Copyright © 1971 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 50, No. 4, April, 1971
Printed in U.S.A.

Statistical Circuit Design:

Nonlinear Circuits and Statistical Design

By I. A. CERMAK and MRS. D. B. KIRBY §

(Manuscript received November 25, 1970)

Despite recent advances in the speed of digital computers and in numerical algorithms for the solution of differential equations, the evaluation of the dynamic response of nonlinear circuits is still too slow to permit Monte Carlo tolerance analysis. However, the performance of many nonlinear circuits can be evaluated on a static basis. One example is a D/A converter built with devices much faster than the converter's cycle time. Algorithms now exist that produce the static or equilibrium solution of such networks in seconds. This paper deals with these algorithms and the associated techniques that have been embodied in a program for the Monte Carlo tolerance analysis of nonlinear, "dc," circuits.

I. INTRODUCTION

To date, most tolerance analysis of circuits has been in the frequency domain, as this series of articles indicates. The need for nonlinear analysis arises not only for large signal circuits but also for small signal ac circuits where device model parameters vary with bias. Recent advances in the speed of digital computers and numerical algorithms have made possible the analysis of circuits with nonlinear behavior. Large signal, or time domain analysis of nonlinear circuits, however, is still such a comparatively slow process that Monte Carlo methods are out of the question. Enough algorithmic innovations have been achieved in the static analysis of nonlinear circuits that Monte Carlo methods can be applied to a wide class of nonlinear problems. DC in the sense used in this paper implies that the dynamic behavior of the nonlinear devices is fast in relation to the response of the rest of the circuit. There are, in general, three types of circuits that fall into this class.

(*i*) Circuits that are essentially dc, such as operational amplifiers, power supplies, and those circuits used as examples in this paper.

(*ii*) Circuits that can be subdivided into dc and ac blocks where the nonlinear behavior of the circuit is present essentially only in the dc part. An example of this type of circuit is the *Touch-Tone®* oscillator, the analysis of which is described in this series.[1] This class of circuits is very similar to the first.

(*iii*) Circuits designed for small signal applications may be analyzed in the frequency domain. Realistic modeling of devices, however, introduces changes in the small signal model parameters for different bias points. If the bias circuits vary randomly, then nonlinear dc analysis is required in the ac tolerance analysis loop.

This paper deals with some of the methods required for the efficient analysis of nonlinear circuits in a dc sense. The techniques and algorithms to be described have been embodied in a computer program which performs Monte Carlo tolerance analysis of nonlinear dc circuits as well as dc and transient analysis. Some of the more critical implementation aspects are described.

II. DESIGN OF A NONLINEAR TOLERANCE ANALYSIS PROGRAM

2.1 *Problems Involved*

Until recently tolerance analysis, even nonlinear tolerance analysis, has been simple in concept. Circuits were manufactured using *discrete* components which generally had independent statistical behavior; transistors and diodes were expensive items and were used sparingly. Hence, a large amount of analysis could be done without a complex software package.

Integrated circuits have opened a whole host of new problems in this area. Circuits designed today typically employ large numbers of transistors (since transistors are as cheap as resistors), posing many problems in their modeling, simulation, and solution. Probably the biggest dilemma in the design of a nonlinear tolerance analysis program is the minuscule past experience to draw upon as to what analysis is required, what to do with the analysis results once they are obtained, and how to interpret them.

Some of the special problems that arise in nonlinear tolerance analysis are:

(*i*) Parameters tend to be statistically correlated. This implies that many new output features have to be present in the software.

(*ii*) There is a very wide range of circuit problems that will have to be solved. This poses special difficulties and restrictions on the methods of analysis, as will be seen in the following sections.

(*iii*) Statistical data for circuits manufactured today are not yet available or are available in limited quantities;[2] manufacturing processes vary from day to day and parameter correlations, aging data, etc., are all important to the analysis.

2.2 *Criteria To Be Met*

One of the most important criteria to be met in the design of a nonlinear tolerance analysis program is that it be easy to use. Some of the characteristics implied by this, both for the users of the program and for the writers of it, are:

(*i*) The program must be humanly engineered to have a simple, clear, and easily learned input language. This applies not only to the network description but also to the description of statistical data and to the command structure. The trend has been for engineers to personally use available computer tools rather than work through intermediaries, and the program itself should present as few obstacles as possible. In addition, the output capabilities must include data reduction schemes so that insight is gained at a glance.

(*ii*) The program must be designed to be flexible enough so that it can be changed easily. Past experience has shown that a general-purpose analysis program will undergo many changes and, in fact, will probably never reach a static condition. This means the program must be written in modular form, a characteristic that very often degrades efficiency. Modularity, however, implies ease of maintenance, upgrading, and debugging.

(*iii*) The program must be portable because of wide demand. This implies that it be written in some high level language, such as FORTRAN, with possibly a very small number of critical routines written in Assembly language for efficiency.

In addition to ease of use, an important criterion is, of course, economy and reliability. The program must be very efficient if it is to be useful. Solution times for each statistical design must be measured in seconds to make the analysis practical at all, and possibly in milliseconds if sophisticated features such as performance contours and large scale sensitivities are to be included.[3] Until recently, one would have been happy to get one solution to a nonlinear circuit; today we are faced with obtaining hundreds and perhaps thousands of these solutions in a reasonable time.

Various algorithms for solution of the nonlinear equations that arise from circuit simulations have been described in the literature; however, they are all basically variations on the Newton-Raphson scheme and are in general not suitable as they stand. The majority of the schemes converge in the order of tens of iterations, if they converge at all. Since solution times for each iteration are generally proportional to the cube of the number of variables, many iterations for each statistical solution preclude their use in a program such as this. Recent breakthroughs, however, in the Newton-Raphson solution of nonlinear circuits and careful implementation of these methods permit solution times short enough (of the order of one second) so that meaningful tolerance analysis is possible.

III. NUMERICAL ALGORITHMS AND THEIR IMPLEMENTATION

3.1 *Problem Formulation*

Given a network topology, there are various ways to write equations describing the behavior of the network. Some examples are nodal equations, loop equations, Branin's[4] mixed mesh/cut-set equations and the state-space formulation, which in the dc case has come to be known as the "normal form."[5] We have adopted the normal equation formulation for the following reasons:

(*i*) The reduced set of equations produces, in general, a small system that has to be solved iteratively, by effectively separating the linear and nonlinear aspects of the problem.

(*ii*) Implementation of various analysis techniques for equations in their normal form is straightforward.

(*iii*) The normal form of the equations can be generated extremely fast (see below) in a manner competitive with the most efficient sparse matrix techniques available today.

Some other formulation (such as nodal equations), coupled with sparse matrix techniques may be more efficient for circuits containing a large number of junctions compared with the number of linear resistors. The formulation employed here, however, allows straight forward implementation of various convergence schemes such as the nonlinear transformation described in Section 3.3.3. In addition, it is felt that the normal formulation is the most practical for small-to-medium size circuits with a significant number of linear resistors (including those in the device models). The operational amplifier example in this paper probably represents the limiting practical size of circuit for this formulation. Most of the methods described here are also applicable to any other equation formulation.

3.2 *Generation of Equations in the Normal Form*

Consider any circuit consisting of current and voltage sources, diodes, transistors and resistors. From the network topology and network parameters, the large signal behavior of the network is characterized by a system of equations in the "normal form," viz.,

$$[A][\mathbf{V}] + [B][\mathbf{U}] + [N(\mathbf{V})] = 0 \tag{1}$$

where

- $[\mathbf{V}]$ is the vector of all device junction voltages and is the set of independent variables to be found;
- $[\mathbf{U}]$ is the vector of independent voltage and current sources in the network;
- $[A]$ and $[B]$ are coefficient matrices dependent on the network resistances; and
- $[N(\mathbf{V})]$ is a vector of functions dependent on the nonlinear properties of the network.

For computer simulation, the network devices are characterized by the Ebers–Moll model[6] where the functional form of $N(V)$ is

$$N(V) = I_D = I_S\,[\exp\,(\theta V) - 1]$$

where I_S is intercept current, and θ is dependent on temperature. Notice that the formulation (1) isolates the linear part of the network from the nonlinear part and that the nonlinear behavior can be characterized by a *vector* quantity.

The set of equations given by (1) must be generated for each statistical design, since the $[A]$ and $[B]$ matrices are dependent on resistor values. The implication of this, of course, is that a very fast algorithm is needed for equation generation.

To completely characterize network behavior, another set of equations is required which relates any requested network currents or voltages to the junction voltages found from (1). If $[\mathbf{Z}]$ is the vector of user-requested outputs (element currents and voltages), then

$$[\mathbf{Z}] = [C][\mathbf{V}] + [D][\mathbf{U}] \tag{2}$$

where again $[C]$ and $[D]$ are coefficient matrices dependent on resistor values and must be recalculated for each statistical design.

The basis for the formulation of the A, B, C and D matrices is mainly topological in nature. From the network incidence matrix, $[T]$, and the choice of a network tree as described below, the fundamental loop and cut-set matrices ($[F]$ and $[-F]^T$, respectively) are derived.[4]

The incidence matrix, T, has elements 0 and ± 1 and dimensions $n \times b$ for a network with $(n + 1)$ nodes and b elements. Let $\mathbf{I}$ be a vector of element currents which is partitioned into tree branch currents, $\mathbf{I}_t$, and link currents, $\mathbf{I}_1$. Then,

$$[T][\mathbf{I}] = [-\mathcal{I}, F]\begin{bmatrix} \mathbf{I}_t \\ \mathbf{I}_1 \end{bmatrix} = 0, \tag{3}$$

$$\mathbf{I}_t = F\mathbf{I}_1 \quad \text{and} \quad \mathbf{V}_1 = -F^T\mathbf{V}_t . \tag{4}$$

Equations (3) and (4) are expressions of Kirchoff's Current and Voltage Laws where $\mathcal{I}$ is the identity matrix and the vector of element voltages, $\mathbf{V}$, is partitioned like $\mathbf{I}$ into $\mathbf{V}_t$ and $\mathbf{V}_1$. $[F]$ has dimensions $n \times (b - n)$, and its elements are 0 and ± 1.

It is desired to place all voltage sources and device junctions in the tree and all current sources in links. Assuming this is possible, the columns of $[T]$ are arranged in the following preference order (given also is the notation to be used for tree and link current and voltage):

	I	V
voltage sources	I_E	E
device junctions	I_D	V_D
resistors—tree	I_R	V_R
—link	I_G	V_G
current sources	J	V_J

The incidence matrix, $[T]$, can be quickly reduced to the form $[-I, F]$ by gaussian elimination, from which the fundamental loop matrix is defined. The reduction process favors the above top elements for inclusion in the tree.

Now partition $[F]$ as follows:

$$\begin{array}{c|cc} & G & J \\ \hline E & F_{11} & F_{12} \\ D & F_{21} & F_{22} \\ R & F_{31} & F_{32} \end{array}$$

and from equation (4) write

$$\mathbf{I}_D = F_{21}I_{\mathbf{G}} + F_{22}\mathbf{J}, \tag{5}$$

$$I_{\mathbf{R}} = F_{31}\mathbf{I}_G + F_{32}\mathbf{J}, \tag{6}$$

$$\mathbf{V}_G = -F_{11}^T\mathbf{E} - F_{21}^T\mathbf{V}_D - F_{31}^T\mathbf{V}_R. \tag{7}$$

Let $[R]$ and $[G]$ be defined as diagonal matrices whose elements are tree resistor values and link conductance values, respectively. Then, from equations (6) and (7),

$$\begin{bmatrix} \mathbf{I}_R \\ \mathbf{V}_G \end{bmatrix} = \begin{bmatrix} R^{-1} & 0 \\ 0 & G^{-1} \end{bmatrix}\begin{bmatrix} \mathbf{V}_R \\ \mathbf{I}_G \end{bmatrix}$$
$$= \begin{bmatrix} 0 \\ -F_{21}^T \end{bmatrix}[\mathbf{V}_D] + \begin{bmatrix} 0 & F_{32} \\ -F_{11}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{E} \\ \mathbf{J} \end{bmatrix} + \begin{bmatrix} 0 & F_{31} \\ -F_{31}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{V}_R \\ \mathbf{I}_G \end{bmatrix}$$

or

$$\begin{bmatrix} R^{-1} & -F_{31} \\ F_{31}^T & G^{-1} \end{bmatrix}\begin{bmatrix} \mathbf{V}_R \\ \mathbf{I}_G \end{bmatrix} = \begin{bmatrix} 0 \\ -F_{21}^T \end{bmatrix}[\mathbf{V}_D] + \begin{bmatrix} 0 & F_{32} \\ -F_{11}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{E} \\ \mathbf{J} \end{bmatrix}. \tag{8}$$

Define

$$\begin{bmatrix} R^{-1} & -F_{31} \\ F_{31}^T & G^{-1} \end{bmatrix}^{-1} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} = [H]. \tag{9}$$

Then

$$\begin{bmatrix} \mathbf{V}_R \\ \mathbf{I}_G \end{bmatrix} = [H]\begin{bmatrix} 0 \\ -F_{21}^T \end{bmatrix}[\mathbf{V}_D] + [H]\begin{bmatrix} 0 & F_{32} \\ -F_{11}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{E} \\ \mathbf{J} \end{bmatrix}. \tag{10}$$

Substituting $\mathbf{I}_G$ into equation (5) results in equation (1), namely

$$\mathbf{I}_D = N(\mathbf{V}_D) = [A][\mathbf{V}_D] + [B][\mathbf{U}]$$

or

$$A\mathbf{V} + B\mathbf{U} - N(\mathbf{V}) = 0.$$

Also, from F and $-F^T$, any network current or voltage can be extracted and the $[\mathbf{Z}]$ vector of equation (2) calculated. In general,

$$[\mathbf{Z}] = [C'][\mathbf{V}] + [D'][\mathbf{U}] + [P]\begin{bmatrix} \mathbf{V}_R \\ \mathbf{I}_G \end{bmatrix} \tag{11}$$

where C', D' and P are matrices whose elements are 0, ± 1 obtained from selective rows and columns of F and $-F^T$. Substituting equation (10) into equation (11) yields equation (2).

The time-consuming task in the calculation of the A, B, C, D matrices is in finding $\begin{bmatrix} \mathbf{V}_R \\ \mathbf{I}_G \end{bmatrix}$ as expressed by equation (10). This calculation requires one matrix inversion and two matrix multiplications.

The number of rows and columns of H^{-1}, as seen by equation (9), is equal to the number of resistors in the network. For a completely characterized network, including parasitics, this number can easily approach 80 or 100. The number of multiplications required to invert an nth order matrix is n^3, or 10^6 for our example! Clearly, the implementation of generating the equations of (1) and (2) is critical.

Fortunately, the matrices that evolve from this formulation have special properties which can be advantageous. These are:

(*i*) The fundamental loop matrix, F, contains only 0, ± 1 entries and is sparse. From experience with a wide range of problems, this matrix is 20 to 50 percent dense (ratio of nonzero entries to total number of entries). Because of these properties, any matrix multiplication involving F or its partitions can be performed by additions and subtractions rather than by multiplications. The operation of addition is at least 3 to 4 times faster than multiplication on most digital computers. Also, the number of additions and subtractions to be performed in multiplying F by A, where A is an $n_1 \times n_2$ matrix, is equal to the product of n_2 and the number of nonzero entries of the F matrix involved.

(*ii*) In addition to F being sparse, other matrices as in equation (10), are sparse with predictably placed submatrices being identically 0.

(*iii*) The special form of H^{-1} of equation (9) allows profitable application of block inversion methods. This warrants further discussion. Consider equation (9):

$$\begin{bmatrix} R^{-1} & -F_{31} \\ F_{31}^T & G^{-1} \end{bmatrix}^{-1} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}.$$

Both R^{-1} and G^{-1} are *diagonal* matrices and the dimension of H is $n \times n = (n_R + n_G)^2$ where n_R is the number of tree resistors in the network and n_G is the number of link resistances. A method of block inversion is chosen based on the min (n_R, n_G).

Case 1: $n_R \leqq n_G$	Case 2: $n_G > n_R$
$H_{11} = [R^{-1} + F_{31}GF_{31}^T]^{-1}$,	$H_{22} = [G^{-1} + F_{31}^TRF_{31}]^{-1}$,

$$H_{12} = H_{11}F_{31}G, \qquad H_{21} = -H_{22}F_{31}^{T}R,$$
$$H_{21} = -GF_{31}^{T}H_{11}, \qquad H_{12} = RF_{31}H_{22},$$
$$H_{22} = G - GF_{31}H_{12}. \qquad H_{11} = R + RF_{31}H_{21}.$$

In case 1, the inversion of an $n_R \times n_R$ matrix (H_{11}) is required; for case 2 the matrix to be inverted is of order n_G (H_{22}). To invert H could take n^3 multiplications. The worst case for this formulation is $n_R = n_G = n/2$, requiring $n^3/8$ multiplications. The calculation of the remaining submatrices of H involve additions, subtractions or, at worst, multiplication by diagonal matrices.

3.3 *Solution of the dc Equations*

3.3.1 *The Newton–Raphson Method and Its Limitations*

The standard Newton–Raphson solution of equation (1) is obtained as follows: For an arbitrary estimate of $\mathbf{V}$, say $\mathbf{V}^k$, equation (1) is not satisfied exactly, and there results:

$$[A][\mathbf{V}^k] + [B][\mathbf{U}] + [N(\mathbf{V}^k)] = [\mathbf{R}^k] \tag{12}$$

where the vector $[\mathbf{R}^k]$ is termed the residual vector and is, in this case, a measure of the current imbalances in the circuit that result from insisting that $[\mathbf{V}] = [\mathbf{V}^k]$. The superscript k refers to the iteration number. Successive estimates of $[\mathbf{V}]$ are formed as:

$$[\mathbf{V}^{k+1}] = [\mathbf{V}^k] + [\mathbf{P}^k] \tag{13}$$

where the step vector $[\mathbf{P}^k]$ is obtained by solution of

$$-[J(\mathbf{V}^k)][\mathbf{P}^k] = [\mathbf{R}^k]. \tag{14}$$

In equation (14), $[J(\mathbf{V}^k)]$ is the Jacobian of the system (1), viz.,

$$[J(\mathbf{V}^k)] = [A] + [N'(\mathbf{V}^k)]. \tag{15}$$

The iterations are terminated whenever either the step vector $[\mathbf{P}]$ and/or the residual vector $[\mathbf{R}]$ is sufficiently close to zero.

Straightforward application of the above iterative method to the system (1) results in several difficulties:

(*i*) If the starting guess $[\mathbf{V}^0]$ is not close to the solution, then there typically results exponential overflow in the nonlinear terms $I_S(e^{\theta V} - 1)$, since the full Newton–Raphson step may be too large in the positive direction. This may be overcome in several ways, the simplest probably being to reduce the step,[7] viz.,

$$[\mathbf{V}^{k+1}] = [\mathbf{V}^k] + S[\mathbf{P}^k] \tag{16}$$

where S is a step size, $0 < S \leqq 1$ and is chosen so that

$$\sum_i (R_i^{k+1})^2 < \sum_i (R_i^k)^2. \tag{17}$$

(*ii*) The method does not necessarily converge. Reliability can be greatly increased by the use of parameter-stepping techniques,[8] as described in the next section.

3.3.2 *Increasing Reliability Through Source-Stepping*

Newton–Raphson methods typically converge whenever the starting guess is close to the solution point. This fact was first utilized by D. F. Davidenko[9] and subsequently by various authors.[10–12] Implementation of the method in nonlinear circuit analysis can be accomplished as follows[7]: For any system (1), one accurate solution is always known, namely, $[\mathbf{V}] = 0$ if $[\mathbf{U}] = 0$. Hence, if the sources $[\mathbf{U}]$ are brought to their full value in small increments, convergence to each intermediate point is more likely.

The strategy for stepping can take on various forms (see, for example, Ref. 12); the approach taken here involves source-stepping only when necessary. If no convergence is obtained after a fixed number of iterations with the sources on full, the sources are reduced to one-half their value and solution is again attempted. If necessary, the sources are progressively reduced until convergence is obtained at some intermediate source value, whereupon solution is again attempted with the sources on full. If convergence is not obtained, the sources are again reduced to a value midway between full and the last point at which convergence was obtained. The process is repeated until convergence at the full source value is obtained.

3.3.3 *Nonlinear Scalar Transformation*

Reliability and speed of the Newton-Raphson method can also be greatly increased through the use of a nonlinear scalar transformation of variables. The transformation is an extension[13] of the notion of "charge-state-variables"[14,15] based on a suitable definition of the "capacitance" of a junction, viz.,

$$q = \int_0^v [1 + \theta K I_S \exp(\theta V)]\, dV \tag{18}$$

where (18) is a *scalar* equation. The potentials are considered to be a function of q, i.e., $V = V(q)$, and the Newton-Raphson iteration is performed in the q-space rather than the V-space.

This is accomplished by noting that

$$[\mathbf{R}(q)] = [\mathbf{R}(V)] \tag{19}$$

and

$$[J(q)] = [J(V)][S(V)]^{-1} \tag{20}$$

where $S(V)$ is a diagonal matrix with elements of the form,

$$S_{ii} = 1 + \theta K_i I_S \exp{(\theta V_i)}. \tag{21}$$

The Newton–Raphson step vector may be written in terms of q as

$$[\mathbf{P}(q)] = -[J(q)]^{-1}[\mathbf{R}] \tag{22}$$

which results in

$$[\mathbf{P}(q^k)] = [S(V^k)][\mathbf{P}(V^k)]. \tag{23}$$

What is required, then, is to transform the usual Newton–Raphson step vector, where now

$$[\mathbf{q}^{k+1}] = [\mathbf{q}^k] + [\mathbf{P}(q^k)]. \tag{24}$$

Since equation (23) represents a *scalar* transformation on the individual q_i, equation (24) may be written in terms of the individual elements V_i of the vector $[\mathbf{V}]$ as

$$V_i^{k+1} + K_i I_{S_i} \exp{(\theta V_i^{k+1})} = V_i^k + K_i I_{S_i} \exp{(\theta V_i^k)} + S_i(V_i^k) P_i(V_i^k) \tag{25}$$

where

$$S_i(V_i^k) \equiv 1 + \theta K_i I_{S_i} \exp{(\theta V_i^k)}$$

and $P_i(V_i^k)$ is the ith element of the usual Newton–Raphson step in V. The transform parameter K_i remains to be determined.

Solution of equation (25) for each V_i^{k+1} gives the new estimate of the vector $[\mathbf{V}]$, given the standard Newton–Raphson step vector $[\mathbf{P}(V)]$, which may be obtained in the usual manner. Note that (25) is a scalar equation which may be solved very simply by Newton's method, as discussed further below.

Empirical studies have shown that the speed of the iterative process is dependent on the value of K, which has the effect of "straightening out" the exponential characteristics at the expense of "warping" the linear parts of the solution space. It was determined that a good choice for K is

$$K_i = \frac{10}{\theta I_{S_i} \exp(\theta V_i^k)}, \qquad 10^2 \leq K \leq 10^4 \tag{26}$$

which, it is believed, results in a near-optimum solution sequence.[13] The value of K varies from iteration to iteration and is different for every variable.

Once the value of K_i is determined, solution of (25) is accomplished as follows:
Set

$$D_i = V_i^k + K_i I_{S_i} \exp(\theta V_i^k) + [1 + \theta K_i I_{S_i} \exp(\theta V_i^k)] P_i(V_i^k) \tag{27}$$

where K_i is picked according to equation (26). To obtain the new estimate, V_i^{k+1}, set

$$Y_i \equiv V_i^{k+1} \tag{28}$$

and establish an iterative procedure for solution of Y as

$$Y_i^{n+1} = Y_i^n - \frac{Y_i^n + K_i I_{S_i} \exp(\theta Y^n) - D_i}{1 + \theta K_i I_{S_i} \exp(\theta Y_i^n)} \tag{29}$$

where the superscript n indicates the iteration number in the *scalar* Newton–Raphson subloop. This is done for each element of [**V**].

A first estimate, Y^0, is formed as follows.
Set

$$Z_i = -\frac{1}{\theta} \ln [K_i I_{S_i}]; \tag{30}$$

then

$$\left.\begin{aligned} \text{if} \quad D_i \leqq Z_i + 0.08, \quad Y_i^0 &= D_i, \\ \text{and if} \quad D_i > Z_i + 0.08, \quad Y_i^0 &= \frac{1}{\theta} \ln D_i + Z_i. \end{aligned}\right\} \tag{31}$$

The iterative procedure in Y_i is terminated whenever

$$\frac{|Y_i^{n+1} - Y_i^n|}{|Y_i^n|} < \epsilon \tag{32}$$

where ϵ is some suitably small constant, such as 10^{-6}.

The procedure outlined above, combined with the source stepping described in Section 3.3.2, provides an extremely powerful and rapid algorithm for solution of circuits with exponential nonlinearities. Solution for most circuits is accomplished in very few iterations (fewer than 10), so that in combination with the efficient generation of the equations, the overall method is competitive with analysis of *linear* systems.

There is, however, one main limitation that must be dealt with as a special case. It arises from the formulation of the circuit equations and is discussed below.

3.3.4 *Treatment of Junction Cut-sets*

Whenever the circuit under consideration contains cut-sets of junctions, the matrix $[A]$ of the system (1) becomes singular. This causes severe instability problems in the solution whenever the junctions in the cut-set are not (or barely) conducting. The problem, and its solution, are best illustrated by an example.

Consider the simple circuit in Fig. 1. The system of equations describing this circuit is:

$$\begin{bmatrix} -\frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & -\frac{1}{R} \end{bmatrix} \begin{bmatrix} V_{D1} \\ V_{D2} \end{bmatrix} + \begin{bmatrix} -\frac{1}{R} \\ -\frac{1}{R} \end{bmatrix} [E] - \begin{bmatrix} I_{S1}[\exp(\theta V_{D1}) - 1] \\ I_{S2}[\exp(\theta V_{D2}) - 1] \end{bmatrix} = 0, \tag{33}$$

for which the Jacobian is

$$[J] = \begin{bmatrix} -\frac{1}{R} - \theta I_{S1} \exp(\theta V_{D1}) & -\frac{1}{R} \\ -\frac{1}{R} & -\frac{1}{R} - \theta I_{S2} \exp(\theta V_{D2}) \end{bmatrix} \tag{34}$$

which, by inspection, is seen to be extremely ill-conditioned whenever $\theta I_{S2} \exp(\theta V_{D2}) \ll 1/R$. The conditioning of the system can be improved by subtracting the second equation from the first in (33) to yield:

$$\begin{bmatrix} -\frac{1}{R} & -\frac{1}{R} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_{D1} \\ V_{D2} \end{bmatrix} + \begin{bmatrix} -\frac{1}{R} \\ 0 \end{bmatrix} [E] + \begin{bmatrix} I_{S1}[\exp(\theta V_{D1}) - 1] \\ I_{S2}[\exp(\theta V_{D2}) - 1] - I_{S1}[\exp(\theta V_{D1}) - 1] \end{bmatrix} = 0, \tag{35}$$

for which the Jacobian is

$$J = \begin{bmatrix} -\frac{1}{R} - \theta I_{S1} \exp(\theta V_{D1}) & -\frac{1}{R} \\ \theta I_{S1} \exp(\theta V_{D1}) & -\theta I_{S2} \exp(\theta V_{D2}) \end{bmatrix}. \tag{36}$$

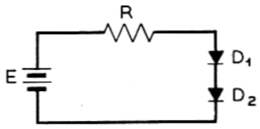

Fig. 1—Junction cut-set example.

The Jacobian shown in (36) can be scaled very simply to produce a well-behaved system.

In the above example, the treatment of the cut-set problem was determined by inspection. Junction cut-sets can be found in any network by forming the so-called "L-tree"[16] of the network and forming the fundamental loop (or cut-set) matrix. The L-tree preference order dictates that junctions be made links with all other types of elements retaining their order. The treatment is simple, once junction/ current source cut-sets are found. For each cut-set, one row of the $[A]$ and $[B]$ matrices corresponding to one of the offending junctions is set to zero. The appropriate additions and subtractions in the vector $[N(V)]$ are performed, and the system scaled. A similar procedure is used by Shichman.[14]

3.4 *Parameter Perturbation and Monte Carlo Analysis*

In addition to methods of solution of the nonlinear circuit equations, a method of statistically perturbing circuit parameters is required. The two are then combined in an overall strategy.

3.4.1 *Parameter Perturbation*

It is desired to generate correlated random variables with a fixed range, corresponding to the tolerance set by the user. Since large arrays of correlated numbers have to be generated, a parametric representation is used that correlates random variables by the use of "pivots."[1] In addition, a linear additive statistical model is used which ensures that parameters stay within tolerance. Generation of fixed interval correlated random numbers can be illustrated by a simple example:

Assume x_0, x_1 and x_2 independent, each from a distribution of mean m and variance $\sigma_{x_i}^2$. Two correlated random variables, y_1 and y_2 can be generated as

$$\begin{aligned} y_1 &= (1 - |\lambda_1|)x_1 + \lambda_1 x_0, \\ y_2 &= (1 - |\lambda_2|)x_2 + \lambda_2 x_0, \end{aligned} \tag{37}$$

where λ_1, λ_2 are tracking coefficients and x_0 is serving as a "pivot." Note that if the x_i are in the interval $(-1, 1)$, then the y_i will be in this interval also. The correlation factor ρ_{12} is easily determined,

$$\rho_{12} = \frac{E(y_1y_2) - E(y_1)E(y_2)}{\sigma y_1 \sigma y_2} \tag{38}$$

resulting in

$$\rho_{12} = \frac{\lambda_1\lambda_2\sigma_{x_0}^2 - \lambda_1\lambda_2 m_0^2}{\{[(1 - |\lambda_1|)^2\sigma_{x_1}^2 + \lambda_1^2\sigma_{x_0}^2][(1 - |\lambda_2|)^2\sigma_{x_2}^2 + \lambda_2^2\sigma_{x_0}^2]\}^{\frac{1}{2}}} \tag{39}$$

with

$$\begin{aligned} E(y_1) &= (1 - |\lambda_1|)m_1 + \lambda_1 m_0, \\ E(y_2) &= (1 - |\lambda_2|)m_2 + \lambda_2 m_0, \\ \sigma_{y_1}^2 &= (1 - |\lambda_1|)^2\sigma_{x_1}^2 - \lambda_1^2\sigma_{x_0}, \\ \sigma_{y_2}^2 &= (1 - |\lambda_2|)^2\sigma_{x_2}^2 + \lambda_2^2\sigma_{x_0}^2. \end{aligned}$$

For the special case of $\lambda_1 = \lambda_2 = \lambda$, $\sigma_{x_0}^2 = \sigma_{x_1}^2 = \sigma_{x_2}^2 = \sigma^2$ and $m_0^2 = 0$.

$$\rho_{12} = \frac{\lambda^2}{1 - 2|\lambda| + 2\lambda^2}. \tag{40}$$

The model actually used allows correlation to two pivots (or other parameters) as

$$y_i = (1 - |\lambda_i| - |\eta_i|)x_i + \lambda_i x_{01} + \eta_i x_{02} \tag{41}$$

and normalized random factors for the various parameters generated as

$$r_i = 1 + t y_i \tag{42}$$

where t is the tolerance.

The independent variables, x_i, are generated as follows: A piecewise-linear probability density *shape* is supplied by the user in the form of a table in arbitrary units for distributions other than uniform, normal or log normal which are "built in." This table is scaled and extended to include the cumulative density function which is a piecewise-quadratic on the interval $(-1, 1)$. A random variable from a uniform distribution is generated by a Tausworthe random number generator[17] and transformed to the desired distribution by quadratic interpolation from the cumulative density function. Parameter perturbations are then calculated according to equations (41) and (42). Nominal (design) values of parameters are taken to be the *median* value of their corresponding distributions.

3.4.2 *Monte Carlo Analysis*

Parameter perturbations and analysis are combined in a standard tolerance analysis loop. The nominal solution is taken as the starting guess for each random design and the solutions along with the parameter values stored on disk for later post-processing. Work is in progress to allow various other procedures (such as tuning or adjustment) in the loop.

IV. IMPLEMENTATION

The techniques and methods described in the preceding section allow fairly efficient nonlinear statistical design. The details of their implementation can, however, spell the difference between success and failure in a general-purpose program, as well as the input/output features and structure of the program. Described below are some of the more critical aspects of the design and structure of a general-purpose nonlinear tolerance analysis program for IBM series 360 computers.

4.1 *Memory Allocation*

As in any large program, there exists a conflict between efficient use of memory and speed of execution. In addition to the program itself, memory is required for the various coefficient matrices, the variables and outputs, as well as the various circuit parameters such as element values, model parameters, statistical data and topological information.

The program itself, written largely in FORTRAN-IV, is overlayed with major divisions separating (*i*) input and initial handling of data, (*ii*) generation of the various topological matrices, (*iii*) generation of the equations, (*iv*) analysis, and (*v*) output. Communication among the various overlays is via a labeled common structure. Data at input time is handled largely via fixed-dimension arrays which are then partly overwritten by run-time data during execution. Run-time data is stored in a dynamic linear array with pointers used for addressing. This data includes such arrays as the A, B, C and D matrices, the variables, the Jacobian, various tabled quantities, and so on. In this way, efficient use is made of fast-access memory in that only data that is needed is stored. At the same time, this structure does not degrade the speed of execution.

4.2 *Algorithms*

The IBM 360 series is not well suited for applications such as described here and special care is required in the implementation of the various algorithms in addition to standard good programming practices. All floating-point computation in the program is done in double precision (8 bytes). It was found that some of the matrix-handling subroutines had to be written in Assembly language in order to achieve any efficiency at all. For example, one routine that multiplies two matrices one of which has only 0, ± 1 entries could be speeded up by a *factor* of 5–10 by direct coding in Assembly language. Assembly language coding is also required for the equation solution and matrix inversion routines.

Matrix sparsity is used to advantage in the solution of simultaneous equations as in equation (14) by column reordering[18] and stability is preserved by row-pivoting. Ill-conditioning is detected by monitoring the magnitude of the smallest pivot used in the gaussian elimination process.

Other small details in the programming are equally critical. It is of utmost importance in an application such as this to preserve as much numerical precision as possible since many mathematical steps are required before a solution is attained. For example, the analysis requires evaluation of quantities such as $I_S[\exp(\theta V) - 1]$. For values of V close to zero, a call to the exponential routine and subsequent subtraction of the constant 1 can yield an inaccurate result. For this situation, a series evaluation of the function is used.

4.3 *Data Reduction and Display*

Some care has to be taken in handling the voluminous data produced by the program. Experience has shown that it is almost impossible to determine beforehand how to analyze and display the output data, at least until a preliminary investigation of the results is available. In addition, any extensive Monte Carlo analysis of most circuits is likely to be expensive (despite the efficient algorithms) so that it becomes worthwhile developing a flexible post-processing scheme. For these reasons, the philosophy adopted here is the following:

(*i*) All output data, including parameter values of every "important" or expensive analysis, is stored in a permanent disk file for later access. This raw data may be later reduced or displayed in whatever way the user sees fit.

(*ii*) Extensive post-processing capability is available immediately following the analysis, accessed via the input language to the program. This facility allows scatter plots and histograms of any outputs or parameters to be produced on-line, including the printing out of extremal cases and various statistics. The facility also allows separation of the data by temperature or any other parameter.

(*iii*) Hard-copy printout of *all* data is produced whether or not the user requires it. In this way an expensive run is not lost even if the disk file is destroyed. Experience has also shown that sometimes it is desired to inspect the raw data weeks after initial analysis. The hard-copy output provides this facility.

V. SAMPLE PROBLEMS

Two sample problems are presented, both of which Monte Carlo analysis has verified to be good designs. The transistor model[2] used in both examples is shown in Fig. 2. In both problems silicon resistors (including the base and collector resistances of the transistor model) were allowed to vary $\pm 15\%$ within a normal distribution truncated at $\pm 3\sigma$, with resistor values tracking to $\pm 5\%$. This is illustrated in the scatter diagram shown in Fig. 3. The intercept currents were picked from a log-normal distribution ranging from $1/4 I_{S_0}$ to $4 I_{S_0}$ and correlated by a factor of 0.85. The βs of the devices were picked from a triangular distribution, typically ranging from $1/2\beta_0$ to $2\beta_0$ and correlated by a factor of 0.3.

The first example, a constant current source, used in a D/A converter, is shown in Fig. 4. Resistors R1 through R4 are thin film tantalum resistors with a tolerance of $\pm 2\%$, with the exception of R4 which was assigned a tolerance of $\pm 0.5\%$. R5 represents the load and was allowed to vary $\pm 25\%$. Analysis of this circuit verified that the output current is essentially insensitive to all parameters except the power supplies and R4. A scatter plot of the output current versus the value of R4 is shown in Fig. 5 for the case where the power supplies are held fixed. Figure 6 shows a histogram of the values of output current with all parameters varying. For this case, the twelve-volt supply was assigned

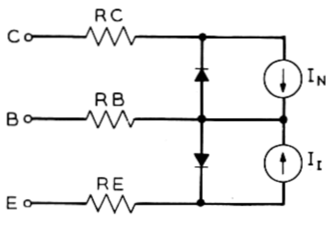

Fig. 2—Transistor model.

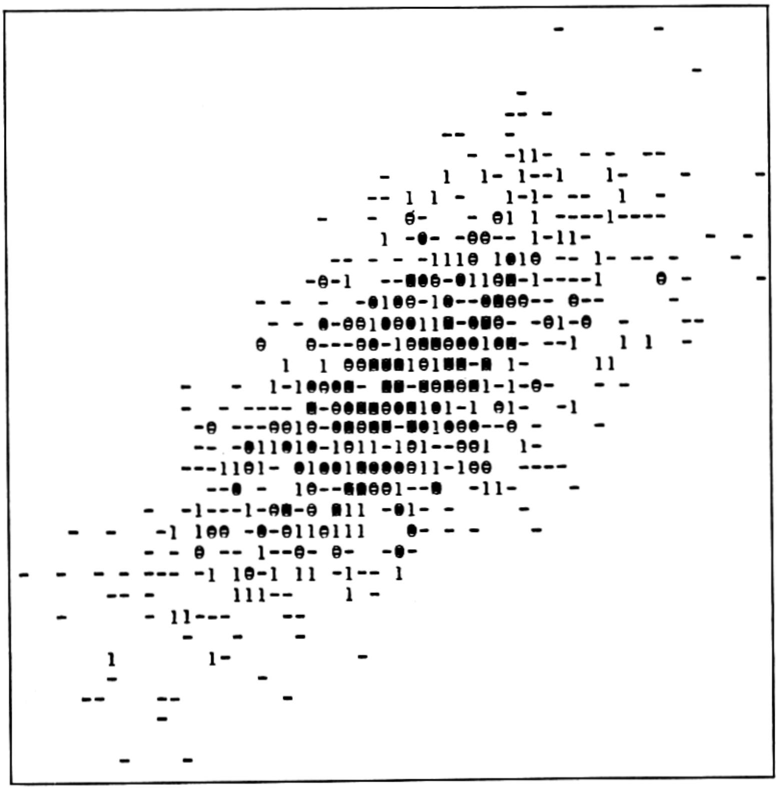

Fig. 3—Resistor correlation, range of each axis ±15%.

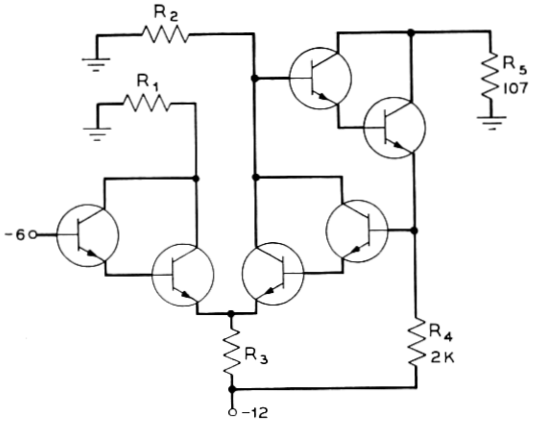

Fig. 4—Constant current source.

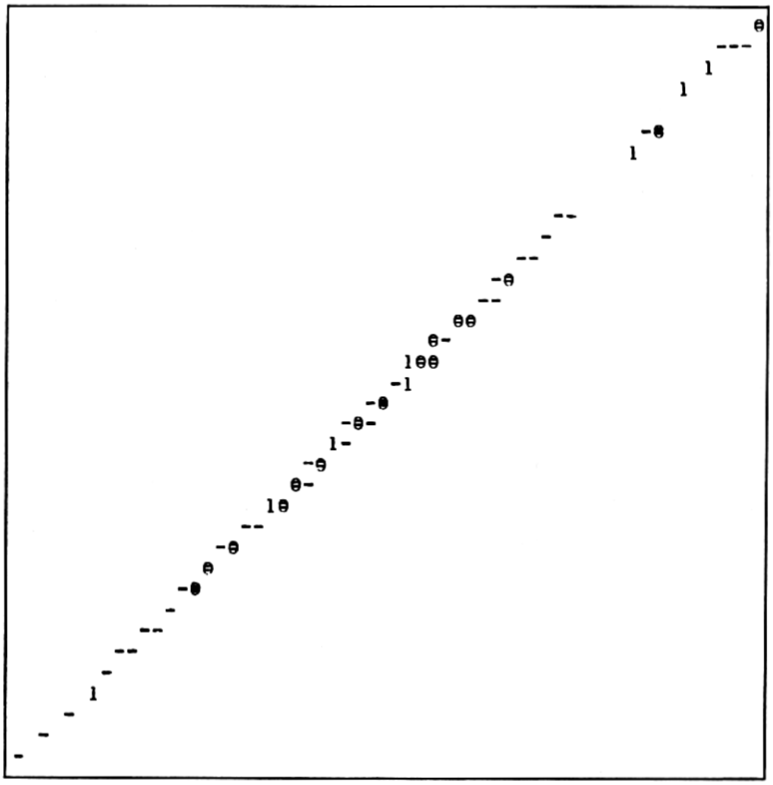

Fig. 5—Current source output (vertical axis) versus R4 (normalized).

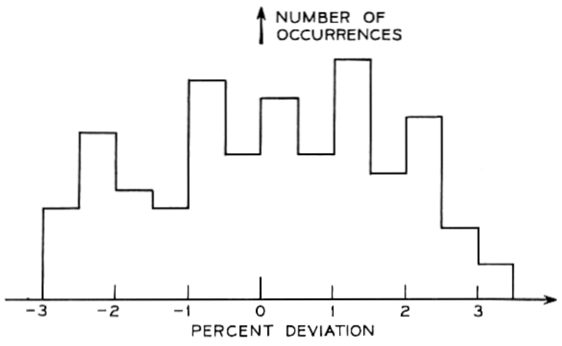

Fig. 6—Current source output with all parameters varying.

to a tolerance of ±2%, with the six-volt supply tracking to ±1%. Analysis time for this example was approximately one second CPU time per random design on an IBM 360/65 computer. The great majority of this time represents overhead in the form of subroutine calls, various bookkeeping operations and writing data on disk and the printer. Solutions were carried to approximately seven digits of accuracy with each random design requiring typically 3–5 iterations.

The second example shown in Fig. 7 is a silicon integrated operational amplifier designed at Bell Laboratories. Of interest in this example is the output offset voltage with the inputs grounded. In addition, minimum and maximum (worst case) currents in the collectors of T13, T14 and T16, as well as dc gain were sought. This circuit represents a rather large simulation with 48 junctions, 76 resistors, 79 nodes and 127 branches. Figure 8 shows a histogram of the output voltage at room temperature with the power supplies held fixed. The range of offsets for this circuit was found to be slightly higher than predicted by approximate hand analysis. Analysis time for this

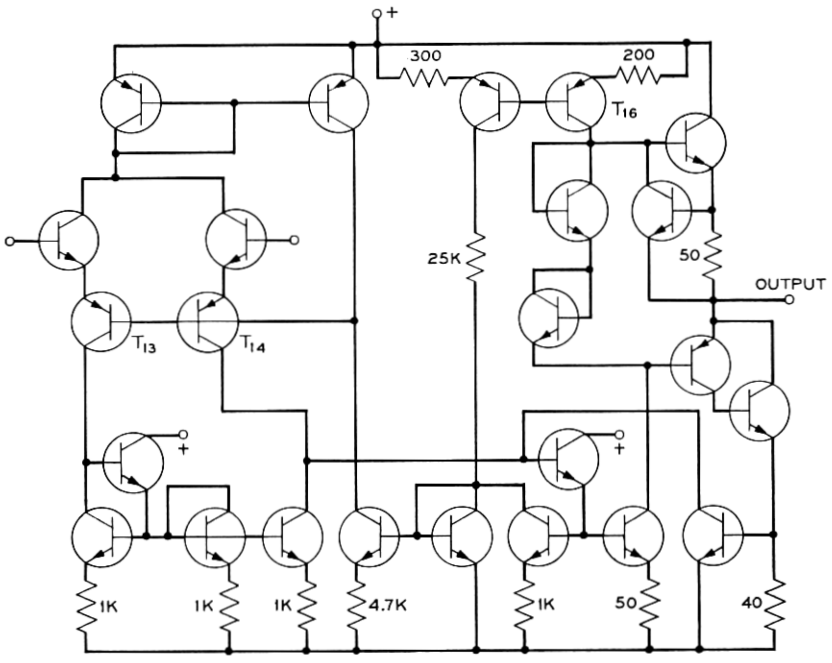

Fig. 7—Operational amplifier.

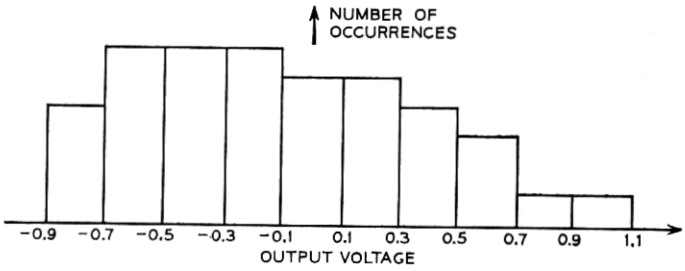

Fig. 8—Operational amplifier output offset voltage.

example was 8 seconds CPU time per statistical design, some of this time being bookkeeping overhead. It is expected that this time will be cut down considerably with some reprogramming. As in the previous example, typically 3-5 iterations were required per statistical design for a seven-digit accuracy in junction voltages.

VI. CONCLUSIONS

Analysis techniques and programming considerations have been presented that allow reasonably economical tolerance analysis of nonlinear "dc" circuits of reasonable size. Many of the ideas presented have evolved from past experience with ac tolerance analysis and will most probably be modified as experience with nonlinear statistical design becomes more plentiful. It is already apparent, however, that the trend in the near future will be to larger scale integration of circuits for which some of the present analysis techniques will likely be inadequate. Research is in progress in analysis methods capable of coping with large and complex circuits, as well as methods to make the present techniques even more efficient.

VII. ACKNOWLEDGMENTS

The authors are indebted to Mrs. H. D. Rovegno for her programming efforts in the postprocessor, and to Messrs. B. J. Karafin and R. G. Olsen for their helpful discussions.

REFERENCES

1. Balaban, P., Karafin, B. J., and Snyder, Mrs. D. B., "A Monte Carlo Tolerance Analysis of the Integrated, Single-Substrate, RC, *Touch-Tone*® Oscillator," B.S.T.J., this issue, pp. 1263–1291.

2. Logan, J., "Characterization and Modeling for Statistical Design," B.S.T.J., this issue, pp. 1105–1147.
3. Butler, E. M., "Large Change Sensitivities for Statistical Design," B.S.T.J., this issue, pp. 1209–1224.
4. Branin, F. H. Jr., "Computer Methods of Network Analysis," Proc. IEEE, *55*, No. 11 (November 1967), pp. 1787-1801.
5. Sandberg, I. W., "Theorems on the Analysis of Nonlinear Transistor Networks," B.S.T.J., *59*, No. 1 (January 1970), pp. 95–114.
6. Ebers, J. J., and Moll, J. L., "Large Signal Behavior of Junction Transistors," Proc. IRE, *42*, No. 12 (December 1954), pp. 1761–1772.
7. Broyden, C. G., "A Class of Methods for Solving Nonlinear Simultaneous Equations," Math. Comp., *19*, No. 92 (October 1965), pp. 577–93.
8. Cermak, I. A., "DC Solution of Nonlinear State-Space Equations in Circuit Analysis," IEEE Trans. on Circuit Theory, *CT 18*, No. 2 (March 1971).
9. Davidenko, D. F., "On a New Method of Numerical Solution of Systems of Nonlinear Equations," Doklady Akad. Nauk. SSSR (N.S.), *88*, No. 4 (1953), pp. 601-602.
10. Freudenstein, F., and Roth, B., "Numerical Solution of Systems of Nonlinear Equations," J. ACM, *10*, No. 4 (1963), pp. 550–556.
11. Kizner, W., "A Numerical Method for Finding Solutions of Nonlinear Equations," SIAM Journal, *12*, No. 2 (June 1964), pp. 424–428.
12. Broyden, C. G., "A New Method of Solving Simultaneous Equations," Computer Journal, *12*, No. 1 (1969), pp. 94–99.
13. Cermak, I. A., to be published.
14. Sandberg, I. W., "Some Theorems on the Dynamic Response of Nonlinear Transistor Networks," B.S.T.J., *48*, No. 1 (January 1969), pp. 35–54.
15. Shichman, H. "The Analysis of a Class of Nonlinear Networks," Ph.D. Thesis, Stevens Institute of Technology, Hobroken, N. J., September 1970.
16. Sedore, S. R., "SCEPTRE: A Program for Automatic Network Analysis," IBM Journal of Research and Development, November 1967, p. 627.
17. Payne, W. H., "FORTRAN Tausworthe Pseudorandom Number Generator," Comm. ACM, *13*, No. 1 (January 1970), p. 57.
18. Tewarson, R. P., "On the Product Form of Inverses of Sparse Matrices," SIAM Rev., *8*, No. 3 (July 1966), pp. 336–342.