IEEE TRANSACTIONS ON CIRCUIT THEORY, VOL. CT-18, NO. 1, JANUARY 1971

Elements of Computer-Aided Circuit Analysis

WILLIAM J. McCALLA, MEMBER, IEEE, AND DONALD O. PEDERSON, FELLOW, IEEE

Abstract—A survey is made of the principal techniques, procedures, and routines that are used in present programs for computer-aided circuit analysis. Programs (simulators) are reviewed and selected features compared for the four major classes of circuit analysis: linear dc and ac, nonlinear dc, nonlinear transient, and linear pole zero.

I. Introduction

SEVERAL recent books and papers have cataloged and compared many existing and proposed computer-aided circuit analysis programs [1]–[11]. This paper is a survey of the principal techniques used in existing programs for the analysis of the four important circuit areas: linear dc and ac, nonlinear dc, nonlinear transient, and linear pole zero. The considerations brought out are based on over four years experience in using, modifying, and writing more than twenty programs.

This survey is organized as follows: first an overview is made comparing nodal and mixed or state-variable methods of formulating the circuit equations. Linear dc and ac analysis programs are then considered with emphasis placed on techniques for solving systems of linear equations. Attention is next given to nonlinear dc analysis programs and the iterative solution of nonlinear algebraic equations. The extension to nonlinear transient analysis follows with an introduction to several numerical integration routines used in the solution of the nonlinear differential equations. Finally, four linear pole-zero circuit analysis techniques and programs are considered. Throughout the paper a number of specific references on computer-aided circuit analysis and design are cited. In addition, several general computer-aided circuit analysis references are included [12]–[17].

II. Circuit Equation Formulation

Virtually all presently available circuit analysis programs start from the same point, an elemental circuit description supplied to the program via keyboard, punched cards, or an interactive graphic display console. This description of circuit elements and their interconnections is converted by the programs into a set of circuit equations. Of interest here are the two major approaches that are now used in formulating these equations. The first approach is the familiar nodal analysis while the second is the mixed or state-variable approach which derives from Bashkow's A-matrix formulation [18].

Manuscript received August 11, 1970. This work was supported in part by the Joint Services Electronics Program, under Grant AFOSR-68-1488, and in part by the U. S. Army Research Office, Durham, N. C., under Contract DAHCO4-67-C-0031.

The authors are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Calif. 94720.

The implementations of these two approaches within the present generation of circuit analysis programs seem to have had a common origin in the transistor analysis program (TAP) [19]–[20] developed by IBM. TAP, though reasonably effective, was short lived and had such severe limitations that it was never made available outside of IBM. From TAP there evolved three programs: ECAP [21], which used nodal analysis, and NET-1 [22] and PREDICT [23], which both used a state-variable approach.

ECAP performs linear dc, linear ac, and piecewise-linear transient analyses and is still very much in use. It is considered in more detail shortly. NET-1, a nonlinear transient analysis program, is available in the IBM 7090/94 assembly languages MAP and FAP. Therefore, this program cannot be used simply in computer systems having monitor systems. NET-1 did, however, influence the development of the program CIRCUS [24]. This program is considered in more detail in the following paragraphs. PREDICT is also a nonlinear transient analysis program but includes no built-in device models and is incapable of automatically performing multiple analyses for several different sets of element values. In addition, it requires separate analysis runs for steady-state (dc) and transient solutions and suffers from a weak numerical integration routine. To overcome these and other deficiencies, the SCEPTRE [25] program, which is considered in the following paragraphs, was developed.

Nodal Analysis Formulation

For both linear and nonlinear (or piecewise-linear) circuit problems, nodal equations may be generated by little more than inspection. The relative simplicity of this approach contrasts sharply with the manipulations required by the state-variable approach. As an example, consider the case of a linear circuit. The nodal equations are of the form

$$\boldsymbol{Y}\boldsymbol{v} = \boldsymbol{i} \tag{1}$$

where $\boldsymbol{Y}$ is the nodal admittance matrix, $\boldsymbol{v}$ is the vector of node voltages to be found, and $\boldsymbol{i}$ is a vector representing independent source currents. The term y_{ii} in $\boldsymbol{Y}$ represents the sum of the admittances of all the branches connected to node i; y_{ij} is the negative of the sum of the admittances of all branches connecting node i and node j; and i_k is the sum of all source currents entering node k. Thus if a resistor of value R connects nodes 5 and 7, $1/R$ is added to y_{55} and y_{77} and subtracted from y_{57} and y_{75}, while if a current source of strength I is directed from node 2 to node 3, I is subtracted from i_2 and added to i_3.

Voltage sources are usually handled in one of two ways. The first is to require that every voltage source appear in series with a resistor so that the source may be transformed to a Norton equivalent current source. The second approach is a generalization of the first but does not depend upon a series resistor. The approach is most easily introduced in terms of grounded voltage sources. The nodal equations are first assembled including all elements other than voltages sources. Columns of the admittance matrix corresponding to grounded voltage source nodes are then multiplied by the value of the source and the result is subtracted from both sides of (1). If the current through the voltage source is required, it may be treated as an unknown in place of the known voltage. Otherwise the equation representing the sum of the currents at the source node may be considered redundant and dropped, thereby reducing the number of unknowns. Floating voltage sources and controlled sources may be handled by similar column and row operations.

For nonlinear analysis, equations can be formulated in a manner similar to that used in the linear case. The equations may be written symbolically as

$$Y(\boldsymbol{v}, t) = \boldsymbol{i}(t). \tag{2}$$

The interpretation of (2) is that each equation represents a summing of current contributions at a node. It should be noted, however, that as now formulated the nonlinearities are restricted to be voltage rather than current dependent. Fortunately, semiconductor devices such as junction diodes and bipolar and field-effect transistors are of the voltage-controlled category, and the above restriction is usually not severe.

As mentioned previously, ECAP is based on nodal analysis formulation. Additional nonlinear analysis programs using nodal analysis include TRAC [26], MTRAC [27], SYSCAP [28], and BIAS-3 [29], while linear programs based on a nodal approach include NATFREQS [30], [31], and from LISA [32], [33] the programs ACCA, POLY, and TRFN.

State-Variable Analysis Formulation

The primary reason for using the state-variable approach is that it yields a set of first-order linear or nonlinear differential equations in a minimal set of unknowns. Deriving a set of such equations explicitly seemed a desirable objective for use with earlier numerical integration algorithms. As is brought out in the following paragraph, this is no longer essential.

From an elementary viewpoint, the state-variable formulation proceeds as follows [34]: a proper tree consisting of all voltage sources, as many capacitive branches and as few inductive branches as possible, and no current sources is selected. The state variables are chosen to be the capacitive tree-branch voltages or charges and inductive tree-link currents or fluxes. A fundamental cut-set equation is constructed for each capacitive tree branch and a fundamental loop equation for each inductive tree link. Independent sources are withdrawn as separate terms in the equations and the equations are normalized with respect to the element values associated with each state variable.

For a linear circuit, the preceding procedure results in a matrix equation of the form [35]

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{3}$$

where $\boldsymbol{A}$ is a coefficient matrix relating the state vector $\boldsymbol{x}$ to its derivative $\dot{\boldsymbol{x}}$ and $\boldsymbol{B}$ is a coefficient matrix coupling the effects of the independent source vector $\boldsymbol{u}$. Any other desired network variables can be expressed in terms of the state variables and independent sources. For the linear case, this procedure results in an equation of the form

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \tag{4}$$

where $\boldsymbol{y}$ is a vector of desired output variables and $\boldsymbol{C}$ and $\boldsymbol{D}$ are again coefficient matrices.

For the nonlinear case, (3) and (4) are of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x}, \boldsymbol{u}, t) \tag{5}$$

$$\boldsymbol{y} = \boldsymbol{C}(\boldsymbol{x}, \boldsymbol{u}, t). \tag{6}$$

An alternative description is based on extracting the nonlinear portion of the circuit from the linear portion. This allows the state equations (5) and (6) to be written in the following equivalent form [15]:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{B}'\boldsymbol{w} \tag{7}$$

$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} + \boldsymbol{D}'\boldsymbol{w} \tag{8}$$

$$\boldsymbol{g}(\boldsymbol{w}) = \boldsymbol{E}\boldsymbol{x} + \boldsymbol{F}\boldsymbol{u} + \boldsymbol{F}'\boldsymbol{w}. \tag{9}$$

Again, $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{B}', \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{D}', \boldsymbol{E}, \boldsymbol{F}$, and $\boldsymbol{F}'$ are coefficient matrices, while $\boldsymbol{x}$ and $\boldsymbol{u}$ remain the state variables and independent source vector, respectively. The vector $\boldsymbol{w}$ represents the controlling or independent variable set associated with the nonlinearities $\boldsymbol{g}(\boldsymbol{w})$. In the state-variable approach, no assumption is made regarding voltage or current control.

While the equation formulation procedure outlined above applies in general, details vary from program to program. Both CIRCUS and SCEPTRE employ variations of Bryant's method [36], [37] as modified by Wilson and Massena [38]. Other programs relying on the state-variable approach include CORNAP [39], [40], BELAC [41], and CIRPAC [42], [43]. In all cases, the formulation procedures entail extensive matrix manipulations. The bookkeeping associated with these manipulations usually requires considerable amounts of core storage. Loops of capacitors and voltage sources and cut-sets of inductors and current sources are accommodated by additional manipulations of the state equations.

III. Linear DC and AC Analysis

The first of the four circuit analysis areas to be treated is linear dc and ac analysis. The desired outputs are the various voltages and currents within a circuit, possibly as a function of frequency. As virtually all programs of this category use a nodal analysis formulation, it is assumed that the circuit equations to be solved have the form of (1). The major techniques presently used to solve such systems of linear equations include Gaussian elimination, pivoting, *LU* factorization, and sparse matrix methods. These techniques are outlined in the following paragraphs and are followed by a comparison of the features of several available analysis programs.

A dc analysis can be considered the special case of an ac analysis performed at zero frequency. However, such an approach is usually not used for two reasons: first, if the formulation proceeds on a nodal admittance basis, the infinite admittance represented by an inductor at zero frequency requires special consideration. Second, where only resistive elements and independent and controlled sources are present, the circuit equations contain only real coefficients and hence may be solved on a computer using only real variables. If complex variables are used, the computation time may be increased by as much as a factor of 4 because multiplication and division (both considered long operations as compared with the short operations of addition and subtraction) require more time than for long operations with real variables.

Gaussian Elimination

With the distinction in mind that the system of nodal equations (1) involves only real variables in the dc case and complex variables in the ac case, the solution of (1) can be written

$$\boldsymbol{v} = \boldsymbol{Y}^{-1}\boldsymbol{i} \tag{10}$$

where $\boldsymbol{Y}^{-1}$ is the inverse of the nodal admittance matrix. Computationally, the efficiency of this approach can be evaluated in terms of the number of long operations required. It can be shown that the number of long operations required to invert an $n \times n$ matrix is n^3, while the number of long operations required to compute its product with a vector of dimension n is n^2. Thus the total number of long operations required by this approach is

$$n^3 + n^2.$$

The number of long operations required to obtain a solution to (1) can be reduced by a factor of 3 using Gaussian elimination [44].

This procedure consists of two steps. The first step consists of converting the nodal admittance matrix $\boldsymbol{Y}$ to an equivalent upper triangular matrix, i.e., a matrix with all zero elements below the diagonal. The second step which is referred to as back substitution, consists of solving the nth equation containing only v_n for v_n, the $n-1$st equation for v_{n-1} in terms of v_n, etc. Gaussian elimination is illustrated in (11) for a third-order system

$$\begin{array}{l} \mathscr{E}_1^{(1)}: \\ \mathscr{E}_2^{(1)}: \\ \mathscr{E}_3^{(1)}: \end{array} \begin{bmatrix} y_{11}^{(1)} & y_{12}^{(1)} & y_{13}^{(1)} \\ y_{21}^{(1)} & y_{22}^{(1)} & y_{23}^{(1)} \\ y_{31}^{(1)} & y_{32}^{(1)} & y_{33}^{(1)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} i_1^{(1)} \\ i_2^{(1)} \\ i_3^{(1)} \end{bmatrix} \tag{11}$$

where the superscript 1 indicates the initial system of equations. As indicated previously, the unknown v_1 is eliminated from equations $\mathscr{E}_2^{(1)}$ and $\mathscr{E}_3^{(1)}$ by subtracting $(y_{21}^{(1)}/y_{11}^{(1)})\mathscr{E}_1^{(1)}$ from $\mathscr{E}_2^{(1)}$ and $(y_{31}^{(1)}/y_{11}^{(1)})\ \mathscr{E}_1^{(1)}$ from $\mathscr{E}_3^{(1)}$. Symbolically this transformation can be represented by the equations

$$\begin{aligned} \mathscr{E}_1^{(2)} &= \mathscr{E}_1^{(1)} \\ \mathscr{E}_2^{(2)} &= \mathscr{E}_2^{(1)} - \frac{y_{21}^{(1)}}{y_{11}^{(1)}}\mathscr{E}_1^{(1)} \\ \mathscr{E}_3^{(2)} &= \mathscr{E}_3^{(1)} - \frac{y_{31}^{(1)}}{y_{11}^{(1)}}\mathscr{E}_1^{(1)} \end{aligned} \tag{12}$$

which yields the system

$$\begin{array}{l} \mathscr{E}_1^{(1)}: \\ \mathscr{E}_2^{(2)}: \\ \mathscr{E}_3^{(2)}: \end{array} \begin{bmatrix} y_{11}^{(1)} & y_{12}^{(1)} & y_{13}^{(1)} \\ 0 & y_{22}^{(2)} & y_{23}^{(2)} \\ 0 & y_{32}^{(2)} & y_{33}^{(2)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} i_1^{(1)} \\ i_2^{(2)} \\ i_3^{(2)} \end{bmatrix}. \tag{13}$$

Finally, the unknown v_2 is eliminated from $\mathscr{E}_3^{(2)}$ by subtracting $(y_{32}^{(2)}/y_{22}^{(2)})\mathscr{E}_2^{(2)}$ from $\mathscr{E}_3^{(2)}$, thus obtaining

$$\begin{aligned} \mathscr{E}_1^{(3)} &= \mathscr{E}_1^{(2)} = \mathscr{E}_1^{(1)} \\ \mathscr{E}_2^{(3)} &= \mathscr{E}_2^{(2)} \\ \mathscr{E}_3^{(3)} &= \mathscr{E}_3^{(2)} - \frac{y_{32}^{(2)}}{y_{22}^{(2)}}\mathscr{E}_2^{(2)} \end{aligned} \tag{14}$$

resulting in the triangularized system

$$\begin{array}{l} \mathscr{E}_1^{(1)}: \\ \mathscr{E}_2^{(2)}: \\ \mathscr{E}_3^{(3)}: \end{array} \begin{bmatrix} y_{11}^{(1)} & y_{12}^{(1)} & y_{13}^{(1)} \\ 0 & y_{22}^{(2)} & y_{23}^{(2)} \\ 0 & 0 & y_{33}^{(3)} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} i_1^{(1)} \\ i_2^{(2)} \\ i_3^{(3)} \end{bmatrix}. \tag{15}$$

This completes the first step in the Gaussian elimination procedure. Back substitution is now performed to obtain the final solution as follows:

$$\begin{aligned} v_3 &= \frac{i_3^{(3)}}{y_{33}^{(3)}} \\ v_2 &= \frac{(i_2^{(2)} - y_{23}^{(2)}v_3)}{y_{22}^{(2)}} \\ v_1 &= \frac{(i_1^{(1)} - y_{13}^{(1)}v_3 - y_{12}^{(1)}v_2)}{y_{11}^{(1)}}. \end{aligned} \tag{16}$$

Careful enumeration shows that for an nth order system the number of long operations required by Gaussian elimination is

$$\frac{n^3}{3} + n^2 - \frac{n}{3}.$$

LU Transformation

A modification of Gaussian elimination which is useful when more than one source or right-hand-side vector $\boldsymbol{i}$ is to be considered is the *LU* transformation [44]. This procedure consists of partitioning $\boldsymbol{Y}$ into an upper triangular matrix $\boldsymbol{U}$ and a lower triangular matrix $\boldsymbol{L}$ (usually with ones on the diagonal) such that

$$\boldsymbol{LU} = \boldsymbol{Y}. \tag{17}$$

This technique is illustrated shortly. The resulting system is solved in two stages. First,

$$\boldsymbol{Uv} = \boldsymbol{L}^{-1}\boldsymbol{i} = \boldsymbol{i}^* \tag{18}$$

and secondly,

$$v = U^{-1}i^*. \tag{19}$$

In this case since both L and U are triangular, L^{-1} and U^{-1} are trivial to compute. Note that (19) is the same back substitution performed during Gaussian elimination, while (18) is also equivalent to back substitution. To obtain U the same reduction procedure is performed as in Gaussian elimination. Thus, in terms of the earlier third-order example,

$$U = \begin{bmatrix} y_{11}^{(1)} & y_{12}^{(1)} & y_{13}^{(1)} \\ 0 & y_{22}^{(2)} & y_{23}^{(2)} \\ 0 & 0 & y_{33}^{(3)} \end{bmatrix}. \tag{20}$$

Similarly, L is found to be a byproduct of the triangularization procedure. For the example,

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \dfrac{y_{21}^{(1)}}{y_{11}^{(1)}} & 1 & 0 \\ \dfrac{y_{31}^{(1)}}{y_{11}^{(1)}} & \dfrac{y_{32}^{(2)}}{y_{22}^{(2)}} & 1 \end{bmatrix}. \tag{21}$$

The elements of L are the coefficients in (12) and (14).

A convenient aspect of the preceding procedure is that since the diagonal elements of L are known to be 1's, L and U may share the same memory locations originally assigned to Y. Note further that the long operations count is the same as for Gaussian elimination. Once L and U have been computed, (18) and (19) can be applied repeatedly for different source vectors i. For m source vectors, the total long operations count becomes

$$\frac{n^3}{3} + mn^2 - \frac{n}{3}.$$

Pivoting

Further methods of reducing core requirements and operations counts are brought out below. First, however, mention should be made of error control. As can be seen from (12) and (14), a division and a subtraction are required at each step in the triangularization process. Suppose a computer which represents a number accurately to d digits, performs arithmetic operations correctly to $2d$ digits, and then rounds the result to d digits. It can be shown [44] that the absolute error resulting from such arithmetic operations is given by $1+\phi\ 10^{-d}$ where $0 \leq \phi \leq 5$. In the floating-point notation, the operations of subtraction and division yield results where magnitudes are less than those of the operands. The relative error in the result is larger; thus division and subtraction steps reduce accuracy. The lower accuracy is felt more severely on computers with smaller word lengths where d is reduced. One obvious solution is to declare all variables to be double precision and perform all operations in double precision. However, this significantly increases both time and core memory requirements.

One compromise which can be made is described by Ralston [45]. He shows that the critical steps involving divisions and accumulation of partial sums can be performed in double precision at the expense of only one additional double precision vector of dimension n. The only major added expense is computation time.

The most commonly used approach of reducing error (used by Ralston with the preceding method) is pivoting [45]. Partial pivoting amounts to scanning the elements of the ith column of the matrix below the diagonal at the ith step in the reduction and determining the element of largest magnitude. The row of this element is exchanged with the ith row and the reduction is continued. For the example considered previously, if at the second step $|y_{32}^{(2)}| > |y_{22}^{(2)}|$, $y_{32}^{(2)}$ would be chosen as the pivot element, $\mathscr{E}_3^{(2)}$ and $\mathscr{E}_2^{(2)}$ would be exchanged and $y_{22}^{(2)}$ would be eliminated from $\mathscr{E}_2^{(2)}$ to obtain $\mathscr{E}_2^{(3)}$. This technique tends to reduce the magnitude of the term being subtracted at each step and thus improve accuracy. Further, it preserves column order and hence the order in the solution vector v.

Complete pivoting involves finding the largest element in the as yet unreduced $(n-i) \times (n-i)$ submatrix at the ith step, exchanging rows and columns such that it appears on the diagonal in the ith row and column. Since column order is not preserved except at the expense of additional bookkeeping, complete pivoting is seldom used.

Sparse Matrix Techniques

In view of the n^3 dependence of the long operations count, computation time can be expected to increase significantly as larger circuits with more nodes are considered. Several recent papers [46]–[48] have focused attention on taking advantage of sparsity in the nodal admittance matrix. There are basically three associated economies. First, efficient means have been found by which only the nonzero entries of the matrix need be stored, thus effecting a savings in core memory. Second, it is possible to process only the nonzero entries at each step in the triangular reduction. Finally, the order in which variables are eliminated can be chosen to preserve sparsity. The long operations count and computation time are then reduced. This savings becomes even more significant when the same equations must be solved many times, as in multifrequency analysis. The optimal order for eliminating variables need only be determined once.

By way of illustration [86], in a new program developed at the University of California, Berkeley, by Prof. R. Rohrer and his students, the third technique of optimal ordering together with nonzero storage leads to a long operation count more closely proportional to n than to n^3. In the analysis of a typical operational amplifier of 22 nodes, the number of long operations was reduced from 2660, using Gaussian elimination, to 130.

Linear DC and AC Analysis Programs

Many linear analysis programs are available including ECAP, the ACCA portion of LISA, and ROHRERX.[1] As previously mentioned, these programs are based on a nodal analysis formulation. Both ACCA and ROHRERX have free

[1] ROHRERX is a program written and developed in 1969 by the IC Group, Electronics Research Laboratory, University of California, Berkeley.

format input languages as do certain time-sharing versions of ECAP. The primary advantages of ECAP are that it is well documented [50] and is available through many commercial time-sharing companies. Its disadvantages are that it is cumbersome to modify and the original input language is cumbersome to use.

ACCA, as part of the larger LISA package, shares a general input routine and has the advantage of being able to interact somewhat with other portions of LISA. Its disadvantage is that, if used as part of LISA, it is expensive to load. ROHRERX does make partial use of sparse matrix techniques and is therefore relatively fast. It presently suffers, however, from a lack of documentation.

At the present time efforts are being made to decrease computation time and to incorporate sensitivity and noise performance using the adjoint network [51]. This work is of major importance in the development of several computer-aided circuit design packages [52]–[54].

IV. Nonlinear DC Analysis

Most nonlinear dc analysis programs are incorporated into more general transient analysis programs. Equation formulation approaches are divided between nodal and state-variable analysis. In either approach, the nonlinear dc analysis problem reduces to one of solving simultaneous nonlinear algebraic equations. As described in the following paragraphs, iterative methods are used. Of particular importance is the convergence properties of the solution algorithm. Several approaches which are used in different programs to improve convergence are examined in this section. Finally, various programs are again compared.

Functional Iteration

The solution method used by virtually all of the presently available nonlinear dc analysis programs is based on the Newton–Raphson iteration technique. It is one of a broad class of techniques known collectively as functional iteration methods [44].

Given a set of nonlinear equations of the form of (2), for the dc case the equations may be expressed

$$\boldsymbol{g}(\boldsymbol{v}) = \boldsymbol{0}. \tag{22}$$

The solution technique is to start from some initial set of values $\boldsymbol{v}^{(0)}$ and to generate a sequence of iterates $\cdots \boldsymbol{v}^{(\nu-1)}$, $\boldsymbol{v}^{(\nu)}$, $\boldsymbol{v}^{(\nu+1)}, \cdots$ which converge to the solution $\hat{\boldsymbol{v}}$.

Newton–Raphson iteration is most easily introduced by considering the case of a single nonlinear equation

$$g(v) = 0. \tag{23}$$

The function $g(v)$ can be expanded about some point v_0 in a Taylor series to obtain

$$g(v) = g(v_0) + (v - v_0)g'(v_0) + \cdots = 0 \tag{24}$$

where the prime denotes differentiation with respect to v.

If only first-order terms are retained, a rearrangement of (24) yields

$$v = v_0 - \frac{g(v_0)}{g'(v_0)}. \tag{25}$$

The form of (25) suggests that a sequence of iterates might be generated by the following:

$$v^{(\nu+1)} = v^{(\nu)} - \frac{g(v^{(\nu)})}{g'(v^{(\nu)})}. \tag{26}$$

Equation (26) is the Newton–Raphson iteration function for the scalar case. Note that at the solution $\hat{v}$, $g(\hat{v})=0$ and $v^{(\nu+1)}=v^{(\nu)}$ as would be expected. The geometrical interpretation of (26) is illustrated in Fig. 1 for the simple case of a current source driving an ideal diode. The line tangent to the nonlinearity at the point $(v^{(\nu)}, g(v^{(\nu)}))$ has the slope $g'(v^{(\nu)})$. Its intercept with the voltage axis defines the next voltage iterate in the sequence as shown in the figure.

The generalization of the Newton–Raphson procedure to a system of n equations is given by

$$\boldsymbol{v}^{(\nu+1)} = \boldsymbol{v}^{(\nu)} - \boldsymbol{J}^{-1}(\boldsymbol{v}^{(\nu)})\boldsymbol{g}(\boldsymbol{v}^{(\nu)}) \tag{27}$$

where the Jacobian $\boldsymbol{J}(\boldsymbol{v})$ of the function $\boldsymbol{g}(\boldsymbol{v})$ is given by

$$\boldsymbol{J}(\boldsymbol{v}) = \begin{bmatrix} \frac{\partial g_1}{\partial v_1} & \frac{\partial g_1}{\partial v_2} & \cdots & \frac{\partial g_1}{\partial v_n} \\ \vdots & & & \vdots \\ \frac{\partial g_n}{\partial v_1} & \cdots & \cdots & \frac{\partial g_n}{\partial v_n} \end{bmatrix}. \tag{28}$$

A physical interpretation of the elements of $\boldsymbol{J}(\boldsymbol{v})$ is brought out below.

The direct application of (27) necessitates computing the inverse of the $n \times n$ Jacobian matrix. As indicated previously, the operation count for inverting a matrix and multiplying the result by a vector is $n^3 + n^2$.

An alternative procedure for obtaining new iterates is to solve the linear system of equations

$$\boldsymbol{J}(\boldsymbol{v}^{(\nu)})(\boldsymbol{v}^{(\nu)} - \boldsymbol{v}^{(\nu+1)}) = \boldsymbol{g}(\boldsymbol{v}^{(\nu)}). \tag{29}$$

A second alternative procedure often used with nodal analysis is to employ the system of equations

$$\boldsymbol{J}(\boldsymbol{v}^{(\nu)})\boldsymbol{v}^{(\nu+1)} = \boldsymbol{J}(\boldsymbol{v}^{(\nu)})\boldsymbol{v}^{(\nu)} - \boldsymbol{g}(\boldsymbol{v}^{(\nu)}). \tag{30}$$

The right-hand side of (30) is found to have a particularly simple interpretation as also brought out below.

Gaussian elimination applied to either (29) or (30) reduces the long operation count to $(n^3/3)+n^2-(n/3)$. An even greater advantage is obtainable using sparse matrix methods. The locations of the nonzero elements of the Jacobian matrix are fixed by the circuit topology and remain unchanged from iteration to iteration. The additional time required on the first iteration to record the nonzero structure and determine the optimal variable elimination order is small in comparison to the total computation time required as the number of iterations becomes large.

Newton–Raphson Applied To Nonlinear DC Analysis

A physical interpretation of the Jacobian matrix and the Newton–Raphson method can be made using the diode circuit of Fig. 2(a). The exponential nonlinearity of the diode is linearized about some trial solution voltage V_0.

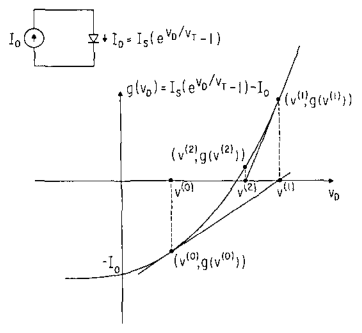

Fig. 1. Newton–Raphson iteration.

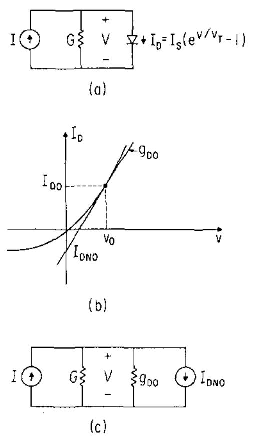

Fig. 2. Nonlinear dc analysis. (a) Diode circuit. (b) Linearized diode approximation. (c) Linearized circuit model.

This is equivalent to a Taylor series expansion as indicated previously where only first-order terms are retained. The expansion is of the form

$$I_D = I_D\Big|_{V=V_0} + (V - V_0)\frac{\partial I_D}{\partial v}\Big|_{V=V_0} \tag{31}$$

$$= I_s(e^{V_0/V_T} - 1) + (V - V_0)\frac{I_s}{V_T}e^{V_0/V_T} \tag{32}$$

$$= I_{D0} + (V - V_0)g_{D0} \tag{33}$$

where I_{D0} is recognized as the current through the diode corresponding to the voltage V_0, and g_{D0} is recognized as the dynamic conductance corresponding to the voltage V_0. Since the diode characteristic as described by (33) has now been linearized, the diode may be modeled in terms of a Norton equivalent current source I_{DN0} in parallel with the conductance g_{D0}. As can be seen from Fig. 2(b), I_{DN0} is given by

$$I_{DN0} = I_{D0} - g_{D0}V_0. \tag{34}$$

Hence, (32) may be written in terms of I_{DN0} as

$$I_D = g_{D0}V + I_{DN0}. \tag{35}$$

The nodal equation for the complete linearized circuit of Fig. 2(c) is

$$(G + g_{D0})V = I - I_{DN0}. \tag{36}$$

In terms of iterate values

$$(G + g_D^{(\nu)})V^{(\nu+1)} = I - I_{DN}^{(\nu)}. \tag{37}$$

If (30) is compared with (37) the physical interpretations of the Jacobian and the right-hand side of (30) become more apparent. The Jacobian consists of the nodal conductance matrix of the linear elements of the circuit together with the linearized conductances associated with each nonlinear circuit element. The vector on the right-hand side of (30) consists of independent source currents and the Norton equivalent source currents associated with each nonlinear circuit element. Thus at each iteration in a nodal analysis, the linearized conductances and Norton equivalent source currents must be recomputed and the linearized nodal conductance equations reassembled.

The application of Newton–Raphson iteration to the state-variable approach is also straightforward once all required coefficient matrices have been assembled. Recall that the equations formulated by the state-variable method can be put in the form of (7)–(9). The state vector $\boldsymbol{x}$ may represent capacitor voltages and inductor currents in which case the steady-state dc solution is characterized by $\dot{\boldsymbol{x}} = \boldsymbol{o}$. This corresponds to setting the current through capacitors and voltages across inductors to zero. Next, (7) may be solved for $\boldsymbol{x}$ in terms of $\boldsymbol{w}$ and the result substituted into (9) to obtain a system of nonlinear equations in terms of $\boldsymbol{w}$. Once $\boldsymbol{w}$ has been obtained $\boldsymbol{x}$ and $\boldsymbol{y}$ can also be obtained. Note that since the coefficient matrices involved are constant, the initial manipulation necessary to obtain the single system of nonlinear equations in $\boldsymbol{w}$ need only be performed once.

Convergence

For the nonlinear dc analysis approach just outlined, the problem of convergence to a solution is now considered. Proofs that an algorithm will converge depend upon a priori knowledge of an initial guess sufficiently close to the solution. Because this knowledge is usually not present in a nonlinear dc analysis, all techniques incorporated into analysis programs for improving convergence of the Newton–Raphson iteration technique are supported by purely empirical justification.

The exponential nonlinearities usually associated with diodes and bipolar transistors are single-valued monotonically increasing continuous functions. However, these expressions are strongly increasing functions. For large reverse bias, the slope approaches zero, while for large forward bias the exponential tends to infinity. Convergence may be slowed or the iteration procedure may be stopped when numbers exceed a computer-imposed limit. An approach commonly used to prevent such overflows is illustrated in Fig. 3 again for a simple diode characteristic. As previously indicated, at a trial junction voltage $v^{(\nu)}$, the characteristic is modeled in terms of a linearized approximation as shown. A new iterate value $v^{(\nu+1)}$ greater than $v^{(\nu)}$ must correspond to a solution on the linearized characteristic at the point 1′. Two choices for a new trial operating point are immediately available. The first is to update

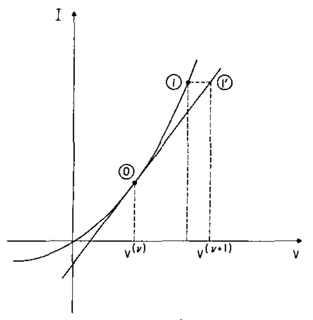

Fig. 3. Selection of new trial operating point.

with voltage by proceeding vertically to a new point on the exponential characteristic. This is the standard Newton–Raphson iteration and can be carried out by a straightforward evaluation of the exponential function. The second choice, which prevents the overflow problem, is to update with current by moving back horizontally to the exponential characteristic. This is done by computing the logarithm of the current corresponding to the point 1′ and results in the selection of point 1 as the new trial linearization point. Note that this procedure, which is particularly suited for junction diode and bipolar transistor circuits, can be used with both nodal analysis and state-variable analysis. This modified Newton–Raphson method can be used with other nonlinear devices but does require that they are described by functions which have an explicit inverse and for which both the function and its inverse are single-valued and continuous. For diodes and transistors the vanishing slope in the reverse direction is usually handled by placing a small leakage conductance across each junction.

Broyden [55] has recently proposed a variation of the Newton–Raphson technique. The method incorporates two modifications. The norm of the vector $\boldsymbol{g}(\boldsymbol{v})$, which must tend to zero as a solution is approached, is never allowed to increase. The method also avoids computation of the inverse Jacobian matrix at each iteration. Instead, an arbitrary approximation to the matrix is chosen at the first iteration and then successively updated. Branin and Wang [56] have applied the method to nonlinear dc problems, including statistical analysis. More recently, Broyden [57] has concluded that enforced norm reduction is not always advantageous and that an adaption of his method used in conjunction with a particular form of Davidenko's method [58] may converge more rapidly. In addition, Brown [59] has proposed a variation of the Newton–Raphson technique in which an inequality constraint is applied to the norm of the vector $-\boldsymbol{J}^{-1}(\boldsymbol{v})\boldsymbol{g}(\boldsymbol{v})$.

Three termination criteria are commonly used. The first is to stop iterating when the absolute difference between each unknown voltage or current iterate and its previous value is reduced below some preset minimum. The second is to stop when the relative error, defined as the absolute difference divided by the value of the iterate, is reduced to a preset minimum. Finally, an approach sometimes used with nodal analysis is to require that the sum of the currents at each node be reduced to a preset minimum. All of the approaches have advantages and disadvantages in specific cases and none is superior in general.

Nonlinear DC Analysis Programs

Both NET-1 and its predecessor TAP formulate the dc equations separately from the transient situation. Further, both programs consider the linear and nonlinear portions of a network separately for dc analysis. The linear portion is analyzed using a modification of Kron's method of tearing [60]. Nonlinearities are treated as a set of side constraints, and Newton–Raphson iteration is used.

In SCEPTRE different trees are chosen for dc and transient analyses. A modified Newton–Raphson procedure [61] is used which ensures that the true operating point on an exponential is approached from below. This amounts to updating with current whenever junction voltage increases.

Both CIRCUS, using the state-variable approach, and TRAC, using nodal analysis, employ Newton–Raphson iteration while updating with voltage in the third quadrant and updating with current in the first quadrant. BIAS-3, using nodal analysis and Newton–Raphson iteration, updates with current in the first quadrant when a junction voltage increases and with voltage when it decreases. In the third quadrant BIAS-3 always updates with voltage while modeling the exponential characteristic in terms of a line through the origin rather than a tangent. This does not affect the dc solution and yet insures that if a junction voltage becomes positive, the new trial linearization point will lie in the first quadrant.

TRAC, CIRCUS, and BIAS-3 converge reasonably well on most moderately sized circuits of up to 40 nodes. Nonlinear models built into TRAC include junction diodes and bipolar transistors. The program's input format is cumbersome while, as supplied by the Harry Diamond Laboratory, TRAC includes several assembly language subroutines which may require translation. CIRCUS has the advantage of a free format input language, zener-diode, tunnel-diode, unijunction, and junction field-effect transistor models in a stored library. It is, however, a much more complex program to implement and modify. BIAS-3 is relatively small but is limited to nonlinear dc analysis of bipolar transistor circuits.

SCEPTRE is by far the most flexible program in that models may be built by the user, nested, and recalled as necessary. The price of this flexibility is size and complexity as the program consists of over 15 000 statements. Its dc solution has also been known to suffer from poor convergence properties.

One recent program should also be mentioned. The DICAP portion of SYSCAP is an extremely general dc analysis program. It is large and is currently available only to users of Control Data Corporation's CYBERNET remote batch system.

V. Nonlinear Transient Analysis

The general procedure for the transient analysis of a nonlinear circuit is to evaluate the state of the circuit at a given point in time and to extrapolate ahead to a new time point.

The computation time required for such an analysis program is directly proportional to the number of time increments into which the analysis (simulation) time must be divided. The total simulation time is usually a multiple of the largest time constant (smallest eigenvalue) associated with the circuit linearized at a time point. On the other hand, for a large class of programs including NET-1, CIRCUS, and SCEPTRE, the length of a time step h in order to retain solution accuracy is determined by the smallest time constant (largest eigenvalue) within the circuit. Physically interpreted, this implies that if high-frequency devices are used in a low-frequency application, the computation time for an adequate simulation may be excessive. To understand how this problem is alleviated in programs such as TRAC and CIRPAC, it is necessary to examine the numerical integration process and the associated problem of stability.

Before proceeding, however, the use of the term numerical integration should be clarified. Strictly speaking, integration is associated with finding the area under some known function. The nonlinear transient analysis problem is one of obtaining the solution to a set of first-order nonlinear differential equations of the form (5). For the nonlinear situation, (5) can be written

$$\dot{x}(t) = f(x(t)). \tag{38}$$

The notation indicating the dependence of $f(x(t))$ on $u(t)$ has been dropped for convenience. Some initial condition

$$x(t)\big|_{t=0} = x(0) = x_0 \tag{39}$$

must be specified *a priori* or found as the result of a dc analysis. Because of the similarity of the formulas used in obtaining a solution to (38) to numerical integration formulas, the term has come to be loosely applied to both numerical processes.

Explicit Integration

Suppose at some point in time that $\dot{x}(t)$ is approximated by

$$\dot{x}(t) = \frac{x(t+h) - x(t)}{h}. \tag{40}$$

If (41) is substituted into (38), the result can be rearranged into the form

$$x(t+h) = hf(x(t)) + x(t). \tag{41}$$

Here $x(t+h)$ is defined explicitly in terms of $x(t)$. The numerical integration procedure defined by (41) is known as the forward Euler method [16] and is illustrated in Fig. 4(a). If $\hat{x}(t)$ is the exact solution to (38), it is easy to visualize the gross errors which can result from choosing h too large. The similarity of the right-hand side of (41) to a truncated Taylor series expansion about t suggests that the error in $x(t+h)$ will be proportional to h^2 multiplied by the second derivative of $x(t)$ evaluated someplace between t and $t+h$.

The forward Euler method defined by (41) is so low in accuracy that it is seldom used. Nonetheless, it serves to illustrate the idea that while the exact solution to (38) must be continuous, the computed solution is at best a piecewise-linear approximation to the exact solution.

In Fig. 4(a) the difference between the exact solution and the computed solution suggests the plausibility of using the

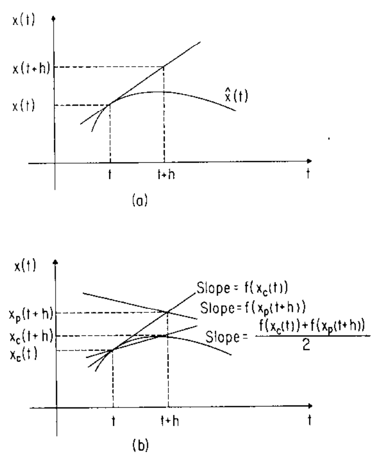

Fig. 4. Numerical integration. (a) Forward Euler method. (b) Improved predictor–corrector method.

average value of the derivatives at t and $t+h$ in (41). This results in what is called the trapezoidal integration method

$$x(t+h) = x(t) + \frac{h}{2}\left[f(x(t)) + f(x(t+h))\right]. \tag{42}$$

Equation (41) can be used to provide an initial approximation to $x(t+h)$ and thus $f(x(t+h))$. This is made clearer if (41) and (42) are rewritten as follows:

$$x_p(t+h) = hf(x_c(t)) + x_c(t) \tag{43}$$

$$x_c(t+h) = x_c(t) + \frac{h}{2}\left[f(x_c(t)) + f(x_p(t+h))\right]. \tag{44}$$

Equations (43) and (44) constitute a predictor–corrector pair, the predictor equation (43) providing $x_p(t+h)$ as an explicit function of $x_c(t)$ and the corrector equation (44) providing $x_c(t+h)$ as an explicit function of $x_c(t)$ and $x_p(t+h)$. This method is illustrated in Fig. 4(b). The increased accuracy which can result in relation to the forward Euler method is apparent as is the possibility of using more complex predictor–corrector pairs [62].

Implicit Integration

Consider now an alternative approach. The approximation to the derivative $\dot{x}(t)$ could just as easily have been made at $t+h$. In this case (41) becomes

$$x(t+h) = hf(x(t+h)) + x(t). \tag{45}$$

This is known as the backward Euler integration formula. Since $x(t)$ is known, (45) represents an implicit equation in the unknown $x(t+h)$. This equation may be solved for $x(t+h)$ by the Newton–Raphson method previously considered and the process repeated at each point in time. This procedure is called implicit numerical integration. The trapezoidal integration formula (42) is also an implicit integration formula.

All of the methods discussed thus far are of the general form

$$x(t+h) = \sum_{i=0}^{k} a_i x(t - ih) + h \sum_{j=-1}^{l} b_j \dot{x}(t - jh). \tag{46}$$

It is convenient to introduce an alternate notation. Without loss of generality it can be assumed that the time step h is constant and thus that $t=nh$. The notational change is defined such that

$$x(t + h) = x((n + 1)h) = x_{n+1}. \tag{47}$$

Equation (46) can now be written as (48)

$$x_{n+1} = \sum_{i=0}^{k} a_i x_{n-i} + h \sum_{j=-1}^{l} b_j \dot{x}_{n-j}. \tag{48}$$

Methods used in which $b_{-1}=0$ such as (41) or (43) and (44) are termed explicit integration methods while methods used in which $b_{-1}\neq 0$ such as (42) or (45) are termed implicit integration methods.

Runge–Kutta Integration

A third class[2] of integration methods differs from the preceding two classes in that the interval between t and $t+h$ is divided into subintervals. These routines, which are referred to as Runge–Kutta methods [45], seek to establish a very good approximation to an average derivative in the subinterval. A fourth-order Runge–Kutta procedure is given by

$$x(t + h) = x(t) + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4) \tag{49}$$

where

$$f_1 = f(x(t), t) \tag{50a}$$

$$f_2 = f\left(x(t) + \frac{h}{2}f_1, t + \frac{h}{2}\right) \tag{50b}$$

$$f_3 = f\left(x(t) + \frac{h}{2}f_2, t + \frac{h}{2}\right) \tag{50c}$$

$$f_4 = f\left(x(t) + hf_3, t + \frac{h}{2}\right). \tag{50d}$$

In addition, both higher and lower order Runge–Kutta methods are available.

Stability of Numerical Integration

With the preceding material as background, the problem of step-size determination with regard to stability can be considered [15]. In this context, stability refers not to the response of the physical circuit but rather to whether or not the errors generated at each step of the numerical integration process tend to decay (stable) or grow (unstable) as the solution progresses in time.

The study of stability is carried out by substituting the differential equation of interest into the particular form of the integration formula being considered and generating a difference equation. The roots of the difference equation are found by converting the difference equation into an algebraic equation. (In some cases the z transform can be used.) If the order of the difference equation exceeds the order of the original differential equation, here assumed to be first order, then all but one of the solutions to the difference equation are parasitic. Stability requires that all parasitic solutions have magnitudes less than unity. Those parasitic solutions whose magnitudes exceed unity represent growing solution modes which can be excited by local truncation errors and thus dominate the correct solution.

Through a transformation, the solutions to the difference equation can be related to the eigenvalues of the circuit at each time point and the time step h. For explicit integration and Runge–Kutta methods,[3] the time step h must inevitably be held smaller than a constant divided by the largest eigenvalue at the present time point in order to keep the parasitic solutions small. The constant involved is usually less than ten [43]. On the other hand, for implicit methods, it is found that this requirement may be relaxed by three or more orders of magnitude [63]. As brought out earlier, an iterative method such as Newton–Raphson must be used to determine the state of the circuit at $t+h$. The additional computation time required to solve the nonlinear system of equations at a time point is more than offset by the comparatively large steps in simulation time which may be taken between points. Note that in this regard both the backward Euler and the trapezoidal methods are found to be stable for all positive values of h when the eigenvalues lie in the open left half-plane. This does not mean that circuits such as oscillators and multivibrators whose eigenvalues may lie in the right half-plane cannot be analyzed but rather that the maximum value which h may be allowed to assume must be reduced.

Truncation Error

Stability of a numerical integration method implies only that parasitic solutions when excited will not grow with time. Thus stability only guarantees that as time is allowed to go infinity, the computed solution will converge to the exact solution. At finite times the solutions may differ significantly due to truncation error. The error term associated with the forward Euler method mentioned previously is an example. In that particular case, the error is associated with h^2 multiplied by the second derivative or curvature of $x(t)$. Thus in regions where the response is rapidly varying, h may have to be chosen many times smaller than stability considerations require. The time step h can be increased when the response is slowly changing.

In general, when a higher order integration formula is used, the truncation error may be made proportional to higher derivatives. Similar to a Taylor series, higher order terms tend to zero. The order of an integration formula can be determined in the following way. If linear differential equations whose solutions are polynomials of finite order are considered, the order of the polynomial of largest degree

[2] An alternative classification scheme is based on the number of previous time points at which values of $x(t)$ or $\dot{x}(t)$ are required to compute $x(t+h)$. Single-step methods require values only at times greater than or equal to t. Multistep methods may require values at $t-h$, $t-2h$, etc. In this sense, the Runge–Kutta method described here is a single-step method. It is also possible to use Runge–Kutta methods in a multistep manner by considering subintervals of width h in a total interval of width nh where $n>1$.

[3] Here only second or higher order methods are implied where order is defined in the subsection on truncation error. For first-order methods, no parasitic solutions exist.

for which (48) is exact is the order of the method. Without further justification, it is stated that backward Euler is first order while trapezoidal integration is second order.

A difficulty with higher order methods is that the number of parasitic solutions to the stability difference equation is increased with the order, and the restrictions on h become increasingly severe. A compromise must be made between using highly stable methods with larger truncation errors and less stable methods with reduced truncation errors. An approach used by several writers [64]–[66] is to vary the order of the method as the calculation proceeds, based on an examination of the eigenvalues or, equivalently, the rate at which the response is varying. The aim, of course, is to use as high an order method as possible consistent with stability [64].

Nonlinear Transient Analysis Programs

With the exception of the Runge–Kutta approach, the numerical integration formulas considered thus far and generalized in terms of (48) have all been linear in the sense that the value x_{n+1} is expressed as a linear combination of function values and derivatives. Pope [67] and Fowler and Warten [68] have developed modifications of predictor–corrector methods in which an exponential term is included. This approach allows a larger step size to be used in comparison with straight predictor–corrector or Runge–Kutta methods and has been used in both SCEPTRE and CIRCUS. The approach does not allow the dramatic increase in time steps which is possible with an implicit method. SCEPTRE includes two additional integration routines, a trapezoidal predictor–corrector method similar to (43) and (44) and a fourth order Runge–Kutta method similar to (49) and (50).

CIRPAC, though not generally available outside of Bell Telephone Laboratories, demonstrates the advantages of implicit integration methods. The program uses the second-order implicit integration formula [43]

$$x_{n+1} = -\tfrac{1}{3}x_{n-1} + \tfrac{4}{3}x_n + \tfrac{2}{3}h\dot{x}_{n+1} \tag{51}$$

which is given here without justification. The step size h, which is allowed to vary, is kept as large as possible consistent with maintaining a small local truncation error. Shichman [43] reports that CIRPAC typically runs up to ten times faster than CIRCUS and up to twenty times faster than an earlier version of CIRPAC which used a predictor–corrector routine.

The programs considered to this point use state-variable formulations where the presence of a first-order differential equation is made readily apparent. The TRAC program which employs nodal analysis uses a trapezoidal implicit integration method. The formulation can be conveniently illustrated for the case of a linear capacitor where the i–v characteristic is given by

$$i_c = C\frac{dv_c}{dt}. \tag{52}$$

In integral form, (52) can be rewritten as

$$\int_t^{t+h} i_c(t)\,dt = C\int_t^{t+h} d[v_i(t) - v_j(t)] \tag{53}$$

where $v_c(t)$ is taken to be the difference between the node voltages $v_i(t)$ and $v_j(t)$. The trapezoidal integration formula (42) applied to the left-hand side of (53) yields

$$\frac{h}{2}[i_c(t+h)+i_c(t)] = C[v_i(t+h)-v_i(t)-v_j(t+h)+v_j(t)]. \tag{54}$$

This equation can be rewritten to obtain

$$i_c(t+h) = \frac{2C}{h}[v_i(t+h) - v_j(t+h)] - \frac{2C}{h}[v_i(t) - v_j(t)] - i_c(t). \tag{55}$$

This last expression can be represented in terms of an equivalent conductance $2C/h$ in parallel with a current source between nodes i and j of value $-(2C/h)[v_i(t)-v_j(t)] - i_c(t)$. Both elements are readily handled in terms of nodal analysis.

Other reactive circuit elements are handled by TRAC in a similar manner. The modified Newton–Raphson method used for dc analysis and previously described is used at each time point. While nearly as fast as CIRPAC, TRAC does not incorporate the same sort of variable step-size feature and truncation error control. Rather the user has the option of specifying up to ten time intervals and the maximum step size to be allowed in each time interval. Thus, where it is known that a switching transient is about to occur, the user can force a smaller step size to be imposed. An advantage of TRAC is that it is readily available. Further, it is a relatively small program which consists of only 2000 statements.

Recently, a modified version of TRAC known as TRACAP and released as a part of the SYSCAP package has been announced. Like DICAP its dc counterpart, TRACAP is only available to users of Control Data Corporation's remote batch service CYBERNET. A version of TRAC known as MTRAC which handles magnetic cores has also been developed [27].

The transient portion of ECAP is restricted to circuits in which nonlinear elements are described by fixed piecewise-linear models. Ideal switches are used to move from one linear segment of a model to another in accordance with the direction of current through a sensing branch. The use of this portion of the program is quite cumbersome if piecewise-linear reactive elements are included. Implicit integration is used.

As to the future, NET-II[4] a completely revised version of NET-I (in FORTRAN) is under extensive test.

VI. Linear Pole-Zero Analysis

Linear pole-zero circuit analysis is considered as a separate topic because of the number of different techniques used and the specialized problems involved relative to frequency-domain analysis. Naturally, the poles and zeros of the transfer function, once obtained, can provide the same frequency response information as the linear ac analysis programs previously considered. However, in many problems it is the poles and zeros themselves which are of prime interest to the circuit designer.

[4] NET-II is a new version of NET-I by A. Malmberg, now being developed.

State-Variable and Eigenvalue Approach

The CORNAP program written by Pottle has found widespread use. As mentioned previously it employs a state-variable formulation leading to a set of equations of the form of (3) and (4). In the complex frequency domain these equations take the form

$$sx = Ax + Bu \tag{56a}$$

$$y = Cx + Du. \tag{56b}$$

The output vector y becomes

$$y = [C(sI - A)^{-1}B + D]u. \tag{57}$$

For the single-input–single-output case, the circuit transfer function $T(s)$ is given by

$$T(s) = \frac{y(s)}{u(s)} = C(sI - A)^{-1}B + D \tag{58}$$

where D is a scalar. The poles of the transfer function (natural frequencies of the circuit) are given by the zeros of det $(sI - A)$.

In an early version of CORNAP the recursive method of Leverrier [69], also known as the Souriau–Frame algorithm [70], was used to construct the adjoint of $(sI - A)$ and the characteristic polynomial associated with A. Muller's method [71] was then used to find the zeros of the polynomial. The recursive method however was particularly sensitive to roundoff errors which severely limited accuracy.

In the present version, use is made of the fact that the zeros of det $(sI - A)$, which are the zeros of the characteristic polynomial and the poles of $T(s)$, are also the eigenvalues of the matrix A. The QR algorithm of Francis [72]–[74] is used to compute these eigenvalues.

The transmission zeros of a circuit transfer function are obtained by making use of the fact that the zeros of the transmission function of a feedback network placed around an ideal amplifier of infinite gain are the poles of the closed-loop transfer function. The transfer function $T(s)$ is considered to be placed around such an ideal amplifier. State equations describing this inverse system [75] are obtained in terms of the original A, B, C, and D matrices. The eigenvalues of the inverse system are then the zeros of the original system.

CORNAP can handle up to 32 state variables (eigenvalues) but does require a 64×64 double precision matrix plus an additional 32×64 double precision matrix. The core requirements are thus large and it has been noted that accuracy is questionable for some large circuits of the order of 64 branches and 24 nodes.

A recent paper by Sandberg and So [76] describes an alternate approach for obtaining transmission zeros. The approach still makes use of the QR algorithm for finding eigenvalues.

Iterative Technique

The TRFN portion of LISA uses a very accurate iterative procedure to find the natural frequencies of a circuit. A nodal formulation is used and each term of the nodal admittance matrix represents a polynomial in s, the complex frequency variable. For single input single output, the transfer function is

$$T_{mn}(s) = \frac{v_n(s)}{i_m(s)} = \frac{\det Y_{mn}(s)}{\det Y(s)}. \tag{59}$$

$Y_{mn}(s)$ is the minor of the element $y_{mn}(s)$, the poles of $T_{mn}(s)$ are the zeros of det $Y(s)$. The zeros of $T_{mn}(s)$ are the zeros of det $Y_{mn}(s)$. The iterative root-finding method of Muller is applied directly to det $Y(s)$ and det $Y_{mn}(s)$. In Muller's method, the determinant is initially evaluated at three points and modeled by a quadratic. A zero of this quadratic is then used in place of one of the original evaluation points. This procedure is continued and a sequence of better and better approximations to each zero of the determinant is developed. The amount of determinantal evaluation time appears excessive; however, the accuracy is excellent. Further, it is possible to program this approach very effectively to reduce greatly the computation time. FRANK,[5] a program of this type, is faster for large circuits than CORNAP, more accurate requiring only single precision, and uses significantly less core. Finally, where element values are to be varied and analyses repeated, a large savings in time may result because previous solutions can be used to supply good initial estimates of new solutions.

Laplace Expansion Approach

By a straightforward Laplace expansion of the determinant of the nodal admittance matrix $Y(s)$, the coefficients of the characteristic polynomial can be obtained. Muller's method can then be used to solve for the zeros of the polynomial. An approach of this type is used in the POLY portion of LISA; however, it is severely limited by the effects of roundoff error and is limited to circuits of less than 12 nodes.

Larger circuits can be handled effectively with programs of this type if a restriction is made to active RC circuits. Several polynomial manipulations are then eliminated and accuracy and speed are improved. Program SPRAGUE[6] has been found to be accurate and fast for circuits up to 19 nodes.

Nodal Analysis–Eigenvalue Approach

The eigenvalue approach based on a nodal equation formulation can be used for circuits restricted to active RC elements. The nodal admittance matrix has the form

$$Y(s) = G + sC. \tag{60}$$

Eigenvalue programs of this type have been developed both at M.I.T. [30], [31] and at the Technical University of Denmark [77]. The M.I.T. program requires the user to enter the elements of the matrices G and C of (60) directly; however, a topological input routine can easily be added.

[5] FRANK is a program written and developed in 1969 by the IC Group, Electronics Research Laboratory, University of California, Berkeley.

[6] SPRAGUE is a program written and developed in 1968 by the IC Group, Electronics Research Laboratory, University of California, Berkeley.

The method is based on the transformation of det $(\boldsymbol{G}+s\boldsymbol{C})$ to det $(\hat{\boldsymbol{G}}\hat{\boldsymbol{C}}^{-1}-s\boldsymbol{I})$ where $\boldsymbol{I}$ is the identity matrix and $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{C}}^{-1}$ are matrices of rank less than or equal to n the order of $\boldsymbol{Y}(s)$. The transformation is carried out such that the zeros of the original determinant are unchanged. The zeros of the alternate determinant, however, are also the eigenvalues of $\hat{\boldsymbol{G}}\hat{\boldsymbol{C}}^{-1}$. The QR method is used to compute these eigenvalues. The transmission zeros are obtained by applying the same procedure to $\boldsymbol{Y}_{mn}(s)$.

If the rank of $\hat{\boldsymbol{C}}$ is equal to n, $\hat{\boldsymbol{G}}\hat{\boldsymbol{C}}^{-1}$ can be obtained by applying a variation of Gaussian elimination known as Gauss–Jordan reduction to $\boldsymbol{C}$ and performing the same operation on $\boldsymbol{G}$. Gauss–Jordan reduction differs from Gaussian elimination in that a matrix is diagonalized and then normalized to obtain the identity matrix. This requires elimination of elements above the diagonal as well as those below. If the rank of $\hat{\boldsymbol{C}}$ is less than n, then redundant rows and columns exist in $\hat{\boldsymbol{G}}$ and $\hat{\boldsymbol{C}}$ and must be eliminated [77].

This method again requires only sufficient core to store the nodal admittance matrix and is very fast. However, it is a relatively new method and its relative accuracy has yet to be determined, though it should be better than that of the Laplace expansion approach.

An extension to active *RLC* networks has also been described [78]. Inductive elements are replaced by ideal gyrator–capacitor equivalent circuits. The effectiveness of this extension to the preceding method has not yet been evaluated, however.

Summary

A number of techniques which are used for computing the poles and zeros of circuit transfer functions have been considered. Though each approach requires more extensive computations than are required to compute responses in the frequency domain, the additional insight obtained from a knowledge of the poles and zeros often merits the additional effort. In all cases, the order of the circuits which can be analyzed is significantly less than that which can be handled with the frequency-domain program described earlier.

No mention has been made of programs using a topological tree enumeration formulation. While accurate [79]–[80] and useful for sensitivity studies because of the explicit form of the coefficients of the polynomials, these programs have been found to be too slow to be of interest and are limited to very small circuits. Similar conclusions have been found to apply to programs based on flowgraph methods [81].

VII. Conclusion

In the preceding sections of this paper, the basic elements of computer-aided circuit analysis have been reviewed with emphasis on those techniques and routines necessary for the adequate simulation of four basic classes of circuits: linear dc and ac, nonlinear dc, nonlinear transient, and linear pole zero. One topic not considered but very important is the development of device models suitable for computer-aided circuit analysis. An overview of modeling would, in itself, require a full paper and thus cannot be undertaken here. However, several articles on the modeling of bipolar and field-effect transistor models are included in [82]–[85].

References

[1] F. F. Kuo and J. F. Kaiser, Eds., *Systems Analysis by Digital Computer*. New York: Wiley, 1966.

[2] G. J. Herskowitz, Ed., *Computer-Aided Integrated Circuit Design*. New York: McGraw-Hill, 1968.

[3] F. F. Kuo and W. G. Magnuson, Jr., Ed., *Computer Oriented Circuit Design*. Englewood Cliffs, N. J.: Prentice-Hall, 1969.

[4] H. Falk, "Computer programs for circuit design," *Electro-Technol.* (New York), vol. 77, pp. 54–57, June 1966.

[5] D. Christiansen, "Computer-aided design—I: the man-machine merger," *Electronics*, vol. 39, pp. 110–123, September 19, 1966.

[6] J. Dunanian, "Check design program availability," *Electron. Des.*, vol. 14, pp. 76–80, October 11, 1966.

[7] R. H. Dickhaut, "Comparison of three computer-aided design programs," *Electro-Technol.*, vol. 78, pp. 88–89, January 1967.

[8] ——, "Computer-aided design—VI: comparing the 'big two' programs," *Electronics*, vol. 40, pp. 74–90, February 1967.

[9] W. G. Magnuson, Jr., "Computer-aided design—VIII: picking transient analysis programs," *Electronics*, vol. 40, pp. 84–87, April 17, 1967.

[10] G. K. Pritchard, "A survey of transient circuit analysis programs," *1967 NASA Computer-Aided Circuit Design Seminar Proc.*, pp. 97–104.

[11] D. F. Dawson, F. F. Kuo, and W. G. Magnuson, Jr., "Computer-aided design of electronic circuits—a user's viewpoint," *Proc. IEEE*, vol. 55, pp. 1946–1954, November 1967.

[12] F. H. Branin, Jr., "Machine analysis of networks and its applications," IBM Development Lab., Poughkeepsie, N. Y., Tech. Rep. TR 00.855, 1962.

[13] ——, "Computer-aided design—IV: analyzing circuits by the numbers," *Electronics*, vol. 40, pp. 88–103, January 9, 1967.

[14] *Proc. IEEE* (Special Issue on Computer-Aided Design), vol. 55, November 1967.

[15] D. A. Calahan, *Computer-Aided Network Design*, preliminary ed. New York: McGraw-Hill, 1968.

[16] L. P. Huelsman, *Digital Computations in Basic Circuit Theory*. New York: McGraw-Hill, 1968.

[17] Cornell Electrical Engineering Conf., Biennial Conf. Proc. on Computerized Electronics, August 1969.

[18] T. R. Bashkow, "The A matrix, new network description," *Trans. IRE Circuit Theory*, vol. CT-4, pp. 117–119, September 1957.

[19] N. G. Brooks and H. S. Long, "A program for computing the transient response of transistor switching circuits—PETAP," IBM Development Lab., Poughkeepsie, N. Y., Tech. Rep. TR 00.700, 1959.

[20] F. H. Branin, Jr., "D-C analysis portion of PETAP—a program for analyzing transistor switching circuits," IBM Development Lab., Poughkeepsie, N. Y., Tech. Rep. TR 00.701, 1960.

[21] "1620 electronic circuit analysis program [ECAP][1620-EE-02X] user's manual," IBM Application Program File H20-0170-1, 1965.

[22] A. F. Malmberg, F. L. Cornwell, and F. N. Hofer, "NET-1 network analysis program," Los Alamos Scientific Lab., Los Alamos, N. Mex., Rep. LA-3119, 7090/94 version, August 1964.

[23] "Automated digital computer program for determining responses of electronic systems to transient nuclear radiation (PREDICT)," IBM Space Guidance Center, Oswego, N. Y., IBM File 64-521-5, July 1964.

[24] L. D. Milliman, W. A. Massena, and R. H. Dickhaut, "CIRCUS—a digital computer program for transient analysis of electronic circuits —user's guide," Boeing Co., Seattle, Wash., Harry Diamond Lab. Rep. AD-346-1, January 1967.

[25] H. W. Mathers, S. R. Sedore, and J. R. Seuts, "Automated digital computer program for determining responses of electronic circuits to transient nuclear radiation (SCEPTRE)," IBM Space Guidance Center, Oswego, N. Y., IBM File 66-928-611, February 1967.

[26] E. D. Johnson, C. T. Kleiner, L. R. McMurray, E. L. Steele, and F. A. Vassallo, "Transient radiation analysis by computer program (TRAC)," Autonetics Div., North American Rockwell Corp., Anaheim, Calif., Tech. Rep. issued by Harry Diamond Labs., June 1968.

[27] D. Nitzan and J. R. Herndon, "MTRAC—a computer program for analysis of circuits including magnetic cores," Stanford Research Institute, Menlo Park, Calif., SRI Project 6408, Rep. 6, June 1969.

[28] "NAR/SYSCAP, system of circuit analysis programs," Autonetics Div., North American Rockwell Corp., 1969.

[29] W. J. McCalla and W. G. Howard, "BIAS-3, a program for the nonlinear dc analysis of bipolar transistor circuits," *1970 Int. Solid State Circuits Conf. Dig.*
[30] P. E. Gray and C. L. Searle, "NATFREQS (FORTRAN)," in *Electronic Principles, Physics, Models, and Circuits.* New York: Wiley, 1969, Appendix C.
[31] W. H. Ohm, "Applications of computer-aided circuit design and analysis," S.B. thesis, Massachusetts Institute of Technology, Cambridge, June 1968.
[32] K. L. Deckert and E. T. Johnson, "User's guide for LISA," IBM, San Jose, Calif., 7094-IBM0001, August 1967.
[33] ——, "LISA—a program for linear systems analysis," IBM, San Jose, Calif., presented at WESCON, Los Angeles, Calif., August 1966.
[34] C. A. Desoer and E. S. Kuh, *Basic Circuit Theory.* New York: McGraw-Hill, 1969, ch. 12.
[35] E. S. Kuh and R. A. Rohrer, "The state-variable approach to network analysis," *Proc. IEEE*, vol. 53, pp. 672–686, July 1965.
[36] P. R. Bryant, "The order of complexity of electrical networks," *Proc. Inst. Elec. Eng.*, monograph 335E, vol. 106C, pp. 174–188, June 1959.
[37] ——, "The explicit form of Bashkow's *A* matrix," *Trans. IRE Circuit Theory* (Correspondence), vol. CT-9, pp. 303–306, September 1962.
[38] R. L. Wilson and W. A. Massena, "An extension of Bryant–Bashkow *A* matrix," *IEEE Trans. Circuit Theory* (Correspondence), vol. CT-12, pp. 120–122, Mareh 1965.
[39] C. Pottle, "State-space techniques for general active network analysis," in *Systems Analysis by Digital Computer*, F. F. Kuo and J. F. Kaiser, Eds. New York: Wiley, 1966, ch. 3.
[40] ——, "A 'textbook' computerized state-space network analysis algorithm," *IEEE Trans. Circuit Theory* (Correspondence), vol. CT-16, pp. 566–568, November 1969.
[41] C. A. George, "BELAC user's manual," General Electric Co., Utica, N. Y., Tech. Info. Ser. R69EML11, August 1969.
[42] H. Shichman, "Computation of dc solutions for bipolar transistor networks," *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 460–466, November 1969.
[43] ——, "The integration system of a nonlinear network-analysis program," *IEEE Trans. Circuit Theory*, vol. CT-17, pp. 378–386, August 1970.
[44] E. Isaacson and H. B. Keller, *Analysis of Numerical Methods.* New York: Wiley, 1966.
[45] A. Ralston, *A First Course in Numerical Analysis.* New York: McGraw-Hill, 1965.
[46] N. Sato and W. F. Tinney, "Techniques for exploiting the sparsity of the network admittance matrix," *IEEE Trans.* (Power App. Syst.), vol. 82, pp. 944–950, December 1963.
[47] 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, November 1967.
[48] *Sparse Matrix Symp. Proc.*, IBM Rep. RA-1, March 1969.
[49] R. D. Berry, "An optimal ordering of electronic circuit equations for a sparse matrix solution," this issue, pp. 40–50.
[50] R. W. Jensen and M. D. Lieberman, *IBM Electronics Circuit Analysis Program: Techniques and Applications.* Englewood Cliffs, N. J.: Prentice-Hall, 1968.
[51] S. W. Director and R. A. Rohrer, "The generalized adjoint network and network sensitivities," *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 318–323, August 1969.
[52] ——, "Automated network design—the frequency-domain case," *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 330–337, August 1969.
[53] A. J. Broderson, S. W. Director, and W. A. Bristol, "Simultaneous automated ac and dc design of linear integrated circuit amplifiers," this issue, pp. 50–58.
[54] B. A. Wooley, "The computer-aided design optimization of integrated broadband amplifiers," *1970 ISSCC Dig. Tech. Papers*, pp. 74–75.
[55] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Math. Comp.*, vol. 19, pp. 577–593, October 1965.
[56] F. H. Branin, Jr., and H. H. Wang, "A fast reliable iteration method for dc analysis of nonlinear networks," *Proc. IEEE*, vol. 55, pp. 1819–1826, November 1967.
[57] C. G. Broyden, "A new method of solving nonlinear simultaneous equations," *Comput. J.*, vol. 12, pp. 94–99, February 1969.
[58] D. F. Davidenko, "On a new method of numerical solution of systems of nonlinear equations," *Dokl. Akad. Nauk SSSR*, vol. 88, pp. 601–602, 1953.
[59] G. C. Brown, "DC analysis of nonlinear networks," *Electron. Lett.*, vol. 5, pp. 374–375, August 1969.
[60] G. Kron, "A method of solving very large physical systems in easy stages," *Proc. IRE*, vol. 42, pp. 680–686, April 1954.
[61] R. H. Dickhaut, "Advances in computer techniques for radiation effects calculations," Boeing Doc. D2-90571, 1964.
[62] R. W. Hamming, *Numerical Methods for Scientists and Engineers.* New York: McGraw-Hill, 1962.
[63] I. W. Sandberg and H. Shichman, "Numerical integration on systems of stiff nonlinear differential equations," *Bell Syst. Tech. J.*, vol. 47, pp. 511–527, April 1968.
[64] C. W. Gear, "Numerical integration of stiff ordinary differential equations," Department of Computer Science, University of Illinois, Urbana, Rep. 221, 1967.
[65] W. Liniger and R. A. Willoughby, "Efficient numerical integration of stiff systems of ordinary differential equations," IBM Watson Research Center, Yorktown Heights, N. Y., Res. Rep. RC-1970, 1968; *SIAM J. Numer. Anal.*, vol. 7, pp. 47–66, March 1970.
[66] P. M. Russo, "On the time domain analysis of linear time-invariant networks with large time-constant spreads by digital computer," this issue, pp. 194–197.
[67] D. A. Pope, "An exponential method of numerical integration of ordinary differential equations," *Comm. ACM*, vol. 6, pp. 491–493, August 1963.
[68] M. E. Fowler and R. M. Warten, "A numerical technique for ordinary differential equations with widely separated eigenvalues," *IBM J. Res. Develop.*, pp. 537–543, September 1967.
[69] A. S. Householder, *The Theory of Matrices in Numerical Analysis.* Waltham, Mass.: Blaisdell, 1964, pp. 166–168.
[70] L. A. Zadeh and C. A. Desoer, *Linear System Theory.* New York: McGraw-Hill, 1963, pp. 303–305.
[71] D. E. Muller, "A method for solving algebraic equations using an automated computer," in *Mathematical Tables and Other Aids to Computation*, vol. 10, pp. 208–215, 1956.
[72] J. G. F. Francis, "The Q-R transformation—I," *Comput. J.*, vol. 4, pp. 265–271, October, 1961; "The Q-R transformation—II," *Comput. J.*, vol. 4, pp. 332–345, January 1962.
[73] J. H. Wilkinson, *The Algebraic Eigenvalue Problem.* New York: Oxford, 1965, ch. 8.
[74] B. N. Parlett, "The *LU* and *QR* algorithms," in *Mathematical Methods for Digital Computers*, vol. 2, A. Ralston and H. S. Wilf, Eds. New York: Wiley, 1967, ch. 5, pp. 116–130.
[75] R. W. Brockett, "Poles, zeros, and feedback: state space interpretation," *IEEE Trans. Automatic Control*, vol. AC-10, pp. 129–135, April 1965.
[76] I. W. Sandberg and H. C. So, "A two-sets-of-eigenvalues approach to the computer analysis of linear systems," *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 509–517, November 1969.
[77] E. V. Sorensen, "Circuit analysis by algebraic eigenvalue technique," to be published.
[78] R. E. Parkin, "A state variable method of circuit analysis based on a nodal approach," *Bell Syst. Tech. J.*, pp. 1957–1970, November 1968.
[79] D. A. Calahan, "Linear network analysis and realization digital computer programs, and instruction manual," *University of Illinois Bulletin*, vol. 62, February 1965.
[80] E. V. Sorensen, "A preliminary note on the analytical network program ANP1," Laboratory of Circuit Theory, Technical University of Denmark, Lyngby, Rep. LKT23, 1967.
[81] L. P. McNamee and N. Potash, "A user's and programmer's manual for NASAP," University of California, Los Angeles, Calif., Rep. 68-38, August 1968.
[82] D. A. Hodges and H. Shichman, "Modeling and simulation of insulated-gate field-effect transistor switching circuits," *IEEE J. Solid-State Circuits*, vol. SC-3, pp. 285–289, September 1968.
[83] D. Frohman-Bentchkowsky and L. Vadasz, "Computer-aided design and characterization of digital MOS integrated circuits," *IEEE J. Solid-State Circuits*, vol. SC-4, pp. 57–64, April 1969.
[84] H. K. Gummel and H. C. Poon, "An integral charge control model of bipolar transistors" (to be published).
[85] F. A. Lindholm, "Integrated-circuit transistor and diode models for network-analysis programs," in this issue, pp. 122–128.
[86] R. D. Berry, private communication.