algebraic systems" (translated by G. J. Tee), Computer Science Dept., Stanford University, Stanford, Calif., Tech. Rept. CS24, 1965.
[29] N. Sato and W. F. Tinney, "Techniques for exploiting the sparsity of the network admittance matrix," *IEEE Trans. Power and Apparatus*, vol. 82, pp. 944–950, December 1963.
[30] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," this issue, p. 1801.
[31] S. Parter, "The use of linear graphs in gauss elimination," *SIAM Rev.*, vol. 3, pp. 119–130, April 1961.
[32] D. Bree, Jr., "Some remarks on the application of graph theory to the solution of sparse systems of linear equations," thesis, Dept. of Math., Princeton University, Princeton, N. J., May 1965.
[33] A. Orden, "Matrix inversion and related topics by direct methods," in *Mathematical Methods for Digital Computers*, A. Ralston and H. S. Wilf, Eds. New York: Wiley, 1960, ch. 2.
[34] F. B. Hildebrand, *Introduction to Numerical Analysis*. New York: McGraw-Hill, 1956.
[35] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Math. Comp.*, vol. 19, pp. 577–593, October 1965.
[36] F. H. Branin, Jr., and H. H. Wang, "A fast reliable iteration method for dc analysis of nonlinear networks," this issue, p. 1819.
[37] J. Katzenelson, "An algorithm for solving nonlinear resistive networks," *Bell Sys. Tech. J.*, vol. 44, pp. 1605–1620, November 1965.
[38] J. G. F. Francis, "The *Q-R* transformation, pt. I," *Computer J.*, vol. 4, pp. 265–271, October, 1961; "The *Q-R* transformation, pt. II," *Computer J.*, vol. 4, pp. 332–345, January 1962.
[39] M. L. Liou, "A numerical solution of linear time-invariant systems," *Proc. 3rd Allerton Conf. on Circuit and System Theory*, pp. 669–676, 1965.
[40] R. W. Stineman, "Digital time-domain analysis of systems with widely separated poles," *J. Assoc. Comp. Mach.*, vol. 12, pp. 286–293, April 1965.
[41] P. I. Richards, W. D. Lanning, and M. D. Torrey, "Numerical integration of large, highly-damped, nonlinear systems," *SIAM Rev.*, vol. 7, pp. 376–380, July 1965.
[42] C. E. Treanor, "A method for the numerical integration of coupled first-order differential equations with greatly different time constants," *Math. Comp.*, vol. 20, pp. 39–45, January 1966.
[43] E. R. Cohen, "Some topics in reactor kinetics," *Proc. 2nd UN Internat'l Conf. on Peaceful Uses of Atomic Energy* (Geneva, Switzerland), vol. 11, pp. 302–309, 1958.
[44] E. R. Cohen and H. P. Flatt, "Numerical solution of quasi-linear equations," *Proc. Seminar on Codes for Reactor Computations* (Vienna, Austria), pp. 461–484, 1960.
[45] J. Certaine, "The solution of ordinary differential equations with large time constants," in *Mathematical Methods for Digital Computers*, A. Ralston and H. S. Wilf, Eds. New York: Wiley, 1960, ch. 11.
[46] J. Katzenelson, "AEDNET: a simulator for nonlinear networks," *Proc. IEEE*, vol. 54, pp. 1536–1552, November 1966.
[47] G. N. Polozhii, *The Method of Summary Representation for Numerical Solution of Problems of Mathematical Physics*. New York: Pergamon, 1965.
[48] *System Analysis by Digital Computer*, F. F. Kuo and J. F. Kaiser, Eds. New York: Wiley, 1966.
[49] G. C. Temes and D. A. Calahan," Computer-aided network optimization—The state-of-the-art," this issue, p. 1832.
[50] G. D. Hachtel and R. A. Rehrer, "Techniques for the optimal design and synthesis of switching circuits," this issue, p. 1864.

Direct Solutions of Sparse Network Equations by Optimally Ordered Triangular Factorization

WILLIAM F. TINNEY, SENIOR MEMBER, IEEE, AND JOHN W. WALKER, MEMBER, IEEE

Abstract—**Matrix inversion is very inefficient for computing direct solutions of the large sparse systems of linear equations that arise in many network problems. Optimally ordered triangular factorization of sparse matrices is more efficient and offers other important computational advantages in some applications. With this method, direct solutions are computed from sparse matrix factors instead of from a full inverse matrix, thereby gaining a significant advantage in speed, computer memory requirements, and reduced round-off error. Improvements of ten to one or more in speed and problem size over present applications of the inverse can be achieved in many cases. Details of the method, numerical examples, and the results of a large problem are given.**

I. Introduction

USUALLY, the objective in the matrix analysis of networks is to obtain the inverse of the matrix of coefficients of a system of simultaneous linear network equations. However, for large sparse systems such as occur in many network problems, the use of the inverse is very inefficient. In general, the matrix of the equations formed from the given conditions of a network problem is sparse, whereas its inverse is full. By means of an appropriately ordered triangular decomposition, the inverse of a sparse matrix can be expressed as a product of sparse matrix factors, thereby gaining an advantage in computational speed, storage, and reduction of round-off error.

The method consists of two parts: 1) a scheme of recording the operations of triangular decomposition of a matrix such that repeated direct solutions based on the matrix can be obtained without repeating the triangularization, and 2) a scheme of ordering the operations that tends to conserve the sparsity of the original system. The first part of the method is not altogether new, but its computational possibilities are not widely recognized. The second part appears to be of recent origin, but it is probably only a rediscovery. Either part of the method can be applied independently, but the greatest benefit is obtained from the combined application of both parts.

The first part of the method is applicable to any matrix. Application of the second part, ordering to conserve sparsity, is limited to sparse matrices in which the pattern of nonzero elements is symmetric and for which an arbitrary

Manuscript received February 10, 1967; revised August 9, 1967. The unrevised version of this paper was published in the *Power Industry Computer Applications Conf. Rec.*, pp. 367–376, 1967. This conference was sponsored by the IEEE Power Group.

The authors are with the Bonneville Power Administration, Portland, Ore.

order of decomposition does not adversely affect numerical accuracy. Such matrices are usually characterized by a strong diagonal, and ordering to conserve sparsity increases the accuracy of the decomposition. A large class of network problems fulfills this condition. Generally it is not worth considering optimal ordering unless at least 80 percent of the matrix elements are zero.

At least three papers[1]-[3] have been written on various aspects of optimally ordered triangular decomposition applied to power system problems. Another paper[4] has been written on the related subject of optimally ordered Gauss-Jordan decomposition.

This paper extends previous work and presents it in a new matrix form. The method is used at Bonneville Power Administration (BPA) in many applications including power flow,[7] short circuit, transient stability, network reduction, switching transients,[8] reactive optimization,[9] tower design,[10] and others.

II. Factored Direct Solutions

This section, covering the first part of the method, shows how to derive an array of numbers from a nonsingular matrix A that can be used to obtain the effects of any or all of the following: A, A^{-1}, A^t, $(A^t)^{-1}$, and certain two-way hybrid combinations of these matrices. The superscript -1 means inverse and the superscript t means transpose. The scheme is applicable to any nonsingular matrix, real or complex, sparse or full, symmetric or nonsymmetric. Although the examples in this paper are limited to nodal equations, the method is also applicable to mesh equations. Its greatest advantage is realized in problems involving large sparse matrices.

The scheme to be described in this section is similar to those associated with the names of Gauss, Doolittle, Choleski, Banachiewicz, and others. All of these closely related schemes are computational variations of the basic process of triangularizing a matrix by equivalence transformations. They were originally developed for, and until very recently have only been described in terms of, manual computational procedures. Very little attention has been given to their special suitability for sparse matrices.

The basic scheme is first presented for the most general case, a full nonsymmetric matrix. Symmetry is then treated as a special case. Sparsity with optimal ordering, which is the primary objective of this development, is explained in Section III. Numerical examples of the basic scheme are given in Appendix I.

A. Triangular Decomposition

Triangular decomposition of a matrix by Gaussian elimination is described in many books on matrix analysis.[5],[6] Ordinarily, the decomposition is accomplished by elimination of elements below the main diagonal in successive columns. From the standpoint of computer programming for a sparse matrix, it is usually much more efficient to eliminate by successive rows. Therefore, this less familiar, but mathematically equivalent, scheme is illustrated here. The development is based on the equation

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

where A is a nonsingular matrix, x is a column vector of unknowns, and b is a known vector with at least one nonzero element. In the computer algorithm A is augmented by b as shown in (2) for an nth-order system.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{bmatrix} \tag{2}$$

The first step is to divide the elements of the first row by a_{11} as indicated in (3).

$$\begin{aligned} a_{1j}^{(1)} &= (1/a_{11})a_{1j} \qquad j = 2, n \\ b_1^{(1)} &= (1/a_{11})b_1. \end{aligned} \tag{3}$$

The superscripts indicate the order of the derived system.

The second step, as indicated in (4a) and (4b), is to eliminate a_{21} from the second row by linear combination with the derived first row, and then to divide the remaining derived elements of the second row by its derived diagonal element.

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ & 1 & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ & & \vdots & \cdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{bmatrix} \tag{4a}$$

$$\begin{aligned} a_{2j}^{(1)} &= a_{2j} - a_{21}a_{1j}^{(1)} \qquad j = 2, n \\ b_2^{(1)} &= b_2 - a_{21}b_1^{(1)} \\ a_{2j}^{(2)} &= (1/a_{22}^{(2)})a_{2j}^{(1)} \qquad j = 3, n \\ b_2^{(2)} &= (1/a_{22}^{(1)})b_2^{(1)}. \end{aligned} \tag{4b}$$

The third step, as indicated in (5a) and (5b), is to eliminate elements to the left of the diagonal of the third row and to divide the remaining derived elements of the row by the derived diagonal element.

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ & 1 & a_{23}^{(2)} & a_{24}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ & & 1 & a_{34}^{(3)} & \cdots & a_{3n}^{(3)} & b_3^{(3)} \\ & & \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} & b_n \end{bmatrix} \tag{5a}$$

$$\begin{aligned} a_{3j}^{(1)} &= a_{3j} - a_{31}a_{1j}^{(1)} \qquad j = 2, n \\ b_3^{(1)} &= b_3 - a_{31}b_1^{(1)} \\ a_{3j}^{(2)} &= a_{3j}^{(1)} - a_{32}^{(1)}a_{2j}^{(2)} \qquad j = 3, n \\ b_3^{(2)} &= b_3^{(1)} - a_{32}^{(1)}b_2^{(2)} \\ a_{3j}^{(3)} &= (1/a_{33}^{(2)})a_{3j}^{(2)} \qquad j = 4, n \\ b_3^{(3)} &= (1/a_{33}^{(2)})b_3^{(2)}. \end{aligned} \tag{5b}$$

Proceeding in this manner the nth derived system is obtained.

$$\begin{bmatrix} 1 & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ & 1 & \cdots & a_{2n}^{(2)} & b_2^{(2)} \\ & & \ddots & \vdots & \vdots \\ & & & 1 & b_n^{(n)} \end{bmatrix} \tag{6}$$

It should be noted that at the end of the kth step, work on rows 1 to k has been completed and rows $k+1$ to n have not yet entered the process in any way.

The solution can now be obtained by back substitution.

$$\begin{aligned} x_n &= b_n^{(n)} \\ x_{n-1} &= b_{n-1}^{(n-1)} - a_{n-1,n}^{(n-1)} x_n \\ x_i &= b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j. \end{aligned} \tag{7}$$

In programming, the x_i's replace the b_i's one by one as they are computed, starting with x_n and working back to x_1.

When A is full and n large, it can be shown that the number of multiplication-addition operations for triangular decomposition is approximately $1/3n^3$ compared with n^3 for inversion.[5]

It can easily be verified that triangularization in the same order by columns instead of rows would have produced identically the same result. Each eliminated element $a_{ij}^{(j-1)}$, $i>j$, would have been the same and the number of operations would have been the same. The back substitution also could have been accomplished by columns instead of rows in the same number of operations.

B. Recording the Operations

If the forward operations on b had been recorded so that they could be repeated, it is obvious that with this record and the upper triangle (6) for the back substitution, (1) could be solved for any vector b without repeating the triangularization. The recording of the forward operations, however, is trivial. Each forward operation is completely defined by the row and column coordinates and value of a single element $a_{ij}^{(j-1)}$, $i \geq j$, that occurs in the process. Therefore, it is unnecessary to do anything to record these elements except to leave them.

The rules for recording the forward operations of triangularization are:

1) when a term $1/a_{ii}^{(i-1)}$ is computed, store it in location ii
2) leave every derived term $a_{ij}^{(j-1)}$, $i>j$, in the lower triangle.

Since the forward as well as the back substitution operations are recorded in this scheme, it is no longer necessary to include the vector b. The final result of triangularizing A and recording the forward operations is symbolized in (8).

$$\begin{matrix} d_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & d_{22} & u_{23} & \cdots & u_{2n} \\ l_{31} & l_{32} & d_{33} & \cdots & u_{3n}. \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ l_{n1} & l_{n2} & l_{n3} & \cdots & d_{nn} \end{matrix} \tag{8}$$

The elements of (8), defined in terms of the derived systems of A in (2)–(6), are:

$$\begin{aligned} d_{ii} &= 1/a_{ii}^{(i-1)} \\ u_{ij} &= a_{ij}^{(i)} \qquad i < j \\ l_{ij} &= a_{ij}^{(j-1)} \qquad i > j. \end{aligned} \tag{9}$$

The matrix brackets are omitted in (8) to emphasize that the array is not strictly a matrix in the same sense as preceding examples, but only a scheme of recording. It will be referred to as the table of factors. In the literature this result is frequently shown as a factoring of the inverse matrix into the product of a lower and an upper triangular matrix, but it is more suitable for this discussion to consider it as a table of factors.

C. Computing Direct Solutions

It is convenient in symbolizing the operations for obtaining direction solutions to define some special matrices in terms of the elements of the table of factors (8). The following nonsingular matrices differ from the unit matrix only in the row or column indicated.

$$\begin{aligned} &D_i: \text{Row } i = (0, 0, \cdots 0, d_{ii}, 0, \cdots 0, 0) \\ &L_i: \text{ Col } i = (0, 0, \cdots 0, 1, -l_{i+1,i}, -l_{i+2,i}, \cdots -l_{n-1,i}, -l_{n,i})^t \\ &L_i^*: \text{Row } i = (-l_{i,1}, -l_{i,2}, \cdots -l_{i,i-1}, 1, 0, \cdots 0, 0) \\ &U_i: \text{Row } i = (0, 0, \cdots 0, 1, -u_{i,i+1}, -u_{i,i+2}, \cdots -u_{i,n-1}, -u_{i,n}) \\ &U_i^*: \text{ Col } i = (-u_{1,i}, -u_{2,i}, \cdots -u_{i-1,i}, 1, 0, \cdots 0, 0)^t. \end{aligned} \tag{10}$$

The inverses of these matrices are trivial. The inverse of the matrix D_i involves only the reciprocal of the element d_{ii}. The inverses of the matrices L_i, L_i^*, U_i, and U_i^* involve only a reversal of algebraic signs of the off-diagonal elements.

The forward and back substitution operations on the column vector b that transform it to x can be expressed as premultiplications by matrices D_i, L_i or L_i^*, and U_i or U_i^*. Thus the solution of $Ax=b$ can be expressed as indicated in (11a)–(11d).

$$U_1 U_2 \cdots U_{n-2} U_{n-1} D_n L_{n-1} D_{n-1} L_{n-2} \cdots L_2 D_2 L_1 D_1 b = A^{-1} b = x \tag{11a}$$

$$U_1 U_2 \cdots U_{n-2} U_{n-1} D_n L_n^* D_{n-1} L_{n-1}^* \cdots L_3^* D_2 L_2^* D_1 b = A^{-1} b = x \tag{11b}$$

$$U_2^* U_3^* \cdots U_{n-1}^* U_n^* D_n L_{n-1} D_{n-1} L_{n-2} \cdots L_2 D_2 L_1 D_1 b = A^{-1} b = x \tag{11c}$$

$$U_2^* U_3^* \cdots U_{n-1}^* U_n^* D_n L_n^* D_{n-1} L_{n-1}^* \cdots L_3^* D_2 L_2^* D_1 b = A^{-1} b = x. \tag{11d}$$

Each of these four equations describes a sequence of operations on the vector b that is equivalent to premultiplication by A^{-1}. For an nth-order system, each equation indicates n multiplications, n^2-n multiplication-additions, and n additions, excluding, of course, multiplications by unity which are only symbolic. This corresponds exactly with the n^2 multiplication-additions required for premultiplication by the inverse. Starting with D_1 and proceeding to the left, (11a) describes the forward and back substitution operations that would be performed on b if it augmented A during triangularization by columns. Equation (11b) describes the same result for triangularization by rows. Equations (11c) and (11d) describe other sequences of the same operations giving the same result. Depending on programming techniques, one of these four equivalent se-

quences usually will prove to be the most convenient.

Direct solutions of other systems based on A can be obtained from the table of factors by use of the following two theorems:

1) the inverse of a product of nonsingular matrix factors is the product of the inverses of the factors in reverse order
2) the transpose of a matrix product is the product of the transposes of the factors in reverse order.

Although the matrix A may have been lost in the triangularization, its effect is recoverable. Given x, the vector b can be obtained as indicated in (12a) or (12b). Two other equations analogous to (11c) and (11d) using U_i^* also could be written.

$$D_1^{-1}L_1^{-1}D_2^{-1}L_2^{-1}\cdots L_{n-1}^{-1}D_n^{-1}U_{n-1}^{-1}U_{n-2}^{-1}\cdots U_2^{-1}U_1^{-1}x = Ax = b \quad (12a)$$

$$D_1^{-1}(L_2^*)^{-1}D_2^{-1}(L_3^*)^{-1}\cdots(L_n^*)^{-1}D_n^{-1}U_{n-1}^{-1}U_{n-2}^{-1} \cdots U_2^{-1}U_1^{-1}x = Ax = b. \quad (12b)$$

Again the number of essential operations is n^2.

Direct solutions for the corresponding transpose system $A^ty=c$ can be obtained as indicated in (13) and (14).

$$D_1L_1^tD_2L_2^t\cdots L_{n-1}^tD_nU_{n-1}^tU_{n-2}^t\cdots U_2^tU_1^tc = (A^t)^{-1}c = y \quad (13)$$

$$(U_1^t)^{-1}(U_2^t)^{-1}\cdots(U_{n-2}^t)^{-1}(U_{n-1}^t)^{-1} \cdot D_n^{-1}(L_{n-1}^t)^{-1}\cdots(L_2^t)^{-1}D_2^{-1}(L_1^t)^{-1}D_1^{-1}y = A^ty = c. \quad (14)$$

Again the number of operations is n^2, and, as in the previous examples, equations could be written using L_i^* and U_i^*.

Although (11)–(14) look somewhat complicated, they represent simple operations that can be guided by the table of factors (8). Each equation indicates the following:

1) a sequence for using the elements of the array for performing operations on the independent vector
2) a rule for using the subscripts of the elements of the array to indicate the elements of the vector to be operated upon
3) an algebraic sign rule for elements l_{ij} and u_{ij}
4) a rule indicating whether to multiply or divide by elements d_{ii}.

The operations can be extended to include certain two-way hybrid solutions with the matrix partitioned at any point desired. Let the hybrid column vector g be defined as

$$g^t = (b_1, b_2, \cdots, b_k, x_{k+1}, x_{k+2}, \cdots, x_n). \quad (15)$$

If g is given, the unknown first k elements of x and the $k+1$ to nth elements of b can be obtained directly. First it is necessary to compute an intermediate vector z.

$$U_{n-1}^{-1}U_{n-2}^{-1}\cdots U_{k+2}^{-1}U_{k+1}^{-1}D_kL_k^*\cdots L_3^*D_2L_2^*D_1g = z. \quad (16)$$

Equation (16) indicates the solution of an upper triangular system to obtain the first k elements of z and the solution of an independent lower triangular system to obtain the remaining elements of z.

By using elements from z and g, the composite vector h is formed.

$$h^t = (z_1, z_2, \cdots, z_k, x_{k+1}, x_{k+2}, \cdots, x_n). \quad (17)$$

Using h, the first k unknown elements of x are obtained from (18).

$$U_1U_2\cdots U_{k-1}U_kh = x. \quad (18)$$

Equation (18) defines the back substitution of (11a) from k to 1.

The $k+1$ to nth unknown elements of b are obtained from

$$(L_{k+1}^*)^{-1}D_{k+1}^{-1}\cdots(L_{n-1}^*)^{-1}D_{n-1}^{-1}(L_n^*)^{-1}D_n^{-1}z = b'. \quad (19)$$

Equation (19) defines the back substitution of (12b) from n to $k+1$. The vector b' is composed of the first k elements of z, which were unaffected by the premultiplication on the left side of the equation, and the $k+1$ to nth elements of b. Since the first k elements of b were given, the solution is complete. Again the total number of operations is n^2.

If only the unknown elements of x are wanted, (20) should be used.

$$U_1U_2\cdots U_{k-1}U_kD_kL_k^*\cdots L_3^*D_2L_2^*D_1g = x. \quad (20)$$

This requires only kn operations. Since no elements of the table of factors below the kth row are needed in this case, it is unnecessary to compute and store them.

A hybrid solution for the equation $A^ty=c$, in which the first k elements of c and the $k+1$ to nth elements of y are given, can be developed analogous to (15)–(20). Other hybrid solutions also can be developed, but they require more than n^2 operations and will not be considered.

D. Symmetry

If A is symmetric, only the d_{ii} and u_{ij} terms of the table of factors are needed. All of the foregoing direct solutions can be obtained by noting that in the symmetric case L_i can be defined in terms of D_i and U_i:

$$L_i = D_iU_i^tD_i^{-1}. \quad (21)$$

Substituting (21) in (11a) and canceling adjacent factors of the form $D_i^{-1}D_i$ gives

$$U_1U_2\cdots U_{n-2}U_{n-1}D_nD_{n-1}U_{n-1}^t\cdots D_2U_2^tD_1U_1^tb = A^{-1}b = x. \quad (22)$$

More convenient expressions from the standpoint of operation and notation can be obtained by noting that products of the form $U_i^tD_j$, $i>j$, are commutative. Thus all of the D_i factors can be grouped into one diagonal matrix

$$D = D_1D_2\cdots D_n.$$

With this arrangement (22) becomes

$$U_1U_2\cdots U_{n-2}U_{n-1}DU_{n-1}^t\cdots U_2^tU_1^tb = A^{-1}b = x. \quad (23)$$

Similar substitutions for the symmetric case can be made in the other equations for direct solutions.

Symmetry also permits a saving of almost one half of the triangularization operations because it is unnecessary to perform any operations to the left of the diagonal in order

to obtain the derived elements $a_{ij}^{(j-1)}$, $i>j$. These elements can be recovered from previously derived elements by the relationship

$$a_{ij}^{(j-1)} = (a_{jj}^{(j-1)})a_{ji}^{(j)} \quad i = 2, n; \quad j = 1, i-1. \quad (24a)$$

An alternative procedure avoids a number of multiplications equal to those indicated in (24a). In the nonsymmetric case each row was normalized by multiplying each term to the right of the diagonal by the reciprocal of the diagonal term before proceeding to the next row. If this step is deferred, leaving the terms of a row unnormalized until later, the normalizing multiplication and recovery of the symmetric term, when needed, can be combined. Thus after processing row i, each term is left in the form $a_{ij}^{(i-1)}$, $i<j$, instead of $a_{ij}^{(i)}$. When the term $a_{ji}^{(i-1)}$, $j>i$, is needed in processing row j to be used as a multiplier of row i in the same way as if row i had been normalized, it is computed from $a_{ij}^{(i-1)}$. The operations using this alternative procedure are as follows:

$$a_{jk}^{(i)} = a_{jk}^{(i-1)}\left[(1/a_{ii}^{(i-1)})a_{ji}^{(i-1)}\right]a_{ik}^{(i-1)} \quad i < j; \quad k = j, n. \quad (24b)$$

The term in brackets, which is $a_{ij}^{(i)}$ in normalized form, replaces $a_{ij}^{(i-1)}$ as soon as it has been used in (24b). The saving in operations is trivial for a full matrix, but significant for a sparse one.

If the triangularized system is left unnormalized, n additional operations are required for each direct solution. This is significant for a sparse matrix; therefore, normalization as described is recommended.

E. Modifications

If A is changed, it is necessary to modify the table of factors to reflect the change. If an element $a_{k,m}$ of A is changed, elements of the table of factors with row and column subscripts i, j are affected as follows:

1) if $k>m$, elements a_{ij} with $i=k, j\geq m$ and $i>k, j\geq k$ are affected.
2) if $k\leq m$ elements a_{ij} with $i\geq k, j=m$ and $i\geq m, j>m$ are affected.

Since most changes in a row involve a change in the diagonal term, the first case is the more important. The number of operations required to effect a change in the kth row is approximately $1/3(n-k+1)^3$. However, any number of additional changes in the lower right submatrix bounded by row k and column k can be included in the same pass without increasing the volume of computations. Changes below row k in columns 1 to $k-1$ can also be made in the same pass with only a small increase in the volume of computations. If the rows in which frequent changes will be required can be anticipated, they should be located at the bottom of the matrix.

In using the inverse, every change in A affects all n^2 elements of A^{-1}; therefore, at least n^2 operations on A^{-1} are required to account for any change in A. Whether the inverse or the table of factors is the easier to modify in a particular situation depends on the nature and number of matrix changes required and the scheme of inverse modification being used.

F. Comparative Advantages for a Full Matrix

When A is full, the advantages of the factored form of direct solution are:

1) the array of factors can be obtained in one-third the number of operations of the inverse
2) the factored form gives the effect of A and the hybrid matrix; the inverse does not.

The methods are comparable in that:

1) complete solutions normally require the same number of operations
2) the storage requirement is the same.

The advantages of the inverse are:

1) complete solutions require only kn operations when the independent vector has only k nonzero elements
2) under some circumstances the inverse can be modified to reflect changes in the original matrix more easily than the table of factors.

III. Sparsity and Optimal Ordering

When the matrix to be triangularized is sparse, the order in which rows are processed affects the number of nonzero terms in the resultant upper triangle.[1] If a programming scheme is used which processes and stores only nonzero terms, a very great savings in operations and computer memory can be achieved by keeping the table of factors as sparse as possible. The absolute optimal order of elimination would result in the least possible terms in the table of factors. An efficient algorithm for determining the absolute optimal order has not been developed, and it appears to be a practical impossibility. However, several effective schemes have been developed for determining near-optimal orders.

A. Schemes for Near-Optimal Ordering

The inspection algorithms for near-optimal ordering to be described are applicable to sparse matrices that are symmetric in pattern of nonzero off-diagonal terms; i.e., if a_{ij} is nonzero, then a_{ji} also is nonzero but not necessarily equal to a_{ij}. These are matrices that occur most frequently in network problems. From the standpoint of programming efficiency, the algorithms should be applied before, rather than during, the triangularization. It is assumed in what follows that the matrix rows are originally numbered according to some external criterion and then renumbered according to the inspection algorithm. Eliminations are then performed in ascending sequence of the renumbered system.

Following are descriptions of three schemes for renumbering in near-optimal order. They are listed in increasing order of programming complexity, execution time, and optimality.

1) Number the rows according to the number of nonzero off-diagonal terms before elimination. In this scheme the rows with only one off-diagonal term are numbered first, those with two terms second, etc., and those with the most terms, last. This scheme does not take into account

any of the subsequent effects of the elimination process. The only information needed is a list of the number of nonzero terms in each row of the original matrix.

2) Number the rows so that at each step of the process the next row to be operated upon is the one with the fewest nonzero terms. If more than one row meets this criterion, select any one. This scheme requires a simulation of the effects on the accumulation of nonzero terms of the elimination process. Input information is a list by rows of the column numbers of the nonzero off-diagonal terms.

3) Number the rows so that at each step of the process the next row to be operated upon is the one that will introduce the fewest new nonzero terms. If more than one row meets this criterion, select any one. This involves a trial simulation of every feasible alternative of the elimination process at each step. Input information is the same as for scheme 2).

The comparative advantages of these schemes are influenced by network topology and size and the number of direct solutions wanted. The only virtue of scheme 1) is its simplicity and speed. For nodal equations of power networks scheme 2) is enough better than scheme 1) to justify the additional time required for its execution. Scheme 3) does not appear to be enough better than scheme 2) to justify its use for power networks, but it is believed that it may be effective for other networks. Since the authors' experience is limited to power networks, these conclusions may not be valid for other networks. Other algorithms may need to be developed. Schemes intermediate between 1) and 2) or between 2) and 3), as well as entirely different schemes, are possible.

It can be demonstrated that there exist certain matrices for which none of the given schemes will be very effective. However, they are unlikely to occur in most network equations.

B. Other Factors Influencing Ordering

Under some conditions it may be advantageous or necessary to number certain rows last even though this adversely affects sparsity. Among these conditions are the following.

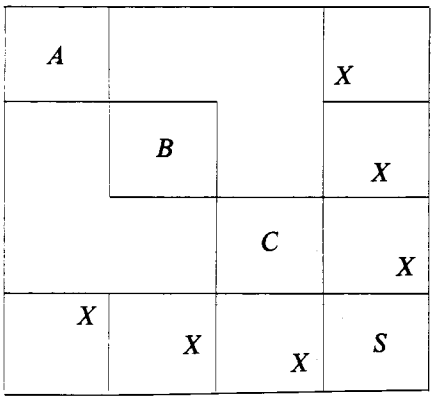
1) It is known that changes in the original matrix will occur only in a certain few rows. If these rows are numbered last, only these rows in the table of factors need to be modified to reflect the changes.

2) It is known that changes in the independent vector will occur only in a certain few elements. If rows are numbered so that no changes in the vector occur above row k, the forward operations preceding row k need be performed only once, and not repeated for each subsequent case.

3) The matrix is only slightly nonsymmetric. If the rows are numbered so that the nonsymmetric portion of the matrix is last, advantage of the symmetry can be taken down to this point.

4) The hybrid operation is to be used and it is necessary to have the appropriate rows last.

When the network on which the matrix is based is composed of subnetworks with relatively few interconnections, the matrix should be arranged as indicated schematically in (25).



(25)

This may be interpreted to be the pattern of a nodal admittance matrix of a very large network composed of three subnetworks. The pattern of the table of factors derived from this matrix would be similar. Submatrices A, B, and C, which presumably are very sparse, represent nodes within subnetworks A, B, and C, respectively; and S represents nodes associated with their interconnections. All submatrices outside of A, B, C, and S are zero except those indicated by the X's, which contain the few off-diagonal terms interconnecting the system in S. The renumbering algorithm should be applied to each submatrix independently. Within each of the submatrices A, B, and C, the rows with nonzero terms in the columns of S should be located last. This arrangement will not ordinarily be achieved by any of the given renumbering algorithms without additional logic or external coding.

The arrangement offers the following advantages:

1) If a change is made in one of the submatrices, say B, it affects only B and S in the table of factors, but not A and C.
2) If, after obtaining a solution, a change is made in the independent vector in the rows of only one partition, say B, only the operations on the vector defined by B and S need to be repeated to obtain a new solution.
3) If computer memory is limited, the procedure can be implemented by operating on only one submatrix at a time.
4) As more subnetworks are added to a system, the relationship between system size and the table of factors tends to be almost linear.

C. Effectiveness of Optimal Ordering

The effectiveness of optimal ordering can be expressed as the ratio between the number of nonzero off-diagonal terms in the table of factors and the number of similar terms in the original matrix. The nearer this ratio is to unity, the more effective the ordering. At BPA it has been found to vary between 1.7 and 2.5 for nodal admittance matrices of power networks of up to 1000 nodes. The ratio is influenced more by network topology than by system size. Obviously, it might be different for matrices based on other kinds of networks.

When the matrix is sparse, the arithmetic operations required to compute the table of factors for an nth-order system are as follows:

$$\text{divisions} = n$$

$$\text{multiplications} = s = \sum_{i=1}^{n-1} r_i. \tag{26a}$$

The number of multiplication-additions is dependent on whether the matrix is symmetric or not. For nonsymmetric matrices with symmetric pattern of nonzero off-diagonal elements:

$$\text{multiplication-additions} = \sum_{i=1}^{n-1} r_i^2. \tag{26b}$$

For symmetric matrices:

$$\text{multiplication-additions} = \sum_{i=1}^{n-1} (r_i^2 + r_i)/2 \tag{26c}$$

where r_i is the number of nonzero terms to the right of the diagonal in row i in the table of factors and s is the total number of such terms.

Determining the operation count for changing the array of factors to reflect changes in the original matrix is quite complicated and dependent upon the actual programming scheme. In general, if the changes in A begin in row k, (26a)–(26c) may be used with summations beginning at k instead of 1 to get an approximate count.

The operations required for computing any of the possible complete direct solutions based on A are:

$$\begin{aligned} \text{multiplications or divisions} &= n \\ \text{additions} &= n \\ \text{multiplication-additions} &= 2s. \end{aligned} \tag{26d}$$

Partial solutions can be obtained in fewer operations.

D. Programming

A comprehensive presentation of the various programming techniques that have been developed for the method is beyond the scope of this paper. Some have been discussed in other papers.[1],[4],[10]

When working with a full matrix, the address in computer memory of each matrix element can be related to its row and column indices in such a way that programming is quite easy. However, in order to achieve the benefits of sparsity, the programming scheme must store and process only nonzero elements. This requires, in addition to the memory allocation for the matrix elements themselves, tables of indexing information to identify the elements and to facilitate their addressing. Programming is more difficult in this case and much of the method's potential advantage can be lost through poorly planned programming. The need for expert programming cannot be overemphasized.

With the most effective programming techniques, the operations are performed as if they are being done by visual inspection and manual computation. At the completion of the optimal ordering algorithm [scheme 2) or 3)], the exact form of the table of factors is established and this information is recorded in various tables to guide the actual elimination process. During the elimination no operation is performed that would lead to a predictable zero result and no memory allocation is made for a predictable zero element. The original matrix, the table of factors, and all indexing tables contain only nonzero elements.

Rows are transferred from the original matrix to a compact working row in which elements to the left of the diagonal are eliminated by appropriate linear combination with previously processed rows from the partially completed table of factors. When work on the row has been completed, it is added to the table of factors. The actual procedure varies depending on the nature of the application. If the matrix is nonsymmetric, the derived elements to the left of the diagonal must be stored. The arrangement of the tables depends to some extent on the subsequent needs for direct solutions.

An example of one possibility for arranging the table of factors is indicated in Table I. This represents the final result for a seventh-order symmetric matrix. The columns labeled Loc refer to relative addresses in computer memory. The ith location in the D Table contains the element d_{ii} and the location in the U Table of the first u_{ij} element of row i. The column of the U Table labeled J contains the column subscript j of u_{ij}. Thus row 2 begins in location 3 of the U Table and contains u_{23}, u_{26}, and u_{27} in locations 3, 4, and 5, respectively. Row 3 begins in location 6.

TABLE I
EXAMPLE OF STORAGE AND INDEXING SCHEME FOR TABLE OF FACTORS

D Table			U Table		
Loc	D Factors	Loc in U Table	Loc	U Factors	J
1	d_{11}	1	1	u_{12}	2
2	d_{22}	3	2	u_{17}	7
3	d_{33}	6	3	u_{23}	3
4	d_{44}	8	4	u_{26}	6
5	d_{55}	10	5	u_{27}	7
6	d_{66}	11	6	u_{36}	6
7	d_{77}	12	7	u_{37}	7
8	—	12	8	u_{45}	5
			9	u_{46}	6
			10	u_{56}	6
			11	u_{67}	7

E. Comparative Advantages for a Sparse Matrix

When A is sparse the advantages of the factored form in addition to those previously listed are:

1) the table of factors can be obtained in a small fraction of the time required for the inverse
2) the storage requirement is small, permitting much larger systems to be solved
3) direct solutions can be obtained much faster unless the independent vector is extremely sparse
4) round-off error is reduced
5) modifications due to changes in the matrix can be made much faster.

The only disadvantage of the method is that it requires much more sophisticated programming techniques.

The effectiveness of the method may be judged by the results of a typical large problem described in Appendix II.

IV. Conclusions

Optimally ordered triangular factorization appears to be better than any other known method for the direct solution of the sparse linear equations that arise in many network problems. The method opens the way for improvements in existing applications and offers possibilities for new applications that would not be feasible with any other approach. It eliminates the need for resorting to slowly converging iterative methods in order to solve many large network problems and the need for compromising with desired problem size in order to obtain the benefits of direct solutions. Further improvements and extensions of the method are expected.

Appendix I

Numerical Examples

The following examples based on the full nonsymmetric matrix A are given only to indicate the operations. They do not show the effects of sparsity or the methods of programming to take advantage of it.

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 7 \end{bmatrix}. \tag{27}$$

The table of factors for A is

$$\begin{matrix} \frac{1}{2} & \frac{1}{2} & \frac{3}{2} \\ 2 & \frac{1}{2} & \frac{1}{2} \\ 3 & \frac{5}{2} & \frac{4}{5} \end{matrix}. \tag{28}$$

With b given, the equation $Ax=b$ can be solved for x using (29), which is based on (11a):

$$\begin{matrix} U_1 & & U_2 & & D_3 & & L_2 \\ \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{3}{2} \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & -\frac{1}{2} \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & & \frac{4}{5} \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & -\frac{5}{2} & 1 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} D_2 & & L_1 & & D_1 & b & x \\ \begin{bmatrix} 1 & & \\ & \frac{1}{2} & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ -2 & 1 & \\ -3 & & 1 \end{bmatrix} & \begin{bmatrix} \frac{1}{2} & & \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 6 \\ 9 \\ 14 \end{bmatrix} = & \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \end{matrix} \tag{29}$$

With x given, b can be obtained from (30), which is based on (12a):

$$\begin{matrix} D_1^{-1} & & L_1^{-1} & & D_2^{-1} & & L_2^{-1} \\ \begin{bmatrix} 2 & & \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ 2 & 1 & \\ 3 & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & \frac{5}{2} & 1 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} D_3^{-1} & & U_2^{-1} & & U_1^{-1} & x & b \\ \begin{bmatrix} 1 & & \\ & 1 & \\ & & \frac{5}{4} \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \frac{1}{2} \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & \frac{1}{2} & \frac{3}{2} \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = & \begin{bmatrix} 6 \\ 9 \\ 14 \end{bmatrix}. \end{matrix} \tag{30}$$

With c given, the equation $A^t y=c$ can be solved for y by using (31), which is based on (13):

$$\begin{matrix} D_1 & & L_1^t & & D_2 & & L_2^t \\ \begin{bmatrix} \frac{1}{2} & & \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & -2 & -3 \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & \frac{1}{2} & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & -\frac{5}{2} \\ & & 1 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} D_3 & & U_2^t & & U_1^t & c & y \\ \begin{bmatrix} 1 & & \\ & 1 & \\ & & \frac{4}{5} \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & -\frac{1}{2} & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ -\frac{3}{2} & & 1 \end{bmatrix} & \begin{bmatrix} 9 \\ 9 \\ 17 \end{bmatrix} = & \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}. \end{matrix} \tag{31}$$

With y given, c can be obtained from (32), which is based on (14):

$$\begin{matrix} (U_1^t)^{-1} & & (U_2^t)^{-1} & & D_3^{-1} & & (L_2^t)^{-1} \\ \begin{bmatrix} 1 & & \\ \frac{1}{2} & 1 & \\ \frac{3}{2} & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & \frac{1}{2} & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ & & \frac{5}{4} \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \frac{5}{2} \\ & & 1 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} D_2^{-1} & & (L_1^t)^{-1} & & D_1^{-1} & y & c \\ \begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & 2 & 3 \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 2 & & \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = & \begin{bmatrix} 9 \\ 9 \\ 17 \end{bmatrix}. \end{matrix} \tag{32}$$

Given the hybrid vector g such that

$$g^t = (b_1, x_2, x_3) = (6, 1, 1) \tag{33}$$

the intermediate vector z is obtained by (34), which is based on (16):

$$\begin{matrix} U_2^{-1} & & D_1 & g & z \\ \begin{bmatrix} 1 & & \\ & 1 & \frac{1}{2} \\ & & 1 \end{bmatrix} & \begin{bmatrix} \frac{1}{2} & & \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 6 \\ 1 \\ 1 \end{bmatrix} = & \begin{bmatrix} 3 \\ \frac{3}{2} \\ 1 \end{bmatrix}. \end{matrix} \tag{34}$$

The vector h is formed from z_1, x_2, x_3 as indicated in (17) and x_1 is computed by (35), which is based on (17):

$$\begin{matrix} U_1 & & h & x \\ \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{3}{2} \\ & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} = & \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \end{matrix} \tag{35}$$

The elements b_2 and b_3 are computed by (36), which is based on (18):

$$\begin{matrix} (L_2^*)^{-1} & & D_2^{-1} & & (L_3^*)^{-1} \\ \begin{bmatrix} 1 & & \\ 2 & 1 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 2 & \\ & & 1 \end{bmatrix} & \begin{bmatrix} 1 & & \\ & 1 & \\ 3 & \frac{5}{2} & 1 \end{bmatrix} \end{matrix}$$

$$\begin{matrix} D_3^{-1} & z & b' \\ \begin{bmatrix} 1 & & \\ & 1 & \\ & & \frac{5}{4} \end{bmatrix} & \begin{bmatrix} 3 \\ \frac{3}{2} \\ 1 \end{bmatrix} = & \begin{matrix} 3 \\ 9 \\ 14 \end{matrix}. \end{matrix} \tag{36}$$

APPENDIX II
EXAMPLE PROBLEM

The purpose of this example problem is to demonstrate the effectiveness of the method. A program has been written using the method for solving the nodal admittance equation $YE=I$. The given data are the elements of the complex symmetric nodal admittance matrix Y and vectors of node currents I. Solutions consist of the vectors of complex node voltages E corresponding to each given vector I. The program consists of three subroutines: 1) optimal ordering, 2) formation of the table of the factors, and 3) direct solutions.

The program is dimensioned for 1000 nodes. It is written mostly in MAP assembly language for the IBM 7040, a comparatively slow computer with an eight-microsecond access time and 36-bit word size. The results of a typical power network problem using scheme 2) for optimal ordering are as follows:

number of nodes = 839
number of branches = 1122
table of factors:
d_{ii} terms = 839
u_{ij} terms = 2210.

Complex arithmetic operations for direction solutions:

additions = 839
multiplications = 839
multiplication-additions = $2 \times 2210 = 4420$
computing times:
optimal ordering by scheme 2 = 8.11 seconds
forming table of factors = 10.42 seconds
each direct solution = 2.80 seconds.

The total time required for optimal ordering and computing the table of factors is the time that should be compared with that of other methods for obtaining the inverse; the time required for direct solutions should be compared with that of multiplying by the inverse. The time for read-in and organization of data is not included.

The dimensions of tables for a 1000-node power network problem are as follows:

d_{ii} terms:	$2 \times 1000 =$	2000 floating point words
u_{ij} terms:	$2 \times 3000 =$	6000 floating point words
solution vector:	$2 \times 1000 =$	2000 floating point words
index tables:		4000 integer words
total		14000 words.

If the matrix were real instead of complex, the time for optimal ordering would be unaffected, but the time for computing the table of factors and the direct solutions would be reduced to less than 25 percent of that for the complex matrix. The dimensions of tables with factor 2 could be reduced by one half.

If the matrix were complex and nonsymmetric, the optimal ordering and direct solution times would be unaffected, but the computation of the table of factors would take almost twice as long. An additional table of $2 \times 3000 = 6000$ floating point words would also be required for the l_{ij} terms.

The time of 2.8 seconds is for a complete direct solution of 839 complex node voltages. Any of the other possible complete direct solutions could be obtained in approximately the same time. Partial solutions could be obtained faster. Any row or column of the inverse, if needed, could be computed in 2.8 seconds. A complete nonsymmetric inverse could be computed in this manner in 839×2.8 seconds, or about 40 minutes. A symmetric inverse for which only a triangular matrix is needed would require about 70 percent as much time. Since each column of the inverse would be computed independently, the round-off error in single precision would be negligible.

The 4000 words of index tables are for unpacked integers of no more than three-decimal digits; they could be considerably compressed. The time for optimal ordering could be improved. The programs for obtaining the table of factors and direct solutions are carefully planned and probably could not be improved very much.

ACKNOWLEDGMENT

The authors wish to aknowledge the valuable assistance of N. R. Isseks and D. Bree, Jr., who contributed to this work by developing algorithms and programs.

REFERENCES

[1] N. Sato and W. F. Tinney, "Techniques for exploiting the sparsity of the network admittance matrix," *IEEE Trans. Power Apparatus and Systems*, vol. 82, pp. 944–950, December 1963.
[2] J. Carpentier, "Ordered eliminations," *Proc. Power System Computation Conf.* (London), 1963.
[3] H. Edelmann, "Ordered triangular factorization of matrices," *Proc. Power System Computation Conf.* (Stockholm), 1966.
[4] R. Baumann, "Some new aspects of load flow calculation," *Proc. Power Industry Computer Applications Conf.* (Clearwater, Fla.), p. 91, May 1965.
[5] A. Ralston, *A First Course in Numerical Analysis*. New York: McGraw-Hill, 1965, pp. 398–415.
[6] E. Bodewig, *Matrix Calculus*. New York: Interscience, 1959, pp. 101–124.
[7] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," presented at IEEE Winter Power Meeting, New York, N. Y., 1967.
[8] H. W. Dommel, "A method for solving transient phenomena in multi-phase systems," presented at Power Systems Computation Conf., Stockholm, 1966.
[9] J. Peschon, W. F. Tinney, D. S. Piercy, O. J. Tveit, and M. Cuenod, "Optimum control of reactive power flow," presented at IEEE Winter Power Meeting, New York, N. Y., 1967.
[10] G. H. Jensen, "Designing self-supporting transmission towers with the digital computer," presented at Power Industry Computer Applications Conf., Pittsburgh, Pa., May 15–17, 1967.