly, the Crout algorithm is not the $o$-minimal algorithm for computing $L\backslash U$. The question of $o$-minimality is not answered even in the full matrix case although some results on the algorithm requiring the least number of multiplications have been obtained [115]–[116]. For sparse matrices, no results are available and we pose the following question: Under what conditions is a Crout algorithm locally $o$-minimal in the sense that the Crout method for computing the next element in the factorization requires the minimal number of arithmetic operations? Our conjecture is that the Crout algorithm is locally $o$-minimal if $(A_s)_{ii} = 1$ for $1 \leqq i \leqq n$. This condition can be shown to be necessary by counterexample. We will not pursue this question further in this paper.

**IV. A Comparison of EFI and PFI for Randomly Generated Matrices.** In Section III we demonstrated that the factored form $U$ determined by the EFI method never had more nonzero elements of type (a) than the factored form of $U^{-1}$. This result says nothing about how much fuller $U^{-1}$ is than $U$. In fact, if a matrix is tridiagonal, then $U^{-1}$ is a full upper triangular matrix while $U$ is co-diagonal; whereas, for the matrix

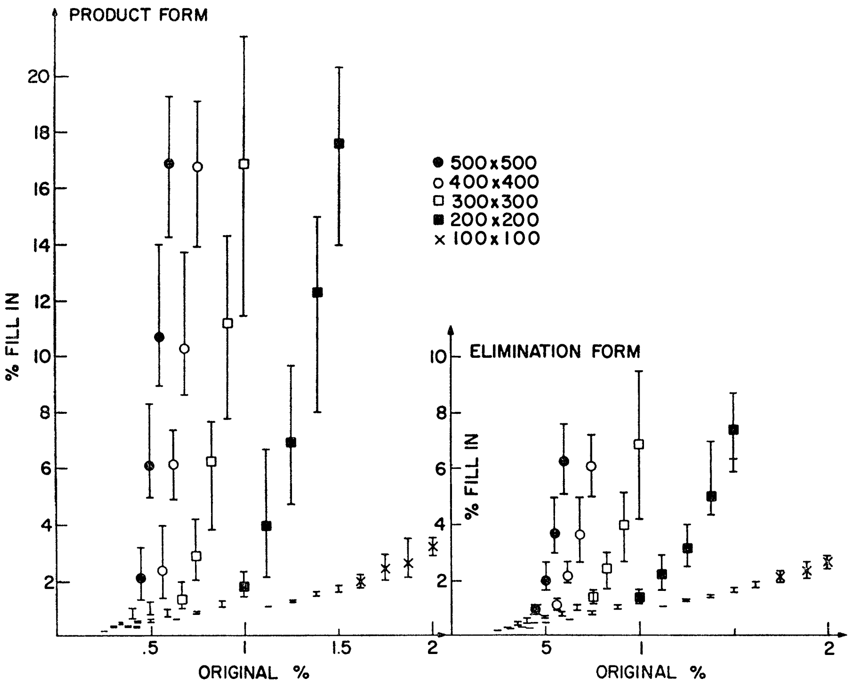
$$A = \begin{bmatrix} X & 0 & X & 0 & X & 0 \\ X & X & 0 & 0 & 0 & 0 \\ 0 & 0 & X & 0 & 0 & 0 \\ X & 0 & 0 & X & 0 & 0 \\ 0 & X & 0 & 0 & X & 0 \\ 0 & 0 & 0 & X & 0 & X \end{bmatrix}$$

$U^{-1}$ and $U$ have the same number of nonzero elements. To obtain some idea of the fill-in characteristics of these two algorithms, we have performed an experiment on a set $S$ of matrices whose elements are generated randomly.

A matrix $A \in S$ is characterized by two parameters $n$ and $p$ as follows. $A$ has order $n$, and nonzero dominant diagonal elements. The other $n^2 - n$ elements of $A$ are chosen nonzero with probability $p$. The numerical value of each nonzero off-diagonal element of $A$ is chosen as a random floating point number between $-1$ and $+1$. The set $S$ consists of five groups of eighty matrices each. In each group, the order is fixed: $n = 100, 200, 300, 400$, and $500$ for the five groups. Within each group, ten matrices of equal density are generated for each of eight increasing values of $p$. The experiment consists of solving $Ax = b$ for each $A \in S$, both by the EFI and PFI method, and noting the amount of fill-in and the execution times for the factorization and back-substitution. All calculations were carried out on an IBM model 360/67 computer.

The computer programs that we designed for the EFI and PFI method are typical of sparse matrix codes and are similar to that described by Tinney and Walker [63].

The results of the experiment are displayed in Figure 1. In both graphs we have plotted percent of fill-in vs. original percent. There are five sets of data points plotted in each graph. For a given set (value of $n$), we have plotted for each of eight abscissa values, the maximum, average and minimum values of fill-in (in percent). It appears from the similarity of each of the five graphs, that each graph is described by a single function. To see this, let $y = f(x)$ represent one of the five curves. Then, to rough approximation, any other curve is obtained as $y = f(\sigma x)$, $\sigma$ a constant. Closer scrutiny reveals that $\sigma$ is proportional to $1/n$, so that, since the abscissa is proportional to $1/n^2$, percent of fill-in is independent of $n$ for constant values of $n'/n^3$, where $n'$ is the number of nonzeros in the original matrix.



*Fill in Comparison of PFI and EFI*

An important result of this experiment is that randomly generated matrices fill in very fast. The graphs in Figure 1 reveal a "take off" in fill-in at about 2 nonzero elements per column. This sharp rise in fill-in continues in both the EFI and PFI at least until the 50 percent fill-in mark is reached. The EFI rate is about one-half the PFI rate. This means that the rate of fill-in for $U$ is one-third that for $U^{-1}$.

The execution time for factorization of the PFI ranges from 1.5 to 5 times that of EFI. For back-substitution, the range is from .7 to 2.5. Since, for full matrices the ratio of factorization time is $\frac{1}{2}n^3$ to $\frac{1}{3}n^3$ or 1.5, we see that the gain is even greater for sparse matrices. The higher time ratios occur for the matrices with the larger fill-in. Back-substitution time, except for indexing, is proportional to the number of nonzero elements in the factorization of $A$. The explanation for a time ratio less than 1 is due to the fact that in the EFI we index from 1 to $n$ twice ($Ly = b$ and $Ux = y$) whereas we index only once in the PFI. This additional indexing shows up for very sparse matrices and becomes a nontrivial part of the total time for back-substitution.

The experiment points out the importance of a practical reordering scheme. One should, before factorization, reorder the rows and columns of $A$ to reduce fill-in. At present, there exist no algorithms which give an ordering of rows and columns which produces minimum fill-in (or minimum operation count) for either the elimination form or the product form. Even if such an algorithm existed, it might not be practical because of its complexity. However, some simple procedures do exist, and these give remarkable reductions in fill-in [63], [72].

One should not conclude that the EFI is twice as good as the PFI. Our experiment is unfair in that a reordering algorithm did not precede both the EFI and PFI cases. Nevertheless, the results of Sections II and III show that the application of the *elimination* form to the best PFI ordering will result in less fill-in than given by the *product* form. Thus, any ordering that is good for the PFI is also good for the EFI. However, no such claim can be made for the PFI. Also, it appears that the best strategies for EFI and PFI will differ since the EFI produces a symmetric factorization ($L$ and $U$) and the PFI an unsymmetric one ($L$ and $U^{-1}$).

**Acknowledgment.** The authors would like to thank the referee for carefully reading the manuscript and for suggesting ways to shorten and clarify certain parts.

IBM Watson Research Center
Yorktown Heights, New York 10598

1. R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, N. J., 1962. MR 28 #1725.

2. T. Koopmans (Editor), *Activity Analysis of Production and Allocation*, Cowles Commission Monograph, no. 13, Wiley, New York and Chapman & Hall, London, 1951. MR 13, 670.

3. V. Riley & R. L. Allen, *Interindustry Economic Studies*, Johns Hopkins Press, Baltimore, Md., 1955.

4. *Linear Inequalities and Related Systems*, Ann. of Math. Studies, no. 38, Princeton Univ. Press, Princeton, N. J., 1956.

5. S. I. Gass, *Linear Programming: Methods and Applications*, McGraw-Hill, New York, 1958. MR 20 #3037.

6. V. Riley & S. I. Gass, *Linear Programming and Associated Techniques*, Bibliographic Reference Series, no. 5, Johns Hopkins Press, Baltimore, Md., 1958. MR 19, 1197.

→ G. B. Dantzig & P. Wolfe, "Decomposition principle for linear programs," *Operations Res.*, v. 8, 1960, pp. 101–111.

8. G. B. Dantzig & P. Wolfe, "The decomposition algorithm for linear programs," *Econometrica*, v. 29, 1961, pp. 767–778. MR 25 #1953.
9. G. B. Dantzig, *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, N. J., 1963.
10. R. L. Graves & P. Wolfe (Editors), *Recent Advances in Mathematical Programming*, McGraw-Hill, New York, 1963.
11. P. Wolfe & L. Cutler, *Experiments in Linear Programming*, Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963, pp. 177–200.
12. D. M. Smith & W. Orchard-Hays, *Computational Efficiency in Product Form LP Codes*, Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963, pp. 211–218.
13. M. Simonnard, *Linear Programming*, Prentice-Hall, Englewood Cliffs, N. J., 1966.
14. W. Orchard-Hays, *Advanced Linear Programming Computing Techniques*, McGraw-Hill, New York, 1968.
15. R. L. Weil, Jr., "The decomposition of economic production systems," *Econometrica*, v. 36, 1968, pp. 260–278.
16. G. B. Dantzig & W. Orchard-Hays, "The product form of the inverse in the simplex method," *MTAC*, v. 8, 1954, pp. 64–67. MR 15, 831.
17. L. J. Larson, "A modified inversion procedure for product form of inverse in linear programming codes," *Comm. ACM*, v. 5, 1962, pp. 382–383.
18. R. P. Tewarson, "On the product form of inverses of sparse matrices," *SIAM Rev.*, v. 8, 1966, pp. 336–342. MR 34 #8631.
19. R. P. Tewarson, "On the product form of inverses of sparse matrices and graph theory," *SIAM Rev.*, v. 9, 1967, pp. 91–99. MR 36 #1092.
→ H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Sci.*, v. 3, 1957, pp. 255–269. MR 22 #3098.
21. G. B. Dantzig, *Compact Basis Triangularization for the Simplex Method*, Recent Advances in Mathematical Programming, McGraw-Hill, New York, 1963, pp. 125–132. MR 32 #1010.
22. A. M. Turing, "Rounding-off errors in matrix processes," *Quart. J. Mech. Appl. Math.*, v. 1, 1948, pp. 287–308. MR 10, 405.
23. L. J. Paige & O. Taussky (Editors), "Simultaneous linear equations and the determination of eigenvalues," *Nat. Bur. Standards Appl. Math. Ser.*, v. 29, 1953.
24. G. E. Forsythe & E. G. Straus, "On best conditioned matrices," *Proc. Amer. Math. Soc.*, v. 6, 1955, pp. 340–345. MR 16, 1054.
25. E. Bodewig, *Matrix Calculus*, North-Holland, Amsterdam, 1956. MR 18, 235.
26. E. Kosko, "Matrix inversion by partitioning," *Aero. Quart.*, v. 8, 1957, pp. 157–184. MR 19, 769.
→ A. S. Householder, "A survey of some closed methods for inverting matrices," *J. Soc. Indust. Appl. Math.*, v. 5, 1957, pp. 155–169. MR 19, 982.
28. L. B. Wilson, "Solution of certain large sets of equations on Pegasus using matrix methods," *Comput.* J., v. 2, 1959, pp. 130–133. MR 21 #6084.
29. E. E. Osborne, "On pre-conditioning of matrices," *J. Assoc. Comput. Mach.*, v. 7, 1960, pp. 338–345. MR 26 #892.
30. G. E. Forsythe, "Crout with pivoting," *Comm. ACM*, v. 3, 1960, p. 507.
31. A. Orden, *Matrix Inversion and Related Topics by Direct Methods*, Mathematical Methods for Digital Computers, vol. 1, Wiley, New York, 1960, pp. 39–55. MR 22 #8682.
32. D. L. Elliott, "A note on systems of linear equations," *SIAM Rev.*, v. 3, 1961, pp. 66–69. MR 22 #12702.
33. S. Parter, "The use of linear graphs in Gauss elimination," *SIAM Rev.*, v. 3, 1961, pp. 119–130. MR 26 #908.
34. J. H. Wilkinson, "Error analysis of direct methods of matrix inversion," *J. Assoc. Comput. Mach.*, v. 8, 1961, pp. 281–330. MR 31 #874.
35. F. Harary, "A graph theoretic approach to matrix inversion by partitioning," *Numer. Math.*, v. 4, 1962, pp. 128–135. MR 25 #2977.
36. D. R. Fulkerson & P. Wolfe, "An algorithm for scaling matrices," *SIAM Rev.*, v. 4, 1962, pp. 142–146. MR 25 #1039.
37. A. L. Dulmage & N. S. Mendelsohn, "On the inversion of sparse matrices," *Math. Comp.*, v. 16, 1962, pp. 494–496. MR 27 #6375.
→ W. M. McKeeman, "Crout with equilibration and iteration," *Comm. ACM*, v. 5, 1962, pp. 553–555.
39. F. L. Bauer, "Optimally scaled matrices," *Numer. Math.*, v. 5, 1963, pp. 73087. MR 28 #2629.
40. D. K. Faddeev & V. N. Faddeeva, *Computational Methods in Linear Algebra*, Fizmatgiz, Moscow, 1960; English transl. of 1st ed., Freeman, San Francisco, Calif., 1963. MR 28 #1742; MR 28 #4659.
→ A. L. Dulmage & N. S. Mendelsohn, "Two algorithms for bipartite graphs," *J. Soc. Indust. Appl. Math.*, v. 11, 1963, pp. 183–194. MR 27 #4224.
42. J. Carpentier, "*Éliminations ordonnées*," *un processus diminuant le volume des calculs dans la résolutions des systèmes linéaires à matrice creuse*, Troisième Congr. de Calcul et de Traitement de l'Information AFCALTI, Dunod, Paris, 1965, pp. 63–71. MR 32 #1896.
43. L. Fox, *An Introduction to Numerical Linear Algebra*, Clarendon Press, Oxford, 1964. MR 29 #1733.

44. A. S. Householder, *The Theory of Matrices in Numerical Analysis*, Blaisdell, Waltham, Mass., 1964. MR 30 #5475.

→ W.-K. Chen, "The inversion of matrices by flow graphs," *J. Soc. Indust. Appl. Math.*, v. 12, 1964, pp. 676–685. MR 30 #2023.

46. F. H. Branin, Jr., et al., "An interpretive program for matrix arithmetic," *IBM Systems J.*, v. 4, 1965, pp. 2–24.

47. J. C. Dickson, *Finding Permutation Operations to Produce a Large Triangular Sub-Matrix*, Twenty-eighth National Meeting Operations Research Society of America, Houston, Texas, 1965.

48. B. H. Mayoh, "A graph technique for inverting certain matrices," *Math. Comp.*, v. 19, 1965, pp. 644–646. MR 33 #5108.

49. W. Oettlie, W. Prager & J. H. Wilkinson, "Admissible solutions of linear systems with not sharply defined coefficients," *J. Soc. Indust. Appl. Math. Ser. B. Numer. Anal.*, v. 2, 1965, pp. 291–299. MR 32 #1888.

50. D. Bree, Jr., *Some Remarks on the Application of Graph Theory to the Solution of Sparse Systems of Linear Equations*, Thesis, Dept. of Mathematics, Princeton University, Princeton, N. J., 1965.

51. H. Edelmann, *Ordered Triangular Factorization of Matrices*, Proc. Power System Computation Conference, Stockholm, 1966.

52. G. G. Alway & D. W. Martin, "An algorithm for reducing the bandwidth of a matrix of symmetrical configuration," *Comput. J.*, v. 8, 1965/66, pp. 264–272.

53. W. Kahan, "Numerical linear algebra," *Canad. Math. Bull.*, v. 9, 1966, pp. 757–801.

54. A. Jennings, "A compact storage scheme for the solution of symmetric linear simultaneous equations," *Comput. J.*, v. 9, 1966/67, pp. 281–285.

→ H. J. Bowdler, R. S. Martin, G. Peters & J. H. Wilkinson, "Solution of real and complex systems of linear equations," *Numer. Math.*, v. 8, 1966, pp. 217–234.

56. A. M. Hershdorfer, et al., *On the Efficient Inversion of Large Structured Matrices*, Proc. ASCE Engineering Mechanics Division Specialty Conference, Washington, D. C., 1966.

57. B. A. Chartres & J. C. Geuder, "Computable error bounds for direct solution of linear equations," *J. Assoc. Comput. Mach.*, v. 14, 1967, pp. 63–71. MR 35 #6390.

58. G. Forsythe & C. B. Moler, *Computer Solution of Linear Algebraic Equations*, Prentice-Hall, Englewood Cliffs, N. J., 1967. MR 36 #2306.

59 J. H. Wilkinson, *The Solution of Ill-Conditioned Linear Equations*, Mathematical Methods for Digital Computers, vol. 2, Wiley, New York, 1967, pp. 65–93.

60. R. P. Tewarson, "Solution of a system of simultaneous linear equations with a sparse coefficient matrix by elimination methods," *Nordisk Tidskr. Informations-Behandling*, v. 7, 1967, pp. 226–239. MR 36 #2308.

61. F. Harary, "Graphs and matrices," *SIAM Rev.*, v. 9, 1967, pp. 83–90. MR 35 #1501.

62. G. E. Forsythe, "Today's computational methods of linear algebra," *SIAM Rev.*, v. 9, 1967, pp. 489–515. MR 36 #1089.

63. W. F. Tinney & J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proc. IEEE*, v. 55, 1967, pp. 1801–1809.

64. A. Nathan & R. K. Even, "The inversion of sparse matrices by a strategy derived from their graphs," *Comput. J.*, v. 10, 1967, pp. 190–194. MR 35 #5128.

65. R. P. Tewarson, "Row-column permutation of sparse matrices," *Comput. J.*, v. 10, 1967, pp. 300–305. MR 36 #1091.

66. J. R. Westlake, *A Handbook of Numerical Matrix Inversion and Solution of Linear Systems*, Wiley, New York, 1968. MR 36 #4794.

67. R. P. Tewarson, "Solution of linear equations with coefficient matrix in band from" *Nordisk Tidskr. Informations-Behandling*, v. 8, 1968, pp. 53–58. MR 37 #2425.

68. W. R. Spillers & N. Hickerson, "Optimal elimination for sparse symmetric systems as a graph problem," *Quart. Appl. Math.*, v. 26, 1968, pp. 425–432. MR 38 #1818.

69. G. Loizou, "An empirical estimate of the relative error of the computed solution $\bar{x}$ of $Ax = b$," *Comput. J.*, v. 11, 1968/69, pp. 91–94. MR 37 #1067.

70. Richard Rosen, *Matrix Band Width Minimization*, ACM National Conference, Las Vegas, Nevada, 1968.

71. *System/*360 *Matrix Language* (MATLAN), Application Description, IBM H20-0479; Program Description Manual, IBM H20-0564; Operations Manual, IBM H20-0559; System Manual, IBM Y20-0261.

72. G. B. Dantzig, R. P. Harvey, R. D. McKnight & S. S. Smith, *Sparse Matrix Techniques in Two Mathematical Programming Codes*, Proc. Sparse Matrix Sympos., IBM Watson Research Center, Yorktown Heights, New York, 1968.

73. G. Kron, *Tensor Analysis of Networks*, Wiley, New York, 1939.

74. P. M. Hunt, "The electronic digital computer in aircraft structural analysis. The programming of the Argyris matrix formulation of structural theory for an electronic digital computer. I: A description of a matrix interpretive scheme and its application to a particular example," *Aircraft Engrg.*, v. 28, 1956, pp. 70–76. MR 17, 902.

75. J. B. Ward & H. W. Hale, "Digital computer solution of power flow problems," *Trans. AIEE*, v. PAS-75, 1956, part III, pp. 398–404.

76. R. J. Brown & W. F. Tinney, "Digital solutions for large power networks," *Trans. AIEE*, v. PAS-76, 1957, part III, p. 347.

77. J. E. Van Ness, "Iteration methods for digital load flow studies," *Trans. AIEE*, v. PAS-78A, 1959, part III, pp. 583–588.

78. R. Livesley, "The analysis of large structural systems," *Comput. J.*, v. 3, 1960/61, pp. 34–39. MR 22 #3195.

79. C. H. Norris & J. B. Wilbur, *Elementary Structural Analysis*, 2nd ed., McGraw-Hill, New York, 1960.

80. J. E. Van Ness, "Convergence of iterative load-flow studies," *Trans. AIEE*, v. PAS-79, 1960, part III, pp. 1590–1597.

81. A. S. Hall & R. W. Woodhead, *Frame Analysis*, Wiley, New York, 1961.

82. J. E. Van Ness & J. H. Griffin, "Elimination methods for loadflow studies," *Trans. AIEE*, v. PAS-80, 1961, part III, pp. 299–304.

83. F. H. Branin, Jr., D-C *and Transient Analysis of Networks Using a Digital Computer*, IRE Internat. Conference Rec., part 2, 1962, pp. 236–256.

84. H. E. Brown, G. K. Carter, H. H. Happ & C. E. Person, "Power flow solution by matrix iterative method," *Trans. AIEE*, v. PAS-82, 1963, part III, p. 1.

85. H. E. Brown, H. H. Happ, G. K. Carter & C. E. Person, "Power flow solution by impedance matrix iterative method," *Trans. AIEE*, v. PAS-82, 1963, pp. 561–564.

86. R. W. Clough, E. L. Wilson & I. P. King, "Large capacity multistory frame analysis programs," *ASCE J. Structural Division*, v. 89, 1963, p. 179.

87. S. J. Fenves & F. Branin, "A network-topological formulation of structural analysis," *ASCE J. Structural Division*, v. 89, 1963, pp. 483–514.

88. S. J. Fenves, *STRESS* (*Structural Engineering System Solver*) *A Computer Programming System for Structural Engineering Problems*, M.I.T. Technical Report T63-2, 1963.

89. G. Kron, *Diakoptics*, Macdonald, London, 1963.

90. W. R. Spillers, "Network analogy for linear structures," *Proc. ASCE J. Engrg. Mech. Division*, v. 89, 1963, no. EM-4, pp. 21–29.

91. W. R. Spillers, "Applications of topology in structural analysis," *Proc. ASCE J. Structural Division*, v. 89, 1963, no. ST-4, pp. 301–313.

92. M. A. Laughton & M. W. H. Davies, "Numerical techniques in solution of power-system load-flow problems," *Proc. IEEE*, v. 111, 1964, p. 1575.

93. W. R. Spillers, "Network techniques applied to structures," *Matrix Tensor Quart.*, v. 15, 1964, pp. 31–41.

94. R. Baumann, "Some new aspects on load-flow calculation: I-impedance matrix generation controlled by network topology," *AIEE Trans.*, v. PAS-85, 1966, pp. 1164–1176.

95. F. H. Branin, Jr., *The Algebraic-Topological Basis for Network Analogies and the Vector Calculus*, Proc. Sympos. on Generalized Networks, vol. 16, Microwave Res. Inst. Sympos. Ser. Poly. Inst. of Brooklyn, 1963, pp. 453–491.

96. F. F. Kuo, "Network analysis by digital computer," *Proc. IEEE*, v. 54, 1966, pp. 820–829.

97. F. F. Kuo & J. F. Kaiser (Editors), *System Analysis by Digital Computer*, Wiley, New York, 1966.

98. F. H. Branin, Jr., "Computer methods of network analysis," *Proc. IEEE*, v. 55, 1967, pp. 1787–1801.

99. G. H. Jensen, "Efficient matrix techniques applied to transmission tower design," *Proc. IEEE*, v. 55, 1967, pp. 1997–2000.

100. G. H. Jensen, *Designing Self-Supporting Transmission Towers with the Digital Computer*, PICA Conference, 1967, pp. 303–319.

101. W. F. Tinney & C. E. Hart, "Power flow solution by Newton's method," *Trans. AIEE*, v. PAS-86, 1967, pp. 1449–1460.

102. A. T. Trihaus & R. Zimering, *A Digital Computer Program for an Exhaustive Fault Study of a Large Power System Network*, IEEE Power Industry Computer Applications Conference Rec., Pittsburgh, 1967, pp. 343–349.

103. W. Weaver, Jr., *Computer Programs for Structural Analysis*, Van Nostrand, Princeton, N. J., 1967.

104. O. C. Zienkiewicz, *The Finite Element Method in Structural and Continuum Mechanics*, McGraw-Hill, New York, 1967.

105. J. S. Przemieniecki, *Theory of Matrix Structural Analysis*, McGraw-Hill, New York, 1968.

106. W. R. Spillers, "Analysis of large structures: Kron's methods and more recent work," *Proc. ASCE* (To appear.)

107. G. W. Stagg & A. H. El-Abiad, *Computer Methods in Power System Analysis*, McGraw-Hill, New York, 1968.

108. N. Sato & W. F. Tinney, "Techniques for exploiting the sparsity of the network admittance matrix," *Trans. AIEE*, v. PAS-82, 1963, pp. 944–950.

109. F. Gustavson, W. Liniger & R. Willoughby, "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations," *J. Assoc. Comput. Mach.*, v. 17, 1970, pp. 87–109.

110. Å. Björck, "Solving linear least squares problems by Gram-Schmidt orthogonalization," *Nordisk Tidskr. Informations-Behandling*, v. 7, 1967, pp. 1–21. MR 35 #5126.

111. T. S. MOTZKIN, *The Assignment Problem*, Proc. Sympos. Appl. Math., vol. 6, Amer. Math Soc., Providence, R. I., 1956, pp. 109–125. MR 19, 822.
1 → J. MUNKRES, "Algorithms for the assignment and transportation problems," *J. Soc. Indust. Appl. Math.*, v. 5, 1957, pp.32–38. MR 19, 1244.
113. L. R. FORD, JR. & D. R. FULKERSON, *Flows in Networks*, Princeton Univ. Press, Princeton, N. J., 1962. MR 28 #2917.
1 → A. YASPAN, "On finding a maximal assignment," *Operations Res.*, v. 14, 1966, pp. 646–651.
115. V. V. KLYUYEV & N. I. KOKOVKIN-SHCHERBAK, *On the Minimization of the Number of Arithmetic Operations for the Solution of Linear Algebraic Systems*, Stanford Comput. Sci. Report CS-24, 1965.
116. S. WINOGRAD, *On the Number of Multiplications Necessary to Compute Certain Functions*, IBM Watson Research Center, RC 2285, 1968.