Solution of Large Sparse Systems by Ordered Triangular Factorization

WILLIAM F. TINNEY AND W. SCOTT MEYER

Abstract—Analysis and/or control problems arising in utility systems are often of high dimensionality. As a result, the usual symbology of formal vector-matrix notation can obscure structural properties of the system or model which might otherwise allow very simple numerical solution. Simultaneous sparse systems arising from a number of different areas are shown to be of this type, including problems of electric power flow, water distribution, mechanical structure analysis, differential equation solution, optimal control, and linear programming. All are highly amenable to efficient solution schemes employing the sparse-matrix method of ordered triangular factorization.

I. Introduction

Sparsity-oriented solution methods (defined in Section II) have successfully resolved a number of large problems encountered in utility engineering. It is the purpose of this paper to convey an appreciation of how and why this has been possible, by means of numerous illustrative examples.

As the subject is far too involved to be comprehensively covered by a single paper, only a selective presentation is possible. Motivational principles have been purposely emphasized, with references to the literature providing avenues for further study. The paper is thus tutorial in nature.

The essence of ordered triangular factorization (OTF) is incredibly simple, merely involving the solution of linear equations, as one learned in high school, by Gaussian elimination. The basic idea is in fact so simple that it was completely overlooked by most mathematicians and engineers for a decade or two (since computers became available). Hopefully this paper will serve to spread the message.

II. Predominance of Sparsity in Large Problems

The term "sparsity" is used to indicate the relative absence of certain problem interconnections. For example, Fig. 6(a) shows a schematic diagram representing the interconnection of pipes in a small water distribution system. While in theory each junction or node could be directly connected to every other one by a branch (pipe), such is far from the case; this network has only 1.26 branches per node, rather than the 31 possible. Here the physical system itself is sparse, and it is crucial that the analysis and solution method exploit this structure (as shall be seen shortly). Other systems having such physical sparsity are power networks and mechanical structures (see Figs. 4 and 7).

Manuscript received November 6, 1972. Paper recommended by J. Peschon, Past Chairman of the IEEE S-CS Estimation and Identification Committee.

The authors are with the Bonneville Power Administration, Portland, Oreg. 97208.

On the other hand, perhaps the problem under study is largely mathematical in nature, not directly associated with a physical structure. This is often the case with linear programming, control, or differential equation problems. Such problems will be called sparse if most functions or equations in question only involve a few of the possible problem variables, as in Figs. 9 and 10. The key to efficient solution will be to take advantage of this structure.

III. The Need to Look Beyond Matrix Inversion

As later examples will show, the crucial calculation of many problems can be reduced to finding the solution x to a system of linear equations

$$[A]x = b. \tag{1}$$

Here b is a known vector and $[A]$ is a known nonsingular matrix, assumed to be sparse (most $a_{ij} = 0$). The symbolic solution to (1) is of course provided by matrix inversion,

$$x = [A]^{-1}b. \tag{2}$$

Yet while $[A]$ is sparse, $[A]^{-1}$ for many problems is completely full (all elements nonzero), with its conventional explicit calculation requiring on the order of N^3 multiply adds (for an $N \times N$ matrix). Not only is this computational effort required to produce the inverse astronomical for large N (see the upper curve of Fig. 1), but even just the storage of the result in core may be impossible. Even if the inverse were handed to one for free, calculation of x by means of (2) requires on the order of N^2 multiply adds, which is prohibitive for large N.

IV. Gaussian Elimination as the Crucial Ingredient

Even if $[A]$ were not sparse, about $\frac{2}{3}$ of the inversion effort above could be saved by simply solving for x using Gaussian elimination. As a simple concrete illustration of this procedure, consider the third-order problem

$$\begin{bmatrix} 3 & 0 & 12 \\ 0 & 6 & 12 \\ 3 & 10 & 16 \end{bmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 15 \\ 15 \\ 24 \end{pmatrix}. \tag{3}$$

The step-by-step solution then proceeds as follows.

Step 1—Eliminate x_1 [from equations 2 and 3]: Divide the first equation by $a_{11} = 3$. Then subtract $a_{31} = 3$ times

$3x_1 + 12x_3 = 15$ (1)

$3x_1 + 10x_2 + 16x_3 = 24$ (2)

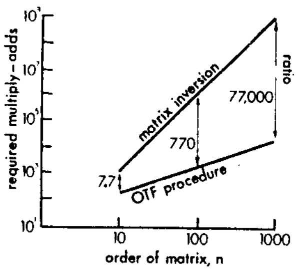


Fig. 1. Comparison of the numerical effort required by matrix inversion with that of OTF solutions to typical power network problems.

this now-modified first equation from the third equation, producing

$$\begin{aligned} x_1 \quad\quad\; + 4x_3 &= 5 \\ 6x_2 + 12x_3 &= 15 \\ 10x_2 + \;4x_3 &= 9. \end{aligned} \tag{4a}$$

Step 2—Eliminate x_2 [from equation 3]: Divide the second equation of (4a) by $a_{22} = 6$. Then subtract $a_{32} = 10$ times this equation from the third equation, giving the result

$$\begin{aligned} x_1 \quad + \quad 4x_3 &= 5 \\ x_2 + \quad 2x_3 &= 5/2 \\ -16x_3 &= -16. \end{aligned} \tag{5}$$

Step 3—Normalize the Last Equation (Find x_3): Complete the downward operations by dividing the last equation by its diagonal element $a_{33} = -16$, producing

$$\begin{aligned} x_1 \quad + 4x_3 &= 5 \\ x_2 + 2x_3 &= 5/2 \\ x_3 &= 1. \end{aligned} \tag{6}$$

Step 4—Find x_2 [from equation 2]: The back-substitution operations begin with the next to the last, or second, equation. Using the now-known x_3 value, x_2 is found from the second equation of (6), giving

$$\begin{aligned} x_1 \quad\quad + 4x_3 &= 5 \\ x_2 = 1/2 \quad\quad x_3 &= 1. \end{aligned} \tag{7}$$

Step 5: Find x_1 [from equation 1]: With x_3 and x_2 known, x_1 can now be found from the first equation of (7), thus completing the solution for x:

$$x_1 = 1 \qquad x_2 = 1/2 \qquad x_3 = 1. \tag{8}$$

Before leaving this example, it is convenient to review the use of a network graph associated with the coefficient matrix $[A]$ and show how it is modified during the elimination steps. Note that $[A]$ of (3) has a symmetric nonzero pattern, with all diagonal elements a_{kk} nonzero. The nonzero structure of $[A]$ is thus uniquely described by Fig. 2(a), with the network graph constructed according to the following rules:

1) one node exists for each equation or variable;
2) a branch (k,m) connects node k to node m if $a_{km} \neq 0$ (or, equivalently, $a_{mk} \neq 0$).

The elimination of x_1 gave rise to (4a), having Fig. 2(b) associated with it. Note that only variables remaining to be eliminated appear, since the nonzero structure of the row and column being dropped is fixed for all time (and hence can no longer be controlled). The general rule for elimination of node k on such a network graph is as follows:

1) erase node k, as well as all of its connected branches;
2) add new branches to the remaining graph as follows: if node k had a branch to node n and one to node m before being erased, then create a branch between nodes n and m (if one does not already exist).

This latter rule 2) is referred to as "fill-in," where an originally zero position of $[A]$ is made nonzero by the elimination process. Although such did not occur in (4a) or (5), it would have happened had node 3 been eliminated first as in Fig. 2(d). Algebraically, this node-3 elimination corresponds to the following manipulations of (3): divide the third equation by 16; subtract 12 times this result from the second equation and also from the first, producing

$$\begin{aligned} (3/4)x_1 + (-15/2)x_2 \quad\quad &= -3 \\ (-9/4)x_1 + \;(-3/2)x_2 \quad\quad &= -3 \\ (3/16)x_1 + \quad (5/8)x_2 + x_3 &= 3/2. \end{aligned} \tag{4b}$$

Gaussian elimination is the most efficient direct ("exact" noniterative) solution scheme when $[A]$ is full, requiring on the order of $\frac{1}{3}N^3$ multiply adds to solve for x. This is $\frac{1}{3}$ of the matrix-inversion effort and therefore represents progress. Yet such a 3-to-1 improvement just is not significant enough; it is the N-cubed dependence that is deadly for large N (such as $N = 1000$).

As will be explained later, the preceding elimination process can be generalized for flexibility, thereupon bearing the name of "triangular factorization." When performed in a carefully chosen order (see next section), one then has OTF.

V. Exploitation of Sparsity Within Gaussian Elimination

When $[A]$ is sparse, experience has shown that rows and columns can usually be interchanged (permuted) so that the number of nonzero terms is kept small throughout the elimination process. The original problem sparsity can be largely preserved during the solution, reducing not only the storage requirements, but even more dramatically the computational effort that is required.

As a small example, consider a matrix $[A]$ having the nonzero pattern of Fig. 3(a). Of the 100 possible matrix elements, only 30 are nonzero (10 diagonals and 20 off-diagonals). If elimination now proceeds in the natural

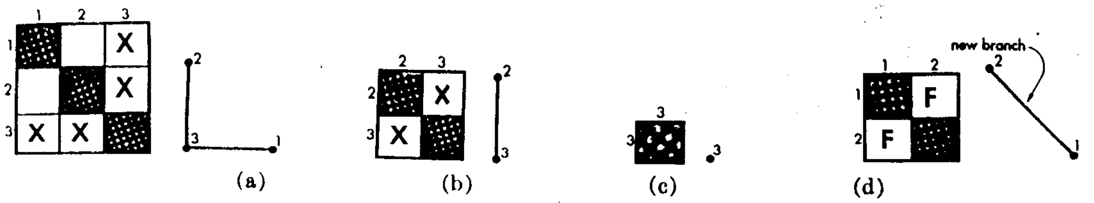


Fig. 2. Representation of the nonzero structure of $[A]$ upon elimination, for the three-variable example of (3). (a) Original nonzero matrix structure, and associated network graph. (b) Result after elimination of x_1. (c) Result after elimination of x_1 and then x_2. (d) Result if, in the first step, node 3 of (a) had been eliminated; there is fill-in between nodes 1 and 2.

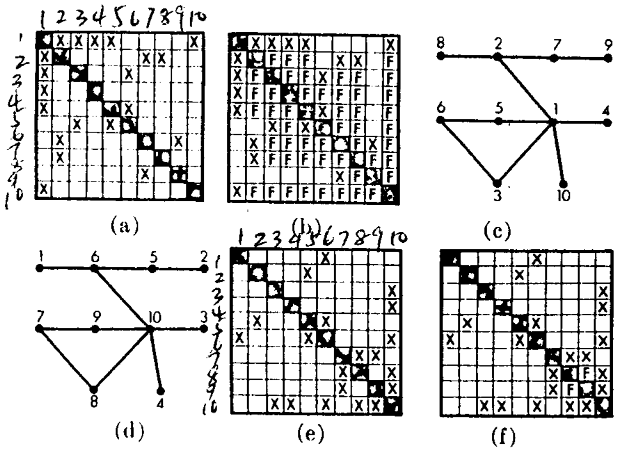


Fig. 3. Illustrative example of order 10, which shows the importance of ordering upon the preservation of the original sparsity during elimination. (a) Nonzero pattern of the original $[A]$ (before any row and column permutations). (b) Nonzero pattern after elimination of the original equations. (c) Network graph associated with the original equation set, with the nonzero pattern of (a). (d) Same network graph as in (c) only renumbered consistent with sparsity considerations. (e) Nonzero pattern of the reordered $[A]$, defined by the graph of (d). (f) Nonzero pattern after elimination of the reordered equations.

order, most of the 70 originally zero locations will fill in during the process; only 20 elements remain zero throughout this elimination [Fig. 3(b)]. Of course this is better than matrix inversion (where all cells are filled in), but far from optimal. To show a better order, it is convenient to use the associated graph of Fig. 3(c). If this graph were renumbered as in Fig. 3(d), the associated permutation of rows and columns places $[A]$ in the form of Fig. 3(e). Elimination of this matrix in natural order then produces only two fill-ins [Fig. 3(f)]. The original sparsity has been largely preserved, reducing not only the core storage requirement, but also (even more dramatically) the computational effort.

VI. Sample Efficiency of OTF for Large Sparse Matrices

Typical results for electric power network equations have been placed in Fig. 1 for comparison with conventional explicit matrix inversion. Experience with electric networks has shown that computational effort to produce a solution varies about linearly with the number of equations N, requiring typically only 1300 multiply adds for 100 equations, as shown on the graph. This is about 0.13 percent of the matrix inversion effort shown, for such a hundredth-order system. Typically only 560 of the $100 \times 100 = 10\,000$ possible elements are ever nonzero during the elimination process, or 5.6 percent of the value for matrix inversion (which yields a full matrix $[A]^{-1}$). Since the ratio of superiority increases about as the square of system size $(3 - 1 = 2)$, such numerical comparisons become astounding for large N. When $N = 1000$, for example, the preceding percentages for computational effort and storage requirements become 0.0013 and 0.056 percent, respectively.

Although patterns such as Fig. 3 simply are not possible for systems as large as $N = 1000$, Fig. 6 does present the counterpart of Fig. 3(d) and (f) for a 62-node water network taken from [7]. This is a network approaching realistic size and one can begin to visualize the extreme sparsity exploitation that is possible in the much larger problems.

VII. Examples of Utility Problems Solvable by OTF

Before detailing the algorithms and procedures used to obtain solutions by OTF, several sample applications from various disciplines of engineering will be presented.

A. Static Optimization of Electric Power Networks

Electric power networks have provided a very fertile field for the application of sparsity programming solution techniques. The so-called optimal power flow, an optimization problem of static (steady-state) control of the network, provides an excellent example of this.

Given an electric network of N nodes, the assumed starting point of the steady-state analysis is Kirchhoff's voltage and current laws in the form of nodal admittance equations $[Y]V = I$, or

$$Y_{kk}V_k + \sum_{j \neq k} Y_{kj}V_j = I_k, \qquad k = 1,\cdots,N. \tag{9}$$

Here I_k is the injected current and V_k the voltage at node k—both complex numbers (phasors). Diagonal elements Y_{kk} are always nonzero and usually are strong (dominant). Off-diagonal Y_{kj} are nonzero only when node k is connected to node j of the network by a branch (a transmission line or transformer); typically about three such nonzero entries exist per row, independent of system size (see small sample graph of Fig. 4). Hence $[Y]$ is extremely sparse for large N and efficient solution schemes should exploit this fact.

The power-flow problem [5] involves a solution of (9)

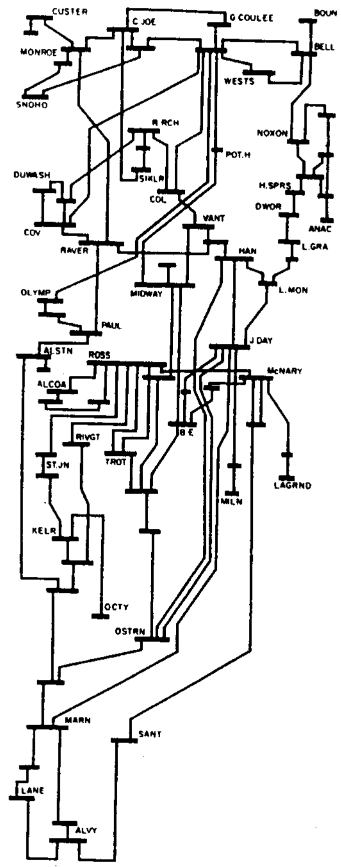


Fig. 4. Network graph of 76-node 102-branch high-voltage electric transmission network for the Pacific Northwest.

subject to certain power and voltage constraints. Injected real and reactive power are given by

$$P_k - jQ_k = V_k^* I_k = |V_k|^2 Y_{kk} + V_k^* \sum_{j \neq k} Y_{kj} V_j \tag{10}$$

so a system of such equations must be solved. At some nodes m, voltage magnitude $|V_m|$ may be fixed with only the angle a variable—in which case only the real part of (10) at node m would be written. But, in general, two real equations in two real unknowns (voltage magnitude and angle) may exist for each node. Such equations are unfortunately nonlinear, but at least the original sparsity of $[Y]$ remains. We shall denote the equation set by

$$g(X) = 0 \text{ or } g_k(x_1, \cdots, x_N) = 0, \qquad k = 1, \cdots N \tag{11}$$

the solution of which (for state vector x) is found by power- (or load) flow programs [5].

Optimization enters the picture when certain problem parameters u_i, which are otherwise constants, are to be adjusted during the solution. Thus the power-flow equations can be written functionally as

$$g(X,u) = 0 \tag{12}$$

where now we want to adjust u so as to minimize some cost functional

$$\min F(X,u). \tag{13}$$

This "cost" may have physical or economic significance, as in the case of economic dispatch (where total system operating cost in dollars/hour is to be minimized by adjusting generator real power outputs); or F may involve penalty terms needed to represent device or operational constraints so as to produce a better adjusted solution, in some sense. Though the problem has many variations and practical complications, it is still basically of the form

$$\min F(X,u)$$
$$g(X,u) = 0. \tag{14}$$

Successful solution of (14) has been provided by gradient (steepest descent) methods—the movement from one feasible solution point to another in the direction of most rapid cost decrease (in the space of control variables u_i). A feasible solution is found by solving (12) for X by Newton's method, holding u fixed:

$$X^{\text{new}} = X^{\text{old}} + \Delta X \tag{15}$$

where ΔX satisfies

$$[J] \cdot \Delta X = -g(X^{\text{old}}, u). \tag{16a}$$

Here $[J]$ is the Jacobian matrix with elements

$$[J] = (J_{im}) \qquad J_{im} = \partial g_i / \partial x_m. \tag{16b}$$

From (10), this matrix has the same extreme sparsity as does $[Y]$. Thus Newton's method just requires a series of sparse-matrix linear equation solutions. This proceeds very rapidly (as attested to by Fig. 1), with typically 3 or 4 iterations sufficient for a solution starting from a poor initial guess such as with all x_i equal. The generalized reduced gradient (GRG) showing in which direction to adjust u is then defined from calculus by

$$\nabla u = \left(\frac{\partial F}{\partial u}\right) + \left[\frac{\partial g}{\partial u}\right]^t \cdot [J^t]^{-1} \cdot \left(\frac{\partial F}{\partial x}\right). \tag{17}$$

This is a trivial calculation if one views the inverse as meaning just the solution of a sparse system of equations:

$$\nabla u = \left(\frac{\partial F}{\partial u}\right) + \left[\frac{\partial g}{\partial u}\right]^t \lambda, \tag{18}$$

where λ satisfies

$$[J]^t \lambda = \left(\frac{\partial F}{\partial x}\right). \tag{19}$$

As $[J]$ was already triangularized on the last Newton iteration of the preceding power-flow solution, finding λ of (19) requires negligible effort (see Section IX). The matrix $[\partial g / \partial u]$ is an extremely sparse matrix generally, so the multiplication of (18) is trivial.

The incremental change in X produced by a small change in the control variable u_k is provided by differentiation of (12):

$$S_k \triangleq \left(\frac{\partial X}{\partial u_k}\right) = -[J]^{-1} \cdot \left(\frac{\partial g}{\partial u_k}\right). \tag{20}$$

Just as with the preceding vector λ, solution for such a sensitivity vector S_k is trivial once $[J]$ has been triangularized. Such sensitivities are useful not only for planning, but also for on-line control applications (such as capacitor adjustment for local voltage control).

Reference [6] developed the optimal power flow primarily to adjust generator voltage magnitudes so as to minimize the network real power loss (thermal heating in the lines). Fig. 5 shows that four iterations were sufficient to solve a 328-node 493-branch problem having 80 control variables u_k. Such convergence is excellent, though unfortunately not always typical of problems having more complex (i.e., badly deformed) objective functions.

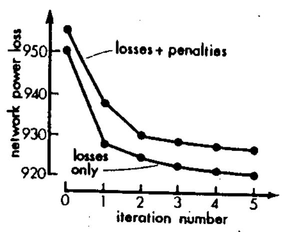


Fig. 5. Decrease in network loss with iteration number, for gradient optimization of 328-node electric power network.

B. Solution of Water Distribution Systems

Given an interconnected network of pipes, pumps, and reservoirs with consumptions specified at nodes (interconnection points of the pipes) of unknown pressure, the problem is to find all unknown node pressures and all pipe flows. This task shall be called the water-flow problem, with a small sample network of 62 nodes shown in Fig. 6.

Nodal equations that exploit the extreme sparsity of the physical network are readily developed. Considering a pipe connecting node k to node j, the pressure differential across the pipe is approximately proportional to the square[1] of the pipe flow:

$$|P_k - P_j| = Q_{kj}^2/c_{kj}^2, \tag{21}$$

where $1/c_{kj}$ is a friction coefficient depending on pipe dimensions. Inverting this relation,

$$Q_{kj} = c_{kj}\sqrt{|P_k - P_j|}\,\operatorname{sign}(P_k - P_j). \tag{22}$$

Note that Q_{kj} is the flow from node k to node j, that flow occurs in the direction of decreasing pressure. Node equations then follow from conservation of mass at node k,

$$I_k = \sum_{j \neq k} c_{kj}\sqrt{|P_k - P_j|}\,\operatorname{sign}(P_k - P_j), \tag{23}$$

where I_k is the net injection into the node. At a supply node, this is the water flow into the network; at a point of consumption, it is the negative of the flow out of the network (the demand). At a node of known pressure (such as a reservoir), no equation is written, with the unknown injection I_k evaluated after all other pressures have been found. Pumps can be shown to not change the problem form, though such details cannot be covered here.

Just as with the power flow equations (10), these constraints are of the form (11); if one wanted to show the dependence on system parameters, then they are of the form (12). Solution for unknown pressures hence can proceed analogously by Newton's method as defined by (15) and (16). Exactly the same sparsity techniques applicable to electric networks are applicable here, though the following points are worthy of note.

1) The Jacobian matrix $[J]$ is not defined when two adjacent nodes k and j are assumed to have the same pressure, since

$$J_{kj} = \frac{\partial I_k}{\partial P_j} = \frac{-c_{kj}}{2\sqrt{|P_k - P_j|}}, \qquad k \neq j. \tag{24}$$

Thus a uniform or "flat" start having all variables P_j equal is not possible, unlike the situation with electric power flow. Likewise, potential trouble develops during the iterative solution if a branch having zero or near-zero pressure difference exists. While sophisticated formulation-change techniques for such a branch are possible, physical intuition suggests practical remedies that seem to work well. For example, $P_k \simeq P_j$ means that the branch flow Q_{kj} is nearly zero; thus one can ignore such a pipe, dropping it from the network while the near equality exists.

2) From (24), it follows that $[J]$ is symmetric ($[J] = [J]^t$). This allows for more efficient solution (see Section IX).

3) Pressure-reducing valves introduce nonbilateral characteristics that require a certain amount of judgment or trial and error. This is comparable to the problem of solving electronic networks having diodes.

4) The use of Newton's method with nodal formulation sparsity is doubly important in view of the square root appearing in (24). In the more general case, an evaluation of $(P_k - P_j)^\eta$ is required, with $\eta = 0.46$ for the Hazen–Williams formula of fluid flow. Such exponentiation is very time consuming, with the standard library subroutine of the Univac 1108 requiring numerical effort comparable to about 80 multiply adds. The secret to fast water network solutions, then, is to reduce the number of such required evaluations. The quadratic convergence of Newton's method serves to minimize the number of iterations, while the use of a sparse formulation (nodal) reduces the number of evaluations per iteration.

Although the sparsity exploitation just outlined has come late to the water-flow problem, extensive progress in the next few years appears likely.

C. Analysis of Pin-Jointed Mechanical Structures

Pin-jointed mechanical structures provide another natural application of sparse-matrix solution techniques. For example, three typical high-voltage electric transmission-line towers are shown in Fig. 7, where the ladder-like physical sparsity of interconnection is obvious. Subject to given assumed maximum loading and design

[1] Formulas using other than the square of the flow, like Hazen–Williams with 1.85, could equally well be developed here, of course.

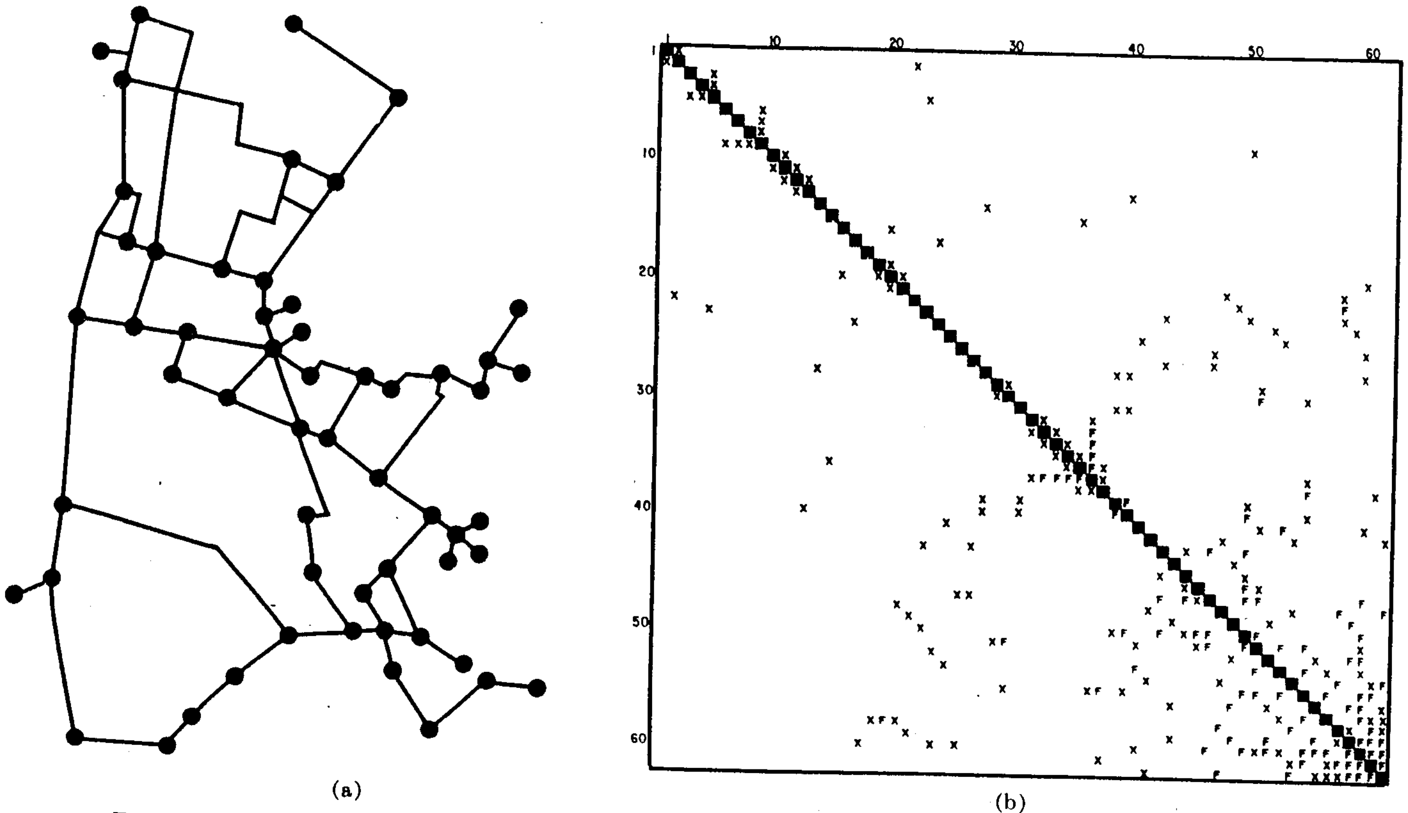


Fig. 6. Sample small 62-node water distribution network of [7]. (a) Network graph, drawn approximately to geographical proportions. (b) Nonzero pattern of resulting matrix $[A]$, showing fill-in (F) upon triangularization.

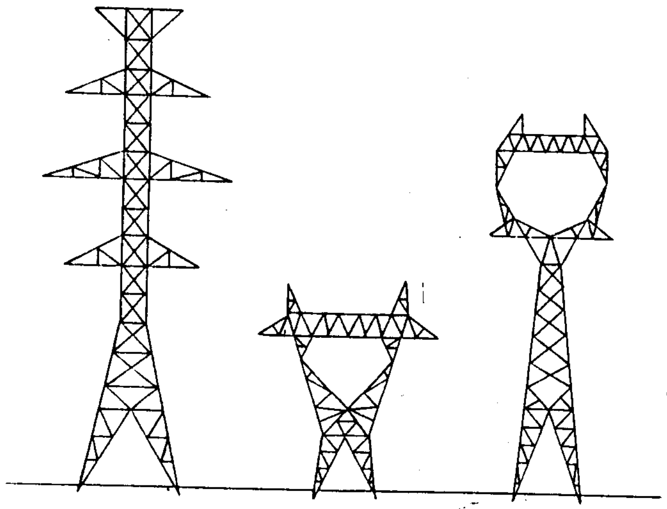


Fig. 7. Sketch showing the ladder-like sparsity of 500-kV electric power transmission towers.

safety margins, it is desired to choose the configuration and member dimensions that will be the most economical (in some sense). The crucial step in such an extremely complicated optimization problem is, of course, just the analysis of a given assumed configuration, which is no simple task.

The procedure of [8] is sophisticated and realistic in that an indeterminate analysis is performed—indeterminate because not only member interconnections are considered, but also the elastic properties thereof (as defined by a 6×6 stiffness matrix for each member). Such mechanical structure analysis is thus more sophisticated than the power or water-flow problem, where the complication of geometric directions does not exist.

The fundamental calculation can be shown [8] to reduce to the following linear equation solution for $\boldsymbol{d}$:

$$[K]\boldsymbol{d} = \boldsymbol{f}, \tag{25}$$

where

$\boldsymbol{f}$	vector of known applied forces;
$[K]$	stiffness matrix for the tower;
$\boldsymbol{d}$	unknown vector of joint deflections.

Having solved for $\boldsymbol{d}$, the internal member stresses (which are what are actually needed) can be found by a straightforward transformation, thus completing the basic solution calculation.

The coefficient matrix $[K]$ has sparsity determined by the physical connectivity of the members and hence is amenable to efficient solution by OTF. Elimination order is determined by matrix banding (see Section VIII), which allows for the solution of very large problems with minimum core storage requirements. Jensen reports that a tower of the type shown in Fig. 7(a) involves a matrix of order 390; it requires only several thousand cells of core storage for the banded-matrix solution, and gives rise to 24 000 nonzero terms (of the 152 000 possible) in the triangularization process. An even larger example is given in [7].

D. Solution of Linear Ordinary Differential Equations by Implicit Integration

A purely mathematical use of OTF is illustrated by the numerical solution of a system of linear ordinary differential equations. In standard state-variable form, the initial value problem can be written as

$$\dot{x} = [A]x + [B]u \tag{26}$$

$$y = [C]X + [D]u, \qquad X(0) \text{ given}; \quad u(t) \text{ given for } t \geq 0, \tag{27}$$

where X is the system state, u the control, and y the output. One wants to find $y(t)$ for $t \geq 0$, an operation that is easy using (27) once $X(t)$ has been found. The real difficulty, then, consists of finding the solution $X(t)$ to

$$\dot{X} = [A]X + f(t), \qquad X(0) \text{ given}; \quad f(t) \text{ given for } t \geq 0. \tag{28}$$

For constant $[A]$ and low-dimensional systems, "exact" closed-form solutions may be obtained. For example, the use of an exponential power series, eigenvalues and eigenvectors, or Laplace transformation is well documented elsewhere [10]. But such is not the case here, where we instead postulate that $[A]$ is a large sparse possibly time-varying matrix. Hence one must resort to numerical solution methods—procedures that only yield $X(t)$ approximately and only at discrete time points. The most common such algorithms are the well-known Runge–Kutta or predictor-corrector procedures [3], which merely require evaluation of the right-hand side of (28) for given X and t. These are "explicit" integration methods and do not require the simultaneous linear-equation solution of the method to follow. Yet while simple to apply, excessive arithmetic effort (very small step size) may be required to keep the numerical process from "blowing up," thus giving totally erroneous answers.

An alternative allowing much larger time steps for certain problems is provided by "implicit integration." Since one can functionally write the solution

$$X(t_1) = X(t_0) + \int_{t_0}^{t_1} \dot{X}(t)\, dt, \tag{29}$$

the only question consists of how the integral on the right is to be approximated. Referring to Fig. 8, we need the shaded area in order to advance the solution one time step. Now rather than just use polynomial interpolation through known points (at times $t_0, t_{-1}, \cdots, t_{-n+2}$) as with predictor methods, implicit integration adds the unknown point at time t_1 to the interpolation formula. The interpolating polynomial is integrated and, upon isolation of the vector of unknowns $X(t_1) \triangleq X^1$, it is found that this vector must satisfy the following system of linear equations:

$$[G]X^1 = r_n X^0 + \sum_{j=-n+2}^{0} d_j \dot{X}^j, \tag{30}$$

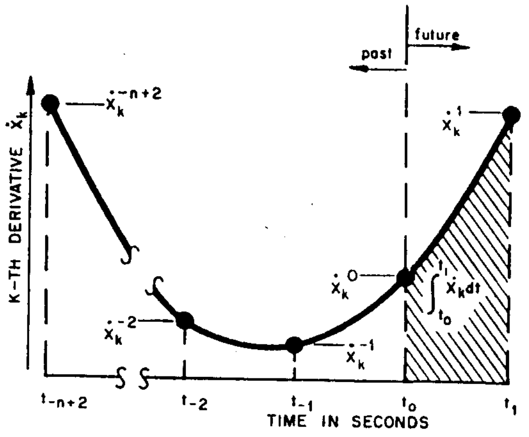


Fig. 8. Polynomial interpolation of $\dot{x}_k$, for use with (29).

where

$$[G] = [A] + s_n[I] \tag{31}$$

with $[I]$ being the identity matrix and $s_n, r_n, d_0, d_{-1}, \cdots, d_{-n+2}$ being scalar constants depending upon the number of points n used in the interpolation and upon their spacing. For example, if parabolic interpolation with uniform spacing Δt is used, one has $n = 3$, and

$$s_n = r_n = \frac{12}{5\Delta t} \qquad d_0 = -8/5 \qquad d_{-1} = 1/5. \tag{32}$$

In any case, a sparse-matrix solution of linear equations is indicated. Note that the nonzero pattern of $[G]$ is identical with that of $[A]$, assuming that all diagonal terms a_{kk} are nonzero.

Only integration by the trapezoidal rule ($n = 2$) was considered by [9], showing that considerable potentiality may exist for application to the transient-stability problem of electric power systems. Since such problems may be of order 1000 or more, the economic consequences are great. As a small example, Fig. 9(a) shows the block diagram of a sample ninth-order system with one very small time constant (0.0001 s). Dommel and Sato [9] show that to successfully integrate such equations subject to a step input $u(t)$, the time step Δt of implicit integration can be taken about 500 times that of Runge–Kutta:

implicit integration ($n = 2$): $\Delta t_{\max} \simeq 0.01$ s;

fourth-order Runge–Kutta: $\Delta t_{\max} \simeq 0.000\,02$ s.

The prediction has been made that average overall running times of large transient stability studies may be cut by a factor of about five due to use of such implicit integration. Use of higher order ($n > 2$) methods is now under investigation.

E. Discrete-Time Optimal Control Problem

The classic discrete-time optimal control problem to be considered here may be stated as follows:

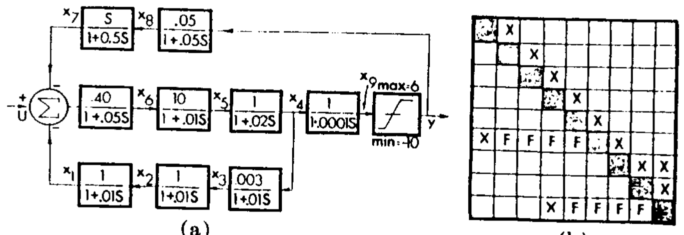


Fig. 9. Sample ninth-order linear differential equation problem of [9]. (a) Block diagram of dynamic system. (b) Nonzero pattern of coefficient matrix $[A]$, showing fill-in upon triangularization by the symbol F.

$$\min \psi = F(X^N) + \sum_{i=0}^{N-1} L_i(X^i,u^i) \tag{33}$$

subject to given state transition equations

$$X^{i+1} = \phi^i(X^i,u^i), \qquad i = 0,\cdots,N-1 \tag{34}$$

and variable constraints

$$a^i \leq X^i \leq b^i, \qquad i = 1,\cdots,N \tag{35}$$

$$\alpha^{i-1} \leq u^{i-1} \leq \beta^{i-1}, \qquad i = 1,\cdots,N. \tag{36}$$

Here the state vector x is of order n and the control vector U of order m,

$$X = \begin{pmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} \qquad u = \begin{pmatrix} u_1 \\ \cdot \\ \cdot \\ \cdot \\ u_m \end{pmatrix}. \tag{37}$$

Further, $F,L_0,\cdots,L_{N-1}$ are given scalar cost functions and a^i, b^i, α^i, and β^i are given vectors of variable bounds.

If one does not worry about dependent-variable constraints (35), then this problem is seen to be identical in form to that of the optimal power flow (14); $X^1,\cdots,X^N$ when all joined together form one giant state vector of $n \times N$ components, while $u^1,\cdots,u^{N-1}$ form a control vector with $m \times N$ entries. The original dimensionality has been multiplied by the number of time steps N and can thus become very large even though n and m are themselves small. Yet as is typical of mathematical programming problems involving successive time steps, the crucial matrix, the Jacobian $[J]$ of (16a), has a block-diagonal structure as shown in Fig. 10(a). Note that $[J]^t$ of (19) is pretriangularized, leading to a simple recursive back substitution for λ:

$$\lambda^i = [J^i]^t\lambda^{i+1} + \left(\frac{\partial\psi}{\partial x_j{}^i}\right). \tag{38}$$

Here the notation $[J^i]$ is used to indicate the Jacobian matrix of (34) for time step i,

$$[J^i] = (J_{kj}{}^i) \qquad J_{kj}{}^i = \partial\phi_k{}^i/\partial x_j{}^i. \tag{39}$$

The same sort of trivial solution procedure can be used to find the feasible solution $X^1,\cdots,X^N$ about which the gradient is being calculated.

But such a simple calculation is not possible if one or more components of $X^1,\cdots,X^N$ reach the limits (35) during the computation. To avoid use of penalty functions that would approximately maintain feasibility, [11] used the technique of switching the roles of x and u. The idea was to treat an $x_k{}^i$ as a control (independent) variable when it has reached its limit, replacing it for solution purposes by some $u_m{}^j$. Thus with $X^1,\cdots,X^N$, $u^0,\cdots,u^{N-1}$ treated as one single composite vector of problem variables, $N \times n$ of these are to be chosen as dependent, and $N \times m$ as independent. The structure of the resulting matrix $[J]^t$ of (19) then is found by only considering the appropriate $N \times n$ associated rows of the derivative matrix of Fig. 10(b). Here all equations of (34) have been differentiated with respect to all possible variables, and then transposed, with definitions

$$[C^i] = (c_{kj}{}^i) \qquad c_{kj}{}^i = \partial\phi_k{}^i/\partial u_j{}^i. \tag{40}$$

A simple low-order recursive calculation like (38) is no longer possible in general, with the simultaneous solution of larger numbers of coupled equations required. As seen, the resulting $[J]$ will always be "banded" (see Fig. 12 and its accompanying discussion) if rows and columns are taken in their natural order. Yet a nonzero diagonal may be a problem, and if n is of moderate size, numerical effort may be excessive. Individual problem structures (i.e., sparsity properties of $[J^k]$ and $[C^k]$) will generally determine what the best elimination order may be [11].

F. Linear Programming Problems

When placed in standard form, the general linear programming problem becomes

$$\max z = c^tX \tag{41}$$

subject to

$$[A]X = b, \qquad X \geq 0 \tag{42}$$

where $[A]$ is an $m \times n$ matrix with $n > m$. It can be shown that only m components of X need be nonzero, forming an m-vector X_B called a basic solution. The associated columns of $[A]$ are assumed to span m-space, forming a nonsingular basis matrix $[B]$. By definition,

$$X_B = [B]^{-1}b. \tag{43}$$

The simplex algorithm solves the problem by moving from one such feasible X_B to another until the optimum is reached, by successively replacing one column of $[B]$ by a column of $[A]$ not then in the basis.

The crucial factor in this solution is the ability to compute and store $[B]^{-1}$—not explicitly, but in factored form. While details are beyond the scope of this paper [1], [2], [12], two general procedures are in use: the elimination form of the inverse (EFI), and the product form of the inverse (PFI). The former is identical to OTF as used in this paper. In fact, it appears that the earliest known large-scale application of such ideas occurred in the operations-research area [13].

Linear programming problems are often very large, with $m = 10\,000$ not an uncommon capability (see [2, pp.

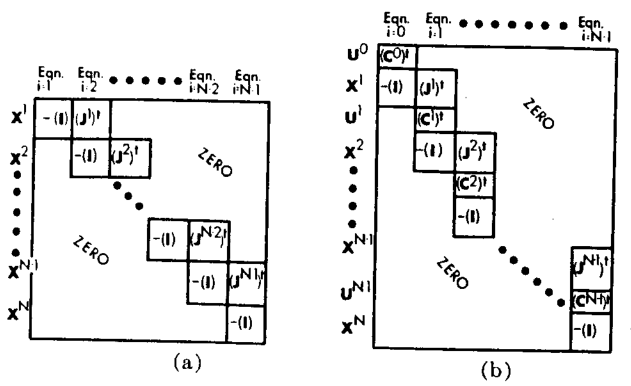


Fig. 10. (a) Structure of $[J]^t$ when $U^0, \cdots, U^{N-1}$ constitute the control (independent) variables of (33)–(34). (b) Structure when all equations of (34) are differentiated with respect to all problem variables and then transposed.

107–112]). Obviously $[A]$ for such problems must be extremely sparse. Details as to exactly how the EFI and PFI exploit such structure can be found elsewhere [1], [2], [12], [13].

VIII. Ordering Schemes to Preserve Sparsity During Elimination

Nothing has so far been said as to how a near-optimal ordering for OTF is accomplished in practice. Yet this is a key point, worthy of considerable attention.

To simplify the presentation, the assumption of a strong diagonal shall be made, so that pivoting to avoid division by zero or near-zero elements is never required. This was implied in the Fig. 3 considerations and shall be continued here. Note that for simplicity $[A]$ is assumed to have a symmetric nonzero pattern, and that rows and columns of $[A]$ are switched simultaneously, so that "once a diagonal element, always a diagonal element." While exceptions to this case are of practical concern, they excessively complicate the presentation and mask an appreciation of the basic idea.

Practical schemes for ordering electric and water networks include the following, in order of increasing sophistication and difficulty.

Scheme 1: Number nodes of the original graph in order of nondecreasing number of adjacent branches (i.e., those with one connected branch are taken first, then those with two, etc.).

In the remaining two schemes, one successively chooses a next node (or variable) to be eliminated based on the current reduced graph. He then alters that graph to correspond to elimination of this node and repeats the process. The criteria involved are as follows.

Scheme 2: Search the reduced graph for a node having minimum number of adjacent branches and take this one as the next node.

Scheme 3: Simulate the elimination of all possible nodes on the reduced graph, keeping track of the number of new branches that each elimination would produce. The next node to be chosen is that which produces the minimum number of new branches.

As an example of these three procedures, consider the network of Fig. 3 with the Fig. 3(c) numbering taken as the old or original numbering, which is to be improved.

Scheme 1 merely requires looking at Fig. 3(c), and noting the number of branches adjacent to each node. Old nodes 4, 8, 9, and 10 have one branch each; nodes 3, 5, 6, and 7 have two; node 2 has three; and finally node 1 has five. Since within a group of k objects there are $k!$ permutations, it follows that there are $4!4!1!1! = 256$ different Scheme 1 orderings for this very small problem. That shown in Fig. 3(d) does not conform, since although selection of the first five nodes is consistent, the sixth (old node 2) is not.

Scheme 2 starts with the full graph of Fig. 3(c). Since old nodes 4, 8, 9, and 10 all have minimum number of adjacent branches, any one can be chosen as the new node 1. Choosing old node 8 as in Fig. 3(d), the corresponding elimination is then performed, producing the reduced graph of Fig. 11(a). Since here old nodes 4, 9, and 10 all have one connected branch, one can take his pick of these as new node 2. Arbitrarily choosing old node 9, its elimination yields the reduced graph of Fig. 11(b). The process is continued until all nodes of the original graph have been so eliminated, until the reduced graph is annihilated.

Scheme 3 begins by trial elimination of all ten possible nodes of Fig. 3(c). One finds that no new branches are created by eliminating old nodes 4, 8, 9, or 10, one branch is created upon eliminating 3, 5, 6, or 7, three upon eliminating node 2, and ten upon eliminating node 1. Hence one can take his pick among the first listed set, with the choice of old node 9 acceptable just as it was for Scheme 2. Hence after its elimination, Fig. 11(a) applies. Simulating all 9 possible eliminations here, one finds that only old nodes 4, 9, or 10 produce a minimum of no new created branches. Just as with Scheme 2, old node 9 can be taken as new node 2, and the subsequent elimination yields Fig. 11(b). The eight possible eliminations here are then simulated, etc., thus ending the example.

Of the three schemes mentioned, Scheme 2, or derivatives thereof, has received the widest use for electric network solutions. Figures on effectiveness are provided in [5], [6], [14], [15]. Scheme 1 is poor enough for large problems so as to be unacceptable, while Scheme 3 is only slightly better (at much greater computational expense).

Networks having ladder-like structure (Fig. 12(a), for example) are well known to behave poorly under the previous three reordering schemes. The ideal elimination order progresses sequentially in the direction of the structure. Using one such numbering as shown in Fig. 12(a), the nonzero elements of $[A]$ are seen to be clustered or banded about the diagonal [Fig. 12(b)]. This is referred to as banded-matrix ordering, with the following important property: only cells within the band need be considered, since those outside are initially zero and are known beforehand to remain that way. This is seen in Fig. 12(c). Another advantage of matrix banding is that the whole problem is not needed in core storage at once. The elimination operations involve movement down the band from upper left to lower right, only affecting elements in

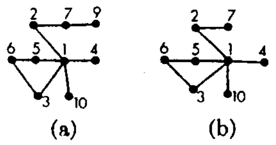


Fig. 11. Reduced graphs produced by two elimination steps applied to the graph of Fig. 3(c). (a) After elimination of old node 8. (b) After elimination of old node 8, then 9.

the immediate vicinity under consideration. As new lower rows must be read into core from auxiliary storage, upper rows are finished and can be released once and for all to auxiliary storage. The tower design problem of Fig. 7 provides an excellent application of this technique.

Another special case is that of weakly interconnected subnetworks, wherein separate disconnected parts would exist, were it not for a few branches. Fig. 13 shows a small 40-node example of this, where node numbering was done as follows:

1) each interconnection branch has one of its two terminal nodes eliminated at the end (nodes 38-40);
2) the remaining nodes of subsystem 1 are all taken first, then all those of subsystem 2, and finally those of subsystem 3.

This produces a nearly block-diagonal structure for $[A]$ as shown. Just as with matrix banding, the whole problem is not required simultaneously in core storage for solution. Rather, it can be worked on subsystem by subsystem, using only a diagonal block and its associated interconnection terms (columns and rows 38–40).

Consideration has been given to the problem of finding an absolutely optimal ordering, one that creates an absolute minimum number of nonzero terms upon elimination [16]. Yet computational difficulties are such as to make this impractical for realistic problems, at least at the present time.

No attempt at generality or completeness has been made in this discussion. Rather, the goal has been motivation and undertanding. It is hoped that the reader is in a position to seek added details elsewhere [1], [2], [14].

IX. The Triangular Factorization Process Itself

To systematize the Gaussian elimination process, one number is saved corresponding to each matrix location which ever becomes nonzero during the elimination. This value for position (i,j) shall be denoted by f_{ij} and is defined as follows:

if $j > i$ $f_{ij} = a_{ij}$, the value existing after the entire elimination process has been completed;
if $j = i$ $f_{ii} = 1/a_{ii}$, where a_{ii} is the diagonal value existing in row i just before it was divided to unity by the elimination process;
if $j < i$ $f_{ij} = a_{ij}$, the value of cell (i,j) just before it is zeroed by the elimination process.

These factors f_{ij} shall be collectively referred to as $[F]$, thus constituting the triangular factorization (or LU decomposition) of $[A]$. When the elimination is performed in a carefully chosen order, this then is an OTF of $[A]$.

Once $[F]$ has been calculated [14] shows that many problems are immediately solvable, including the following.

Original: Given b in (1), find X.
Reverse: Given X in (1), find b.
Transpose: Given b in (44), find X.
Hybrid Original: Given b_1 and X_2 of (45), find b_2 and X_1.
Hybrid Transpose: Given b_1 and X_2 of (46), find b_2 and X_1

$$[A]^t X = b \tag{44}$$

$$[A]\begin{pmatrix} X_1 \\ \hline X_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ \hline b_2 \end{pmatrix} \tag{45}$$

$$[A]^t\begin{pmatrix} X_1 \\ \hline X_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ \hline b_2 \end{pmatrix}. \tag{46}$$

Thus $[F]$ possesses not only the power of both $[A]$ and also $[A]^{-1}$, but also of their transposes and partitions at an arbitrary point. Each of these solutions requires $N + M$ multiplies and M adds,

$$\text{effort} = (N + M)\text{ mult} + (M)\text{ add}, \tag{47}$$

where

N the number of equations;

M the number of nonzero f_{ij} for $j \neq i$.

For example, Scheme 3 ordering of the water network o Fig. 6 yields $N = 62$, $M = 244$, making a total of 30 multiplies and 244 adds. Figures for much larger network are given in [14]. For an application that uses all but th second property simultaneously, see [15]. A three-variabl example is included in the Appendix.

An important computational variation of the conventional elimination procedure is the idea of elimination by rows (rather than columns). Instead of eliminating x_1 from equation 2 through N, then eliminating x_2 from equation through N, etc., row elimination proceeds as follows (se Fig. 14):

1) divide equation 1 by a_{11};
2) eliminate x_1 from equation 2 and then divide it by a_{22}
3) eliminate x_1 and x_2 from equation 3 and then divid it by a_{33};

· · · etc.

N) eliminate $x_1, x_2, \cdots, x_{N-1}$ from the last (Nth) equation and then divide it by $a_{N,N}$.

It can be shown that the factors $[F]$ so generated ar identical to those produced by column elimination. Whil of no special merit if square-array Fortran storage wer being used, this alteration is of great importance whe working with sparse matrices [14], [15]. A row can b formed and eliminated in dummy storage and then packe away into permanent storage in its natural order (increasing column numbers within any given row), never t be altered as the elimination proceeds.

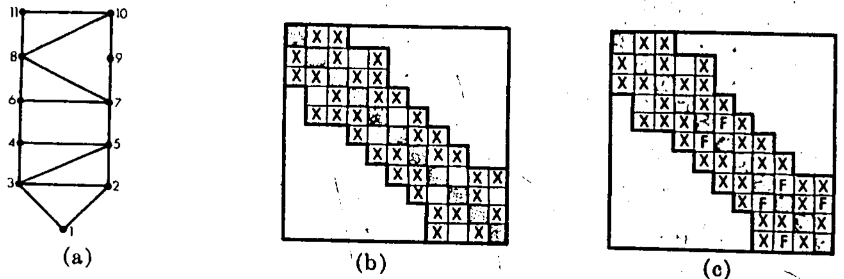


Fig. 12. Considerations of matrix banding for sample 11-node ladder network. (a) Ladder network with nodes numbered in direction of structure. (b) Resulting nonzero pattern of $[A]$ showing upper and lower banding lines. (c) Nonzero pattern after elimination (with F used to show fill-ins).

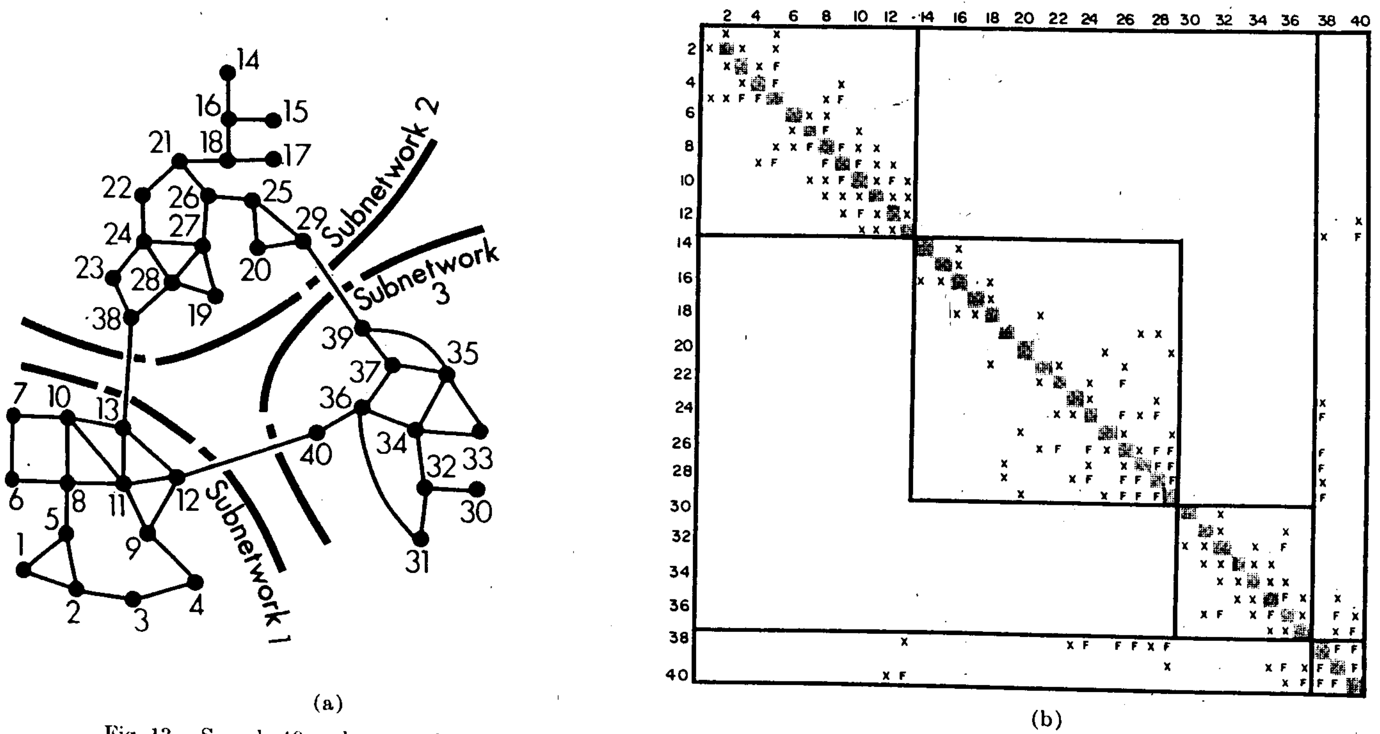


Fig. 13. Sample 40-node network with three weakly interconnected subnetworks identified. (a) Network graph with node numbers shown being the final ordering. (b) Resulting structure of $[A]$ with fill-in locations upon triangularization marked by F.

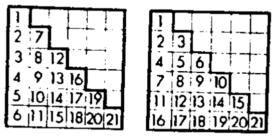


Fig. 14 Comparison between column and row elimination as to order of processing of matrix elements, for 6 × 6 matrix. (a) Order for column elimination. (b) Order for row elimination.

When $[A]$ is symmetric ($a_{kj} = a_{jk}$), then considerable saving is possible, both in triangularization effort and also in storage requirements. As shown in [14], only f_{ij} for $j \geq i$ are needed. The technique is illustrated for a three-variable problem in the Appendix, and [15] documents the programming procedures that completely ignore all lower triangle arithmetic operations during the triangularization. The water-flow and tower-design problems can exploit symmetry in this way.

A few words also should be said about updating the factors $[F]$, when $[A]$ is changed. Of course if the change is general, then one simply recomputes $[F]$ from the beginning; if changes can be confined to only lower rows, then only the end of the tirangularization need be redone. But if special, more selective changes occur, then the original $[F]$ can be retained, along with an appropriate correction vector. Two cases are of great practical importance:

1) suppose that one wants to replace one column of $[A]$ with an arbitrary new vector.
2) suppose one wants to alter only four elements of $[A]$ in positions (k,k), (k,m), (m,k), and (m,m).

The first case is fundamental to linear programming (Section VII-F), while the latter corresponds to removal of a branch in the nodal admittance matrix of an electric network, among other things (Section VII-A; [17] details an application). Both updating operations require just one linear equation solution using the old $[F]$ to find an appropriate correction vector, thus saving considerable effort.

Needless to say, the essence of sparsity programming requires only the handling of nonzero elements. Hence square-array Fortran storage is never used. Details of various efficient storage and calculation schemes are given elsewhere in the literature [1], [2], [5], [14], [15].

X. Conclusions

The ability to rapidly and accurately solve systems of sparse linear equations has had an enormous impact upon a number of engineering and mathematical problems, of which several have been presented. Probably the newest and most comprehensive statement as to what has been done to date is in [1]. Interested readers are encouraged to use this as a starting point for a more detailed investigation of the subject.

Appendix

Principles and terminology of the OTF solution technique shall be reviewed and summarized by means of small real examples. Consider the following system of equations $[A]X = b$, which is to be solved by Gaussian elimination:

$$\begin{aligned}(2)X_1 + (1)X_2 + (3)X_3 &= 6\\ (2)X_1 + (3)X_2 + (4)X_3 &= 9\\ (3)X_1 + (4)X_2 + (7)X_3 &= 14. \end{aligned} \tag{48}$$

For a computer solution, maximum efficiency dictates elimination by rows, rather than the more familiar column order. The successive reduced sets of equations are as follows:

1

$$\begin{aligned}(1)X_1 + (1/2)X_2 + (3/2)X_3 &= 3\\ (2)X_1 + (3)X_2 + (4)X_3 &= 9\\ (3)X_1 + (4)X_2 + (7)X_3 &= 14 \end{aligned} \tag{49}$$

2.

$$\begin{aligned}(1)X_1 + (1/2)X_2 + (3/2)X_3 &= 3\\ (2)X_2 + (1)X_3 &= 3\\ (3)X_1 + (4)X_2 + (7)X_3 &= 14 \end{aligned} \tag{50}$$

3 before

$$\begin{aligned}(1)X_1 + (1/2)X_2 + (3/2)X_3 &= 3\\ (1)X_2 + (1/2)X_3 &= 3/2\\ (3)X_1 + (4)X_2 + (7)X_3 &= 14 \end{aligned} \tag{51}$$

4

$$\begin{aligned}(1)X_1 + (1/2)X_2 + (3/2)X_3 &= 3\\ (1)X_2 + (1/2)X_3 &= 3/2\\ (5/2)X_2 + (5/2)X_3 &= 5 \end{aligned} \tag{52}$$

5

$$\begin{aligned}\underline{(1)}X_1 + (1/2)X_2 + (3/2)X_3 &= 3\\ (1)X_2 + (1/2)X_3 &= 3/2\\ \underline{(5/4)}X_3 &= 5/4. \end{aligned} \tag{53}$$

These steps are referred to as the "downward" or "elimination" operations. The solution x may be immediately determined by the so-called "back-substitution" operations using (53): X_3 is known and may be substituted into the second equation to find X_2; then the first equation yields X_1.

This solution process may be generalized by the "table of factors" or LU-decomposition idea. If one carefully reviews (48)–(53), he sees that $[F]$ in (54) completely summarizes the solution procedure, and permits finding X for any constant vector b (if $[A]$ remains unchanged).

$$[F] = \begin{bmatrix} f_{11} & f_{12} & f_{13}\\ f_{21} & f_{22} & f_{23}\\ f_{31} & f_{32} & f_{33}\end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 & 3/2\\ 2 & 1/2 & 1/2\\ 3 & 5/2 & 4/5\end{bmatrix}. \tag{54}$$

The array of f_{ij} in (54) has been called the table of factors; it guides the rapid solution for X in any of the problems:

$$[A]X = c \qquad \text{[(regular) network]} \tag{55}$$

$$[A]^tX = c \qquad \text{(transposed network)} \tag{56}$$

$$[A]\begin{pmatrix}X_a\\X_b\end{pmatrix} = \begin{pmatrix}c_a\\c_b\end{pmatrix} \qquad \begin{array}{l}\text{(hybrid network)}\\ X_a\ c_b \text{ unknown}\end{array} \tag{57}$$

$$[A]^t\begin{pmatrix}X_a\\X_b\end{pmatrix} = \begin{pmatrix}c_a\\c_b\end{pmatrix} \qquad \begin{array}{l}\text{(transposed hybrid network)}\\ X_a,\ c_b \text{ unknown.}\end{array} \tag{58}$$

Here c, c_a, and X_b are known vectors. $[A]$ is the coefficient matrix of (48), (57), and (58) involve comformable partitions of X and c and the problems will go by the names in parentheses. Derivation of the solution formulas for (55)–(58) is given in [14] and illustrated by the following examples. Note the first example is a mechanized version of (48)–(53) and should familiarize the reader with the notation. In these examples, we denote the solution vector at any stage of its development by $(y_1\, y_2\, y_3)^t = y$.

$$\begin{bmatrix} 3 & 1 & 1/2\\ 3 & 3 & 3\\ 3 & 3 & 3\end{bmatrix}$$

Example 1 [(Regular) Network]

When solving (55) by use of the table of factors $[F]$ in (54), successive steps appear as columns (left to right) in

c	1,1	2,1	2,2	3,1	3,2	3,3	2,3	1,3	1,2	X
6	3	3	3	3	3	3	3	3/2	1	1
9	9	3	3/2	3/2	3/2	3/2	1	1	1	1
14	14	14	14	5	5/4	1	1	1	1	1

The heading row indicates which element of the table of factors produces the vector below it. c is on the far left, X is on the far right, and the operation i,j is defined by

$$y_i = (y_i)(f_{ii}), \qquad \text{if } j = i \tag{59}$$

$$y_i = y_i - (f_{ij})(y_j), \qquad \text{if } j \neq i. \tag{60}$$

Example 2 (Transposed Network)

To solve (56) with $c = (9\ 11\ 18)^t$, the Example 1 operations are reversed:

c	1,2	1,3	2,3	3,3	3,2	3,1	2,2	2,1	1,1	X
9	9	9	9	9	9	6	6	2	1	1
11	13/2	13/2	13/2	13/2	4	4	2	2	2	2
18	18	9/2	5/4	1	1	1	1	1	1	1

$$y_i = (y_i)(f_{ii}), \qquad \text{if } j = i \tag{61}$$

$$y_j = y_j - (f_{ij})(y_i), \qquad \text{if } j \neq i. \tag{62}$$

Example 3 (Hybrid Network)

We shall solve (57) with the partition

$$\begin{pmatrix} X_a \\ c_b \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \hline c \end{pmatrix} \qquad \begin{pmatrix} c_a \\ X_b \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \\ \hline 0 \end{pmatrix}. \tag{63}$$

This constraint $X_3 = 0$ represents grounding the reference generator in the B-coefficient algorithm of [15]. X is found by the operations

c	1,1	2,1	2,2	set $X_3 = 0$	1,2	X
3	3/2	3/2	3/2	3/2	1	1
5	5	2	1	1	1	1
c=?	?	?	?	0	0	0

This is like Example 1, with certain simplifications. Downward operations for row 3 were ignored, since we knew the result had to give $X_3 = 0$. Then $X_3 = 0$ allowed ignoring (60) for the last column ($j = 3$).

Example 4 (Transposed Hybrid Network)

Assuming the partition of (63), X in (58) is found by the operations

c	1,2	set $X_3 = 0$	2,2	2,1	1,1	X
3	3	3	3	−1/2	−1/4	−1/4
5	7/2	7/2	7/4	7/4	7/4	7/4
?	?	0	0	0	0	0

This resembles Example 2, except one ignores operations (62) having $j = 3$, since X_3 is known and is manually set to zero; operations (62) for $i = 3$ could likewise be omitted (pass operations).

If $[A]$ is symmetric, then as well known [14], only f_{ij} for $j \geq i$ are needed for solutions. Of course $[A]^t = [A]$ means there are only two distinct problems—(55) and (57). For $[A]$ in (64), one finds $[F]$ to be given by (65):

$$[A] = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 4 \\ 3 & 4 & 8 \end{bmatrix} \tag{64}$$

$$[F] = \begin{bmatrix} 1/2 & 1/2 & 3/2 \\ 1 & 2/5 & 1 \\ 3 & 5/2 & 1 \end{bmatrix}. \tag{65}$$

Note that elements of $[F]$ satisfy (66), and solutions obtained using only f_{ij} for $j \geq i$ will be illustrated next.

$$(f_{ij})(f_{jj}) = f_{ji}, \qquad \text{for } j < i. \tag{66}$$

Example 5 (Symmetric Network)

For $c = (8, 9, 18)^t$ and the $[A]$ of (64), X in (55) is found by the operations

				diagonal						
c	1,2	1,3	2,3	1,1	2,2	3,3	2,3	1,3	1,2	X
8	8	8	8	4	4	4	4	5/2	2	2
9	5	5	5	5	2	2	1	1	1	1
18	18	6	1	1	1	1	1	1	1	1

Note all f_{ii} are applied in the middle and f_{ij} $(j > i)$ are each applied twice—once to the left of the diagonal operations and once to the right. The order of application on the right is the reverse of that on the left, and operation rules are

$$y_j = y_j - (f_{ij})(y_i) \text{ (when on left)} \tag{67}$$

$$y_i = (f_{ii})(y_i) \tag{68}$$

$$y_i = y_i - (f_{ij})(y_j) \text{ (when on right).} \tag{69}$$

In general, the order of application on the left is by rows: /1,2/1,3/$\cdots$/1,N/2,3/$\cdots$/2,N/$\cdots$etc.

Example 6 (Hybrid Symmetric Network)

With $[A]$ of (64) and the partition of (63), one solves (57) as

			diagonal			
c	1,2	set $X_3 = 0$	1,1	2,2	1,2	X
3	3	3	3/2	3/2	4/5	4/5
5	7/2	7/2	7/2	7/5	7/5	7/5
?	?	0	0	0	0	0

Here the f_{ij} were applied just as in Example 5, except all last-column ($j = 3$) operations were omitted on the left, X_3 was set to zero just before applying f_{ii}, and f_{33} and all last column f_{ij} on the right were omitted (pass operations).

References

[1] D. J. Rose and R. A. Willoughby Ed., *Sparse Matrices and Their Applications.* New York: Plenum, 1972.

[2] R. A. Willoughby Ed., in *Proc. Symp. Sparse Matrices and Their Applications*, IBM Watson Res. Cen., Yorktown Heights, N. Y., 1969.

[3] A. Ralston, *A First Course in Numerial Analysis.* New York: McGraw-Hill, 1965.

[4] G. E. Forsythe and C. B. Moler, *Computer Solution of Linear Algebraic Systems.* Englewood Cliffs, N.J.: Prentice-Hall, ch. 9, 12, 13.

[5] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Trans. Power App. Syst.*, vol. PAS-86, pp. 1449–1460, Nov. 1967.

[6] H. W. Dommel and W. F. Tinney, "Optimal power flow solutions," *IEEE Trans. Power App. Syst.*, vol. PAS-87, pp. 1866–1876, Oct. 1968.

[7] I. P. King, "An automatic reordering scheme for simultaneous equations derived from network systems," *Int. J. Num. Meth. Eng.*, vol. 2, pp. 523–533, 1970.

[8] H. G. Jensen, "Efficient matrix techniques applied to transmission tower design," *Proc. IEEE*, vol. 55, pp. 1997–2000, Nov. 1967.

[9] H. W. Dommel and N. Sato, "Fast transient stability solutions," *IEEE Trans. Power App. Syst.*, vol. PAS-91, pp. 1643–1650, July/Aug. 1972.

[10] O. I. Elgerd, *Control Systems Theory.* New York: McGraw-Hill, 1967.

[11] R. K. Mehra and R. E. Davis, "A generalized gradient method for optimal control problems with inequality constraints and singular arcs," *IEEE Trans. Automat. Contr.*, vol. AC-17, pp. 69–79, Feb. 1972.

[12] G. Hadley, *Linear Programming:* Reading, Mass.: Addison-Wesley, 1963.

[13] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Manag. Sci.*, vol. 3, no. 3, pp. 255–269, 1957.

[14] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proc. IEEE*, vol. 55, pp. 1801–1809, Nov. 1967.

[15] W. S. Meyer and V. D. Albertson, "Loss formula computation by optimal ordering techniques which exploit the sparsity of the network admittance matrix," in *1969 Midwest Power Symp.*, Univ. Minnesota, Minneapolis, Oct. 23, 1969.

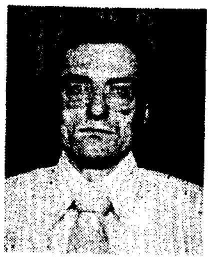
[16] E. C. Ogbuobiri, W. F. Tinney, and J. W. Walker, "Sparsity

directed decomposition for Gaussian elimination on matrices," in *Power Industry Computer Appl. Conf. Rec.*, 1967, pp. 215–225.
[17] N. M. Peterson, W. F. Tinney, and D. W. Bree, "Iterative linear ac power flow solution for fast approximate outage studies," presented at the 1972 Winter Power Meeting.

William F. Tinney (M'50–SM'63) was born in Portland, Oreg., on May 5, 1921. He received the B.S. and M.S. degrees from Stanford University, Stanford, Calif., in 1948 and 1949, respectively.

Since 1950 he has been employed by the Bonneville Power Administration, Portland, Oreg., and since 1955 has been working on power system computer applications. His special interests include sparse matrix methods, network analysis, and optimization. His present position is Head of the Methods Analysis Unit.

Mr. Tinney is a Registered Professional Engineer in the State of Oregon.



W. Scott Meyer (S'69–M'69) was born in Madison, Minn., in 1942. He received the B.S., M.S.E.E., and Ph.D. degrees, from the University of Minnesota, Minneapolis.

From 1969 to 1971, he taught power system engineering at the University of Minnesota as an Assistant Professor. He spent almost a year with Systems Control, Inc., before moving to his present position with the Bonneville Power Administration, Portland, Oreg. The efficient computer solution of power system network problems is his principal area of interest.

Structure and Design of Linear Model Following Systems

A. STEPHEN MORSE

Abstract—This paper investigates the problem of designing a compensating control for a linear multivariable system so that the impulse response matrix of the resulting closed-loop system coincides with the impulse response matrix of a prespecified linear model; this is the *model following problem*. A new formulation of the problem is developed, and necessary and sufficient conditions for a solution to exist are given. An upper bound is determined for the number of integrators needed to construct the compensating control and, if the open-loop plant in question possesses a left-invertible transfer matrix, this bound is shown to be as small as possible.

The relationship between the internal structure of a model following system and the model being followed is explained, and a description is given of the possible distributions of system eigenvalues which can be achieved while maintaining a model following configuration. This leads to a statement of necessary and sufficient conditions for the existence of a solution to the problem which results in a stable compensated system.

Introduction

THIS PAPER investigates the problem of designing a compensating control for a linear multivariable system so that the impulse response matrix of the resulting closed-loop system coincides with the impulse response matrix of a prespecified linear model. We call this the *model following problem* (MFP). Model following synthesis arises in connection with handling qualities design [1], [2] and model-reference adaptive control [3], and it is applicable to a number of standard design problems in linear multivariable control.

The model following problem has been studied in one form or another for at least 15 years. In early investigations, Kavanagh [4] developed preliminary results in transfer matrix terms, and Tyler [1] studied a closely related problem using optimal regulator theory. Recent investigations by Erzberger [5], Wolovich [6], Wang and Desoer [7], and Moore and Silverman [8] have been primarily concerned with problem solvability under various hypotheses. In this paper we present new results regarding both solvability and system structure which clarify and extend the previously cited work.

In Section I we develop a new formulation of MFP in simple algebraic terms. The formulation is completely general and avoids restrictive hypotheses concerning plant and model. In Section II we state a necessary and sufficient condition for MFP to be solvable; the condition is explicit, easy to check, and leads to a constructive procedure for finding a solution, should one exist.

The design of model following systems often requires the use of integrators to construct the compensating control. In Section II we determine an upper bound for the number of integrators needed. If the open-loop plant in

Manuscript received July 3, 1972; revised March 23, 1973. Paper recommended by L. M. Silverman, Past Chairman of the IEEE S-CS Linear Systems Committee. This work was supported by the United States Air Force Office of Scientific Research under Grant NO-72-2211.

The author is with the Department of Engineering and Applied Science, Yale University, New Haven, Conn. 06520.