PERIODIC RESPONSE TECHNIQUE FOR LARGE NONLINEAR CIRCUITS

Farag S. Fattouh, B.Sc.(Hons. El.), B.Sc.(Hons. Phys.) (Cairo U.)

COMPUTATIONAL TECHNIQUE FOR THE PERIODIC RESPONSE OF LARGE NONLINEAR CIRCUITS

by

Farag S. Fattouh, B.Sc. (Hons. Elect.), B.Sc. (Hons. Phys.), (Cairo U.)

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering.

Department of Electrical Engineering
McGill University,
Montreal, Canada.

December, 1977.

To:

Badria,

Sáadia,

Nadia,

and

Sanáa.

ABSTRACT

The gradient algorithm for computing the steady-state, periodic response of circuits has been derived on the basis of a generalized tableau representation of the network equations. In contrast to the state variable formulation which was recently reported, the present one lends itself to straightforward implementation in modern transient analysis programs for large scale, nonlinear circuits, which make use of sparse matrix techniques. The algorithm has been implemented in one such program, SCAMPER, and details of the program as well as results of tests with several circuits are presented. The rate of convergence depends on the optimization subroutine and is sensitive to scaling of both the variable as well as the gradient vectors.

RESUME

L'algorithme du gradient pour le calcul de la réponse périodique d'un circuit a été derivé à partir d'une representation généralisée en tableau des équations du circuit. En contraste avec la formulation selon les variables d'état qui a été récemment publiée, ce rapport presente une méthode plus aisement applicable avec les programmes moderne pour l'analyse transitoire des réseaux non-linéaire et de grande dimension, utilisant des techniques de matrices creuses. On a implanté l'algorithme sur le programme SCAMPER, les détails du programme aussi bien que les résultats des tests sur plusieurs circuits sont presentés. La convergence des résultats dépend des sous-programmes d'optimisation et est sensible aux transformations linéaires des variables aussi bien que du vecteur gradient.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Professor N. Rumin for his continual encouragement and helpful criticism during the course of this work. Professor M.L. Blostein was the source of many valuable suggestions as well as of considerable assistance during the period of familiarization with the intricacies of SCAMPER, for which the author is deeply indebted.

Thanks are also extended to Mr. D.J. Millar who also helped to reduce the author's period of familiarization with SCAMPER, and to Dr. M.A. Nakhla and Professors C. Paige and A. Buckley who very kindly provided the optimization subroutines which were used in this work as well as many useful suggestions.

Special thanks are due to Mrs. P. Hyland for her skillful typing of the manuscript.

The work was supported in part by the National Research Council of Canada.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Many important classes of electronic circuits, such as power supplies, large signal amplifiers, oscillators, et cetera, operate in the periodic steady-state mode and yet, because of their nonlinearity, the designer is usually forced to analyze them using time domain, transient analysis computer programs. The periodic steady state solution can be found by the continuous integration ("brute force" method) of the system equations until all the transient components die away. However, such circuits are, more often than not, lightly damped, in which case this method becomes prohibitively expensive because the integration process has to be continued for a very large number of periods. Recently, this problem has received considerable attention which has yielded several procedures for computing the steady-state response of nonlinear circuits at a cost which is considerably less than that for the "brute force" method.

In 1972, Aprille and Trick [1] proposed a method for the steady-state analysis of nonlinear circuits with periodic input which is based on the Newton algorithm and which they implemented using a state variable formulation of the network equations. The Newton step is used iteratively to adjust the initial conditions at some time t_0 until the difference between these and the conditions at time $t = t_0 + T$, where T is the period, is negligible. An extension of this method to the oscillatory (autonomous) case was given in [2], where the period T is

considered as an unknown together with the initial conditions. The restriction of using the state variable representation of the network equations is inconvenient because it prevents the implementation of the method in transient analysis programs which are based on the tableau representation and which, consequently, can handle large problems. Generalization of the Newton method without this restriction were reported in [3], [4], and [5]. Director [5] used the concept of the "adjoint networks" which he had introduced in [6], while Trick et al [3], [4] based their approach on the "sensitivity circuits" concept. In fact, both the adjoint networks and the sensitivity circuits concepts can be applied to either the state variable or the tableau form, which ever is more suitable. If the network has N reactive elements, both techniques - the sensitivity circuits or the adjoint networks - involve the integration of N adjoint systems of equations in addition to the original network equations over one full period. For networks of moderate order N, the Newton method works quite well. But for high order systems (say $N > 50$), the computational cost may be prohibitive, and numerical difficulties may be encountered due to necessity of solving systems of large and full matrices.

A second direction for solving the periodic steady-state problem was initiated by Nakhla and Branin [10], [11], using an optimization-like approach, called the gradient method. A performance function ϕ, is defined which reflects the state of the system relative to its steady-state. The first derivatives (gradient) of the

objective function with respect to the initial conditions of the state variables x_0, in addition to both x_0 and ϕ are passed to an optimization routine. The new values of x_0 returned by the optimization routine are used to update the initial conditions of the system. This process is repeated until the objective function ϕ reaches a specified minimum which is considered to correspond to the steady-state. The effort required in the gradient method for each iteration is the integration of only two systems of equations over one period, the original system in the forward direction, and the corresponding adjoint variational system in the backward direction. The gradient method is characterized by its low cost of integration per iteration step as well as by its freedom from potential numerical stability problems. Nakhla and Branin [10] implemented the gradient method using the state variable formulation, although they did also discuss a more general approach to their method. However, it was not particularly suitable for application with analysis programs based on the tableau representation of the network equations. It should be mentioned here that a similar gradient method was used by Director [7 - 9] in optimizing nonlinear circuits. In [8], [9] Current and Director showed that the optimization problem can be extended to include that of determining the periodic steady-state as well. In other words, the variable space can be extended to include not only the optimization parameters but also the initial conditions of the network.

A third direction for determining the steady-state of periodically forced nonlinear systems was given by Skelboe [16] and called the extrapolation method. This approach is based on the ε - algorithm by Wynn [15] and is, in fact, a sequence-to-sequence transformation where the ε - algorithm step is used to modify the initial conditions in a nonlinear sequence of operations. The algorithm does not require the computation of first derivatives (such as the sensitivity matrix for the Newton method, or the gradient vector for the gradient method), but, in place, it requires the integration of the system over many periods. The advantage of the extrapolation method is its quadratic rate of convergence (or even higher). But there are many restrictions on the method such as the assumption that the system is mildly nonlinear and the requirement that the order of the system be not too large [16].

As mentioned above, the gradient method introduced in [10], [11] was based on the state variable approach. In the work reported here, a more general formulation is developed for use in transient analysis programs based on tableau representation of network equations. The detailed implementation of the resultant algorithm in one such program SCAMPER is given as well as the numerical results too of some test problems solved on the new program.

In Chapter II, a survey of the above three methods for the periodic steady-state analysis (Newton method, gradient method, and extrapolation method) are presented. For the Newton method, the implementa-

tion is discussed according to two different approaches, the sensitivity circuits [4], and the adjoint networks [5]. Also, for both the Newton and the extrapolation methods, executable algorithms suited to the general representation of network equations are presented. In the case of the gradient method the approach is the same as that given by Nakhla [11]. The chapter ends with a discussion of the advantages and disadvantages of these methods.

The gradient method based on the general tableau representation is developed in Chapter III. Lagrangian multipliers are used in a manner similar to that employed by Director [8] to define the adjoint system of equations. A fairly general representation of the network nonlinearities is assumed. The method is applied to the dc and transient analysis program SCAMPER on the basis of a particular choice of the objective function ϕ

In Chapter IV, some considerations related to the problem of scaling are presented, and the numerical results of some test problems are given. It should be noted that the same examples were studied by many authors including Trick [1], [3], Director [5], Skelboe [16], and Nakhla [11]. Since the algorithm presented in Chapter III is essentially a generalization of Nakhla's method and because the same optimization routine [22] was used, the results of both studies are compared where possible and are shown to be in good agreement.

Chapter V is concerned with the programming problem. In Part A, a review of the existing dc and transient analysis program SCAMPER [19], [20] is given. The key features and procedures are discussed, in particular, code generation, sparse matrix and pivoting techniques, and error and step size control. In Part B, the main features of the additional programming required for implementing the steady-state algorithm presented in Chapter III, are discussed. These features are : (1) relation between the original system and its adjoint, (2) the procedures for generating the necessary code segments for integrating the adjoint system, (3) the problem of data manipulation (retrieval from disk, interpolation), (4) setting the initial conditions of both the original and adjoint systems, and (5) error and step size control.

The main results of the present work are summarized in Chapter VI, which concludes with a brief discussion of further improvements to the present method as well as of topics for work in this area.

CHAPTER II

PERIODIC STEADY-STATE ANALYSIS ALGORITHMS

The periodic steady state of an electronic network can be analyzed either in the time domain or in the phase (frequency) domain. If the network is almost linear, with relatively few nonlinear elements, and the harmonic content is not too large, the phase domain is preferable [25]. But, when the network contains a significant number of nonlinear elements with sharp nonlinearities, the time domain is more suitable. Since it is this latter class of circuits which we are interested in, the following discussion will concentrate on time domain algorithms.

The objective here will be to examine several algorithms for the computation of the periodic steady state, from the point of view of their suitability for inclusion in an existing dc and transient analysis program based on the sparse tableau approach. Furthermore, although we are primarily interested in the nonautonomous case, ease of extension to the autonomous network is very desirable. Finally, an approach which would permit the subsequent inclusion of circuit optimization evidently merits particularly serious consideration.

The most recent algorithms are of the iterative type, and can be divided into two groups:

(i) The self sustained algorithms, i.e., those which do not need external steps, such as optimization. The

Newton Methods [1 - 5] and the extrapolation methods [16], are members of this group.

(ii) The optimization - like algorithms, i.e., those that involve an external optimization step to achieve the circuit analysis. The best known method in this group is the gradient method [10], [11].

In order to simplify the discussion, the reduced (state variable) form of equations will be used, but more general representations will be introduced where necessary.

Assume that we have a system of nonlinear differential equations given by,

$$\dot{q} = f(q, t) \qquad (2.1)$$

where q is the differentiated variable vector, and f is a R^n vector and has a continuous, first order partial derivative w.r. to q for $t < 0 , \infty >$. Furthermore, we assume that the system has a unique periodic solution trajectory of period T, i.e.:

$$q(t) = q(t + n T) \quad ; \quad n = 1, 2, 3, \ldots$$

We seek a set of initial conditions $q(t_0)$ at $t = t_0$, which will put the system into its periodic steady-state, i.e.,

$$q(t_0) = q(t_0 + T) \qquad (2.2)$$

Following is a discussion of three steady-state analysis methods, with particular emphasis on the algorithms and computational techniques.

2.1 Newton Method

The application of the Newton method for the periodic steady-state analysis was first introduced by Aprille and Trick [1]. Several improvements of the computation techniques as well as extensions of the area of application are given in [2 - 5]. In this section, the Newton iteration will be stated, and two different computational techniques will be given, one using an approach based on sensitivity circuits [4], and the other on adjoint networks [5].

For iteration number i, assume q_0^i is the initial condition vector. It is required to adjust q_0, until the difference between two succeeding iterations initial conditions, $|\Delta q_0^i| = |q_0^i - q_0^{i+1}|$, is less than an acceptable lower limit, at which the system will be considered in steady-state. The Newton iteration used to adjust the initial condition is given by:

$$q_0^{i+1} = q_0^i - \left[I - \frac{\partial q^i(q_0, T)}{\partial q_0^i}\right]^{-1} (q_0^i - q^i(q_0, T)) \qquad (2.3)$$

where q_0 is an R^n initial condition vector, $q(q_0, T)$ is the solution after one full period beyond the initial point t_0, and i is the iteration number.

Equation (2.3) can be set in a more suitable form:

$$J^i \Delta q_0^i = E^i \quad , \tag{2.4}$$

where

$$J^i = I - D^i = I - \frac{\partial q^i (q_0, T)}{\partial q_0^i} \quad , \tag{2.5a}$$

$$\Delta q_0^i = q_0^i - q_0^{i+1} \quad , \tag{2.5b}$$

$$E^i = q^i (q_0, T) - q_0^i \quad . \tag{2.5c}$$

The term $D^i = \frac{\partial q^i (q_0, T)}{\partial q_0^i}$ is called the sensitivity matrix, as it gives the variation of the final state at, $t = t_0 + T$, w.r. to a perturbation in the initial conditions at, $t = t_0$.

Hence, if the Jacobian J and the right hand side vector E can be determined, equation (2.4) can be solved to give $\Delta q_0^i = q_0^i - q_0^{i+1}$. The iteration will be terminated if $|\Delta q_0^i|$ is less than a specified limit.

The following are the main steps in the Newton algorithm:

Algorithm 2.1

(1) Start with an initial guess q_0^o (say, by integrating for a few periods).

(2) $i = 0$.

(3) Integrate further one complete period and determine $q^i(q_0, T)$; hence determine E^i .

(4) Compute J^i (may be done simultaneously while determining E^i) .

(5) Solve equation (2.4) (may be done using L U factorization) for Δq_0^i and thence determine q_0^{i+1} .

(6) Is $|\Delta q_0^i|$ less than certain acceptable limit?

If yes, STOP (the steady state solution is found)

If not:

$i = i + 1$.

IF i is greater than a certain limit, STOP.

If not, put q_0^{i+1} as the new initial conditions, and go to Step 3.

If the order of the system n (number of storage elements) is small, this algorithm seems to be very attractive as it has remarkably good convergence criteria. But in the case of high order systems (say $n \geq 50$), the drawbacks will outweigh the convergence properties.

Firstly, the cost of computing the Jacobian J. will be very high, since it involves the analysis of n circuits (adjoint networks, or sensitivity circuits). Secondly, the solution of equation (2.4) involves full matrices of order n which, in addition to slowing down the execution speed, has some critical numerical stability problems. Thirdly, it will not be possible to make use of the sparse matrix techniques used in most recent programs, for solving equation (2.4).

The heart of the above stated algorithm is the computation of the Jacobian J, which can be achieved using one of two methods: the sensitivity circuits approach [4], or the Adjoint Network method [6]. These will now be briefly discussed.

2.1.1 Computation of the Jacobian J Using the Sensitivity Circuit Approach

From equation (2.5a)

$$J^i = I - D^i = I - \frac{\partial q^i (q_0, T)}{\partial q_0^i} .$$

Put

$$D^i = \frac{\partial q^i (q_0, T)}{\partial q_0^i} = [D_1^i , D_2^i , \ldots , D_k^i , \ldots D_n^i] \quad (2.6)$$

where

$$D^i_k = \frac{\partial q^i (q_0, T)}{\partial q^i_{0_k}} \qquad (2.7)$$

is the kth column of the sensitivity matrix D^i.

To show how equation (2.7) can be computed, it is necessary to examine the sensitivity circuit [4], and thence to show how the computation of each column of the sensitivity matrix D_k will correspond to the solution of a sensitivity circuit, after one full period with specified initial conditions.

Consider circuit equations defined in tableau form by

$$A\ i = 0 \qquad K\ C\ L \qquad (2.8a)$$

$$A^t v = \mathring{E} \qquad K\ V\ L \qquad (2.9a)$$

The branch constitutive (B.C.) relations - self or mutually dependent - can be set in a general form.

For a non-reactive branch,

$$v_j = v_j (i_m) \qquad (2.10a)$$

For a capacitive branch,

$$q_{c_j} = q_{c_j} (v_m) \qquad (2.11a)$$

where

$$i_{c_j} = \frac{d\, q_{c_j}}{d\, t} \quad , \quad \text{and} \quad v_m(0) = v_{m0}$$

For an inductive branch

$$F_{\ell_j} = F_{\ell_j}(i_m) \tag{2.12a}$$

where

$$v_{\ell_j} = \frac{d\, F_{\ell_j}}{d\, t} \quad , \quad \text{and} \quad i_m(0) = i_{m0}$$

Finally, for independent source

$$I_s = I_s(t) \qquad \text{(current source)} \tag{2.13a}$$

$$E_s = E_s(t) \qquad \text{(voltage source)} \tag{2.14a}$$

Assuming that the system has a unique solution, let us perturb the initial conditions of the differentiated variables (generally, we will call them q_0) one at a time. The result is obtained by differentiating equation (2.8a) through (2.14a) with respect to the kth differentiated variable q_{0_k}

$$A \frac{\partial i}{\partial q_{0_k}} = 0 \tag{2.8b}$$

$$A^t \frac{\partial v}{\partial q_{0_k}} = 0 \tag{2.9b}$$

$$\frac{\partial v_j}{\partial q_{0_k}} = \frac{\partial v_j}{\partial i_m} \cdot \frac{\partial i_m}{\partial q_{0_k}} \qquad (2.10b)$$

$$\frac{\partial q_{c_j}}{\partial q_{0_k}} = \frac{\partial q_{c_j}}{\partial v_m} \cdot \frac{\partial v_m}{\partial q_{0_k}} \qquad (2.11b)$$

where

$$\frac{d}{dt}\left(\frac{\partial q_{c_j}}{\partial q_{0_k}}\right) = \frac{\partial i_{c_j}}{\partial q_{0_k}} ,$$

and

$$\frac{\partial v_{m0}}{\partial q_{0_k}} = \begin{cases} 1 & \text{if } q_k \text{ corresponds to the differentiated controlling variable } v_m . \\ 0 & \text{if } q_k \text{ does not correspond to } v_m . \end{cases}$$

$$\frac{\partial F_{\ell_j}}{\partial q_{0_k}} = \frac{\partial F_{\ell_j}}{\partial i_m} \cdot \frac{\partial i_m}{\partial q_{0_k}} \qquad (2.12b)$$

where

$$\frac{d}{dt}\left(\frac{\partial F_{\ell_j}}{\partial q_{0_k}}\right) = \frac{\partial v_{\ell_j}}{\partial q_{0_k}} ,$$

and

$$\frac{\partial i_{m0}}{\partial q_{0_k}} = \begin{cases} 1 & \text{if } q_k \text{ corresponds to the differentiated controlling variable } i_m . \\ 0 & \text{if } q_k \text{ does not correspond to } i_m . \end{cases}$$

$$\frac{\partial I_s}{\partial q_{0_k}} = 0 \qquad (2.13b)$$

$$\frac{\partial E_s}{\partial q_{0_k}} = 0 \qquad (2.14b)$$

A close examination of the corresponding pairs (a - b) of equations (2.8 - 2.14) will show that equations (2.8b) - (2.14b) describe a circuit having the following properties:

1. ~~Linear~~ circuit (time-varying if the original circuit contains nonlinear elements) having the same topology as the original circuit;

2. the type of an element is the same as that of the corresponding element of the original circuit;

3. the value of an element

   (i) for a linear element is the same as that of the original;

   (ii) for a nonlinear element is replaced by the piecewise linearization at each time point;

   (iii) of an independent source is replaced by a zero source (i.e., short circuit for an independent voltage source, and open circuit for an independent current source) ;

4. the initial conditions of the differentiated variables are replaced by a source of unity value if the differentiation is taken w.r. to this variable, and with a zero valued source if not.

This circuit is called the sensitivity circuit. It should be noted that, corresponding to each reactive element, there must be an equivalent sensitivity circuit. By solving the kth sensitivity circuit over one complete period starting at the initial point t_0, the solution vector of the differentiated variables at the end of the period is just the kth column D_k of the sensitivity matrix D.

From a computational point of view, only one circuit (the original) is formed and the same coefficient Jacobian matrix, describing the linearized system of equations (in tableau form), is used but with a different forcing vector for the computation of each D_k. To show how this can be done, consider the general linearized tableau form

$$B \begin{bmatrix} w \\ q \end{bmatrix} = [\, E \,] \qquad (2.15)$$

where B is the coefficient Jacobian matrix, w the algebraic variable vector, q the differential variable vector, and E is the forcing vector.

The tableau form for the kth sensitivity circuit is

$$B \begin{bmatrix} u \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \end{bmatrix} \quad ; \quad \lambda(0) = e_k \tag{2.16}$$

where e_k is the kth unity vector (i.e., vector of all zeros except the kth element value is unity), and u, λ are, respectively, the vectors of the algebraic and differentiated variables.

The above method can be implemented by means of the following algorithm for computing the sensitivity matrix D_{NxN} where $J = I - D$:

Algorithm 2.2

1. Start at an initial time point t_0 ;
2. $k = 0$;
3. form the coefficient Jacobian matrix (B) of the original circuit; save the nonlinear entries (if any) ;
4. solve the original system of equations (2.15) and save the solution vector and the necessary variables ;
5. $k = k + 1$;
6. load the Jacobian B (using the stored values of the nonlinear entries (if any)) ;
7. If it is the first time point, put $\lambda(0) = e_k$.
8. Solve the kth system of equations (2.16) and save the solution vector and the necessary variables (e.g., variables required to compute $\dot{\lambda}$) .

9. If k less than N go to step 5.

10. If $t < t_0 + T$, choose the next step size, and go to step 2.

11. Form the sensitivity matrix D, by column, where the kth column D_k is the solution vector of the differentiated variables of the kth system. (Then $J = I - D$).

2.1.2 Computation of the Jacobian J
Using the Adjoint Network Approach

Transform equation (2.6) into the form

$$D^i = \frac{\partial q^i (q_0, T)}{\partial q_0^i} = [D_1^{i\,t}, D_2^{i\,t}, \ldots, D_k^{i\,t}, \ldots D_N^{i\,t}]^t \quad (2.17)$$

where

$$D_k^{i\,t} = \left[\frac{\partial q_k^i (q_0, T)}{\partial q_0^i} \right]^t , \quad (2.18)$$

is the kth row of the sensitivity matrix D.

Before describing an algorithm for computing D, it is necessary to review the adjoint network concept [6], and its application [5].

Consider an original network, with current and voltage variables i and v, respectively, and a corresponding "adjoint" network with current and voltage variables $\hat{i}$ and $\hat{v}$, respectively. Let us assume the "adjoint" network to have the same topology as the original network. In that case, using Tellegen's theorem [27]

$$\int_{t_0}^{t_0+T} \sum_j \left[v_j(t)\, \hat{i}_j(\tau) - i_j(t)\, \hat{v}_j(\tau) \right] dt = 0 \qquad (2.19)$$

where τ is the arbitrary time variable of the adjoint network.

If the original network variables are perturbed by Δv_j, Δi_j, Tellegen's theorem still holds for the perturbed states, i.e.

$$\int_{t_0}^{t_0+T} \sum_j \left[\Delta v_j(t)\, \hat{i}_j(\tau) - \Delta i_j(t)\, \hat{v}_j(\tau) \right] dt = 0 \qquad (2.20)$$

Using the assumptions given in [5], the adjoint network can be extracted from the original network using the following properties:

1. The time variable of the adjoint system is $\tau = t_0 + t_f - t$, where the final time $t_f = t_0 + T$.

2. The type of an adjoint network element is the same as the corresponding one in the original network and both networks have the same topology.

3. The value of an adjoint element is given by:

(i) for a linear element, its value;

(ii) for a nonlinear element, the piecewise linearization of the original element value. Thus the corresponding adjoint element is linear and time varying.

(iii) The independent source is replaced by a zero source (i.e., independent voltage source is replaced by a short circuit, the independent current source is replaced by an open circuit).

It should be noted [5] that, for both original and adjoint networks each reactive element with initial condition x_0 has to be replaced by a similar reactive element with zero initial conditions accompanied by an auxiliary source (parallel current source for inductive element or series voltage source for capacitive element). The value of that auxiliary source is equal to the initial condition of the corresponding branch.

Now there remains the problem of setting the initial conditions of the adjoint reactive elements. Suppose we are computing the kth row of the sensitivity matrix, D_k. There are two cases [5] to be considered.

(i) If the kth element is a capacitor, the initial condition of the kth adjoint network is

$$\hat{q}_{0_k} = \frac{1}{C_k} \cdot e_k \qquad (2.21a)$$

where e_k is equal to (0, 0, ..., 1, .., 0) with the one in the kth position. It can be shown that if the mth element of the kth adjoint network is capacitive,

$$\frac{\partial v_{c_k}(t_0 + T)}{\partial v_{c0_m}} = C_m \left(\hat{i}_{c_m}(t_0 + T) - \hat{i}_{c_m}(t_0) \right). \qquad (2.22a)$$

But if it is an inductor

$$\therefore \quad \frac{\partial v_{c_k}(t_0 + T)}{\partial i_{\ell 0_m}} = - L_m \left(\hat{v}_{\ell_m}(t_0 + T) - \hat{v}_{\ell_m}(t_0) \right) \qquad (2.23a)$$

(ii) If the kth element is an inductor hence the initial condition of the kth adjoint system is

$$\hat{q}_{0_k} = - \frac{1}{L_k} \cdot e_k . \qquad (2.21b)$$

If the mth element of the kth adjoint system is a capacitor

$$\therefore \quad \frac{\partial\, i_{\ell_k}\,(t_0 + T)}{\partial\, v_{c0_m}} = C_m \left(\hat{v}_{c_m}\,(t_0 + T) - \hat{v}_{c_m}\,(t_0)\right) \qquad (2.22b)$$

But if it is an inductor

$$\therefore \quad \frac{\partial\, i_{\ell_k}\,(t_0 + T)}{\partial\, i_{\ell 0_m}} = -L_m \left(\hat{i}_{\ell_m}\,(t_0 + T) - \hat{i}_{\ell_m}\,(t_0)\right) \qquad (2.23b)$$

Due to the restriction imposed on the adjoint time variable $\tau = t_0 + t_f - t$, the coefficient Jacobian of the adjoint networks is not computed at the same time point as the original network. So the adjoint computations have to be delayed until after the original network integration over the whole period $(t_0$ to $t_0 + T)$, and the necessary variables have to be stored. Then, after finishing the original system integration, the adjoint system can be integrated in the opposite direction, using the stored information. This is equivalent to integrating backward from $t = t_0 + T$ up to t_0. From this discussion it should be clear that the adjoint network approach will slow down the execution speed relative to the sensitivity circuit approach. Also it may cause programming difficulties related to the retrieval of information from disk and the interpolation process as well.

A computational algorithm can be established for computing the sensitivity matrix D, using the adjoint network approach.

Algorithm 2.3

1. Start at an initial time t_0. Integrate the original system up to a final time $t_f = t_0 + T$. At each (accepted) time point, store the necessary values of the nonlinear elements (if any).

2. Construct the coefficient Jacobian of the adjoint system and generate the part of the code that perform the LU factorization. Hence, for each adjoint system, generate the other parts of the code (such as that required for the forward and backward substitution) to be used for integrating each system, and save these codes.*

3. k = 0 ;

4. k = k + 1 ;

5. If it is the first point, put the initial conditions according to equations (2.21a, b);

6. Load the Jacobian and the necessary variables. Solve the system (by executing the generated code)

---

* This step is oriented for a specific time domain analysis program (SCAMPER) in which the integration is done using a generated machine code string. But it can be modified depending on the features of the program used in the analysis.

and save the necessary variables (such as those needed for computing the derivatives $\dot{x}$).

7. If k less than N go to step 4.

8. If $t \leq t_0$ go to step 9 ;
If $t > t_0$, choose the next step size and go to step 3 ;

9. Formulate the matrix D, by row, using the solution vectors of the adjoint systems, where the kth row is that corresponding to the kth adjoint system.

2.2 The Extrapolation Methods

The extrapolation methods [16] are based on the ε - algorithm [15] which does not require the calculation of derivatives (such as the gradient in the optimization-like algorithms or the sensitivity matrix in the Newton methods). But, in place of this, for a system with N independent differentiated variables, $2M$ solution vectors ($M \leq N$) must be determined over a time interval $2MT$ in order to carry the algorithm one step, where T is the period.

Originally, the ε - algorithm proposed by Wynn [15] was oriented to the solution of systems of linear differential equations. With mildly nonlinear systems, and near the equilibrium (steady-state

condition), the algorithm, hopefully, can be applied to find this solution with a quadratic convergence [12], [13].

2.2.1 The ε - Algorithm

The ε - algorithm is a nonlinear sequence-to-sequence transform. Assume that the system solution sequence is $q_0, q_1, \ldots$ which may be produced by any integration process. Also assume vectors $\{ \varepsilon_k^r \}$ where k represents the transformation step and r is the sequence number at this step. If initial vectors are given by:

$$\varepsilon_{-1}^{(r)} = 0 \quad ; r = 1, 2, \ldots 2m \tag{2.24a}$$

$$\varepsilon_0^{(r)} = q_r \quad ; r = 0, 1, \ldots 2m \tag{2.24b}$$

The recursion formula

$$\varepsilon_{k+1}^{(r)} = \varepsilon_{k-1}^{(r+1)} + \left(\varepsilon_k^{(r+1)} - \varepsilon_k^{(r)} \right)^{-1} \quad ; k, r = 0, 1, \ldots \tag{2.25}$$

is known as the ε - Algorithm.

There are two forms of the ε - algorithm, depending on the definition of the inverse of

$$x = (x_1, x_2, \ldots x_N) .$$

- The scalar ε - algorithm is based on the primitive inverse definition

$$x^{-1} = (x_1^{-1}, x_2^{-1}, \ldots, x_N^{-1}) ,$$

so that each component is treated independently.

- The vector ε - algorithm involves the Euclidean norm definition of the inverse,

$$x^{-1} = x / ||x||_2^2 \quad ; \quad ||x||_2^2 = \sum_{i=1}^{N} x_i^2$$

The following discussion will be restricted to the vector ε - algorithm because the organization of the computation is easier and, moreover, there is some evidence that this form is more stable [12].

It was shown by McLeod [14] that if a sequence $\{q_r\}$ satisfies the linear recursion form

$$\sum_{i=0}^{m} \alpha_i \, q_{r+i} = \left(\sum_{i=0}^{m} \alpha_i\right) q \quad ; \quad r = 0, 1, \ldots \tag{2.26}$$

where $\{\alpha\}$ is a real vector and q is a constant vector, then

$$\varepsilon_{2m}^{(r)} = q \; ; \; r = 0, 1, \ldots, \text{ if } \sum_{i=0}^{m} \alpha_i \neq 0 \tag{2.27a}$$

or

$$\varepsilon_{2m}^{(r)} = 0 \ , \quad r = 0, 1, \ldots, \quad \text{if} \quad \sum_{i=0}^{m} \alpha_i = 0 \tag{2.27b}$$

In the case of a nonlinear system, where the solution sequence $\{q_k\}$ is produced, say, by the contraction mapping

$$q_{r+1} = F(q_r) \ ,$$

the ε-algorithm can be applied if [12], [13]

$$q_{r+1} - q = F'(q)(q_r - q) + 0\| q_r - q \|_2^2 \tag{2.28}$$

where this notation means the last term has norm of the order $\| q_r - q \|_2^2$.

Under this condition, the convergence of the sequence q_r to the fixed point q of the mapping F is at least quadratic.

It was stated at the beginning that an integration over $2M$ periods is required for each iteration, hence, the determination and minimization of M is of great importance. This point has been studied by Skelboe [16] who examined closely each circuit and, in particular, the relative values of the circuit time constants and the period(s) of the forcing function(s). The maximum value of M is the order of the system N i.e., the number of independent reactive elements. One way of reducing M is by the proper choice of the starting time t_0. This point can be selected such that some of the faster transients will have died out by $t = t_0$.

From the previous discussion, an executable procedure, based on the ε - algorithm can be stated:

Algorithm 2.4

1. Initialization step:

   (i) $k = -1$

   (ii) Define (m), the starting point (t_0) and the acceptable error limit (δ).

   (iii) Integrate over $t \in (0, t_0)$ and set:

   $$z_k = q(t_0)$$

2. Integration step:

   (i) $k = k + 1$

   (ii) $q_0 = z_{k-1}$

   (iii) Integrate over $t \in (t_0, t_0 + 2mT)$ and set:

   $$q_r = q(t_0 + rT) \quad , \quad r = 1, 2, \ldots 2m$$

3. ε - algorithm step:

   (i) put:

   $$\varepsilon_{-1}^{(r)} = 0 \quad , \quad \varepsilon_0^{(r)} = q_r \quad , \quad r = 0, 1, \ldots 2m$$

   and

   $$j = -1$$

(ii) Solve:

$$\varepsilon_{j+1}^{(r)} = \varepsilon_{j+1}^{(r+1)} + (\varepsilon_j^{(r+1)} - \varepsilon_j^{(r)})^{-1} \quad , \quad r = 0, 1, \ldots, 2m-1$$

$$j = j + 1$$

(iii) If $j < 2m - 1$, go to step (3 - ii).

4. Test of convergence step:

(i) $z_k = \varepsilon_{2m}^{(0)}$

(ii) If $||z_k - z_{k-1}||_2^2 \leq \delta$ the steady-state solution is found by using z_k as the initial condition at $t = t_0$. STOP.

If not, go to step 2.

The computations in step (3) involve a tradeoff between execution time and storage requirements. If the execution time is to be minimal, at least 2 x (2M) vectors (each having N entries) must reside in the high speed memory, i.e., storage allocation of at least 4MN entries is required, which is four times bigger than that required for the Newton algorithm. If this storage area is to be minimized, then 2M retrievals of data from disk memory are required, which will slow down the execution speed.

2.3 Gradient Method

The gradient method for the periodic steady-state analysis of nonlinear circuits was first proposed by Nakhla and Branin [10], [11]. Their implementation was based on the state variable formulation of the circuit equations. Although one of the objectives of this work was to make use of the gradient method in conjunction with the more general and convenient tableau representation of the circuit, the succeeding discussion will be based on the notationally simpler state variable approach as given in [11]. The detailed analysis of a more general tableau representation is left for a later Chapter (III).

Let the state variable equations describing the circuit be given by:

$$\dot{q} = f(q, t) \tag{2.29}$$

where f and the differentiated variables $q \in R^N$. Given that the forcing function is periodic and of period T, it is required to find a set of initial conditions $q(t_0) = q_0$ such that

$$q(t_0) = q(t_0 + kT) \quad , \quad k = 1, 2, \ldots . \tag{2.30}$$

For convenience $k = 1$. The deviation of the circuit from its periodic steady-state is expressed in terms of a performance function Φ which is defined by

$$\Phi = [\delta(q_0)]^t [\delta(q_0)] \quad , \tag{2.31}$$

where $\delta\ (q_0)$ is the discrepancy vector given by

$$\delta\ (q_0) = q\ (t_0) - q\ (t_0 + T) \quad , \tag{2.32}$$

and the superscript t indicates transposition.

Thus the objective is to find a set of q_0 which will minimize Φ .

The nonlinear programming technique used to minimize Φ requires the first derivatives (gradients) $\partial\ \Phi\ /\ \partial\ q_0$. The crux of the method is to demonstrate an efficient scheme for computing the gradient $G = \partial\ \Phi\ /\ \partial\ q_0$. It has been shown [11] that the gradient can be computed by only integrating two systems of equations over one complete period: the original system of equations from $t = 0$ to $t = T$ and a second system, which will be referred to as the adjoint variational system, from $t = T$ to $t = 0$. From theorem 2.1 [11] and equations (2.29) to (2.32) it can be proved that the gradient G has the simple form

$$G = \frac{\partial\ \Phi}{\partial\ q_0} = 2\ [\ \lambda\ (\delta\ ,\ 0\ ) - \delta\ (\ q_0\ )\ ] \tag{2.33}$$

where $\lambda\ (\ \delta,\ t)$ is the solution of the adjoint variational system

$$\dot{\lambda} = -\ [\ \frac{\partial\ f}{\partial\ q}\ ]^t \Big|_{q\ (q_0,\ t)} \lambda \tag{2.34}$$

with initial conditions

$$\lambda\ (\delta\ ,\ T)\ =\ \delta\ (q_0) \tag{2.35}$$

From equations (2.33) to (2.35) it is evident that the adjoint system of equations has to be integrated backwards from $t = T$ to $t = 0$. This requires knowledge of the initial conditions $\lambda\ (\delta\ ,\ T) = \delta\ (q_0)$ and the trajectory of the coefficients of the Jacobian matrix $\frac{\partial f}{\partial q}$ over the whole period. The initial conditions are given simply by the difference between the solutions of the original system at $t = 0$ and $t = T$. The Jacobian $\partial f / \partial q$ can be formed from the forward integration of the original system using one of two methods, namely, by either saving the nonlinear entries of $\partial f / \partial q$ directly, or by saving the necessary variables from which $\partial f / \partial q$ can be evaluated.

A computational algorithm can be written on the basis of the above discussion [10], [11].

Algorithm 2.5

1. Choose a first estimate of q_0, perhaps by integrating equation (2.29) until a predefined time t_0.
2. Using q_0 as initial conditions, integrate equation (2.29) for one full period, saving the necessary variables to compute the Jacobian $\partial f / \partial q$.
3. At $t = t_0 + T$ compute the discrepancy vector $\delta\ (q_0)$

defined by equation (2.32). Hence compute the performance function Φ defined by equation (2.31). If Φ is less than or equal to an acceptable limit, the steady-state solution is found. STOP.

4. Otherwise, use $\delta(q_0)$ as initial conditions of the adjoint system defined by equation (2.34), and integrate backwards over one full period from $t = T$ to $t = 0$ using the saved values from the forward integration (in reverse order) to compute the Jacobian.

5. Compute the gradient G defined by equation (2.33).

6. Pass the arguments q_0, gradient G and the objective function Φ to the optimization routine.

7. Go to step (2) using the new arguments q_0 returned by the optimization routine as initial conditions.

2.4 Comparison of Methods

Three possible methods for computing the periodic steady-state response of nonlinear circuits have been reviewed. In order to

arrive at a decision as to which method is the most suitable, it is necessary to define the class of problems which has to be solved. The present work was undertaken with the objective of finding a method which would not only be used to compute economically the periodic steady-state response of large, stiff and nonlinear systems, but which could also be easily extended to include circuit optimization and sensitivity analysis. In the following sections the three methods are compared with respect to those characteristics which would have the greatest influence on the final choice.

2.4.1 Computation Cost and Convergence Criteria

Let us first consider the convergence problem. For the gradient method (GM), the rate of convergence depends essentially on (1) the stiffness of the network equations, (2) the initial state of the network relative to the steady-state conditions, and (3) the optimization routine used for minimizing the objective function Φ. With the optimization routine based on Fletcher's [22] variable metric method, the GM has been demonstrated by Nakhla [11] to converge quadratically near the equilibrium (steady-state) conditions. When the network initial conditions are quite far from the steady-state, the method still converges but at a slower rate. In the case of the Newton method (NM), a comparison of results reported by Trick [3] and Nakhla [11] shows that the NM is more likely to converge faster

than the GM but with a narrower range of convergence. The convergence of NM depends on (1) the stiffness of the Network equations, and (2) the state of the network relative to the steady-state [1]. For the extrapolation method (EM), the few examples reported by Skelboe [16] seem to indicate that the rate of convergence of the EM is at least quadratic with a range as wide as that of the GM. In comparison to the Newton method [1], [3], the EM [16] seems to be even faster. The convergence of the EM depends on (1) the stiffness of the network equation, and (2) the mildness of the network non-linearities.

From the above discussion and in the light of the published results for a relatively small number of modest size problems, it appears that, within the range of problems where the three methods GM, NM, and EM are applicable, the extrapolation method converges faster than the two other methods while the gradient method (with the variable metric optimization routine) is the slowest. But the number of iterations required for convergence by each of these three methods differ by ratios which are not so far from unity.

Let us turn now to the problem of computation cost. Assume that the network equations are set in the form

$$J\ X = E \quad ,$$

where J is the Jacobian matrix, and E is the forcing vector.

Suppose that a LU factorization method is to be used for solving the network equations at each integration step, where the number of steps over one full period is n_s*. The major portion of the computation cost per step can be deduced by considering the costs of (1) computing and loading the Jacobian (J), (2) computing the forcing vector (E), (3) factoring the matrix (LU), and (4) the forward and backward substitution (FB).

Now, for the gradient method it has been shown that an integration of two systems (the original system in the forward direction, and the adjoint system in the backward direction) over one full period is required for each iteration. If we assume that the cost of integrating the adjoint system is the same as that for the original one, and that the number of iterations required for convergence is n_G, then, the integration cost of the gradient method is

---

* In some time-domain analysis programs, such as NDP [17], the integration process proceeds as follows: at each time point, the system is solved with a predetermined step size h. If the system does not converge, a Newton iteration is applied repeatedly (with the same step size) k times until it converge. If we consider the number of time steps equal to n_t, and the average number of Newton iterations per step is k, then the number of integration steps n_s will be given by $n_s = n_t k$.

In some other programs, such as SCAMPER [19], [20], the integration process is as follows: at each time point a step size h is chosen and the system of equations is solved. If it does not converge, the step size is reduced until convergence occurs. In this case, the number of integration steps n_s will be the number of solution trials (both accepted or not accepted).

$$GM = 2 \cdot n_s \cdot \eta_G \cdot (J + E + LU + FB) \quad (2.36)$$

For the Newton method, in addition to the original system, an integration of N adjoint systems (N sensitivity circuits [4] simultaneously with the original system, or N adjoint networks [5] in the reverse (backward) direction) must be carried out, where N is the number of reactive elements in the network. Consider the case [4], where the integration of the N sensitivity circuits is carried out simultaneously with integrating the original system. The integration of the N sensitivity circuits can be carried out using the same Jacobian J and LU factorization used for integrating the original system. Hence the computation cost of the Newton method can be approximately given by:

$$NM = n_s \cdot \eta_N \cdot [J + LU + N. (E + FB)] \quad (2.37)$$

where η_N is the number of iterations required for convergence.

For the extrapolation method, it has been shown that an integration of one system (the original) has to be carried out over 2M periods per iteration where $M \leq N$. Accordingly, the computation cost of the extrapolation method will be given by

$$EM = 2 \cdot n_s \cdot M \cdot \eta_E \cdot (J + E + LU + FB) \quad (2.38)$$

where η_E is the number of iterations required for convergence.

From equations (2.36) - (2.38) the relative cost of the three methods can be given by

$$GM : NM : EM = 2 \cdot \eta_G \cdot (J + E + LU + FB) : \eta_N \cdot [J + LU + N \cdot (E + FB)] : 2 \cdot M \cdot \eta_E \cdot (J + E + LU + FB) \tag{2.39}$$

In order to simplify this equation, let us consider a practical situation. As mentioned before, we are interested in moderate and large scale nonlinear networks, with quite a large number of reactive elements N ($N \geq 20$). Suppose that the network equations are to be written in the tableau form with the number of equations equal to NE (which is proportional to the number of branches in the network). Consequently, the Jacobian J will be sparse. For sparse matrices with quite large dimension NE, Norin and Pattle [26] reported that LU and FB requires O (NE) operations, with LU = 0.4 O(NE), FB = 0.6 O (NE), approximately. In general we can write

$$LU = \alpha_{LU} \cdot NE$$

$$FB = \alpha_{FB} \cdot NE \tag{2.40}$$

The cost of computing and loading the Jacobian J and the forcing vector E depends on (1) the number of equations NE, (2) number and types of nonlinear elements, and (3) the number of reactive elements. For simplicity, let us assume that these two costs, J and E, depends linearly on the number of equations NE such that

$$J = \alpha_J \cdot NE$$

$$E = \alpha_E \cdot NE \qquad (2.41)$$

It has been found experimentally* that the cost of J or E is less than that for LU or FB (e.g., $\alpha_E \approx 0.3\ (\alpha_{LU} + \alpha_{FB})$). If we normalize the units of the computation cost such that $\alpha_J + \alpha_E + \alpha_{LU} + \alpha_{FB} = 1$, then equation (2.39) can be replaced by

$$GM : NM : EM = 2\ \eta_G : \eta_N\ [\alpha_J + \alpha_J + \alpha_{LU} + N\ (\alpha_E + \alpha_{FB})] : 2 \cdot M \cdot \eta_E \qquad (2.42)$$

---

* A network, with the number of branches NB = 78 of which 40 were nonlinear, and the number of tableau equations NE = 199, was analysed using SCAMPER [19], [20] with the steady-state algorithm included (see Chapter V). Both J and E are computed in part using Fortran statements and in part using machine instructions (code), while LU and FB are done using a generated code only. The length of the codes for this problem was as follows:

LU + FB = 7083 , E = 1498 , J = 146 .

The computation of J was mainly by Fortran statements, while the computation of E was mainly using the generated code (E code). From a close examination of the network we expect that the total cost of E will be approximately 50% higher than the number given above. Also we expect the cost of computing the Jacobian to be less than or equal to the cost of computing E. According to these estimates, a rough cost comparison can be given as follows:

E, J ≈ 1.5 (1498 / 7083) (LU + FB)

i.e., E, J ≈ 0.3 (LU + FB) .

Assuming M is of the order of N ($M \leq N$), it appears that for networks with quite a large number of reactive elements ($N > 20$), the gradient method yields the lowest computational cost provided that η_G, η_N and η_E are approximately equal.

2.4.2 Network Size and the Numerical Stability Problem

For the Newton method, the sensitivity matrix $\partial q (q_0, T) / \partial q_0$ is, in general, a full matrix. When the order of the system N becomes large (say $N > 50$), difficulties may be encountered because : (1) a solution of a large scale system with a full Jacobian matrix may present serious numerical stability problems as well as poor accuracy due to round-off error ; (2) the storage requirements for the Jacobian is of the order of N^2 which may be prohibitively large.

In the case of the extrapolation method, the numerical properties of the method are less apparent due to the nonlinearity of the ε - algorithm step used with this method. But it was reported by Brezinski [12] that the round-off error may cause severe numerical problems especially for stiff systems. Furthermore, it was shown previously that storage of order N^2 is required to minimize the execution time.

For the gradient method, the optimization step does not require solution of simultaneous set of equations (even with the variable metric method [22], the only matrix operations involved are addition or subtraction). Accordingly, there will be none of the numerical problems which may be encountered with the other methods. What concerns the storage (hence the order of the system N) problem, if a variable metric algorithm is used for the optimization, then storage of the order of N^2 is required just as for the other two methods. But if a conjugate gradients method could be used for the optimization, then storage of only order N will be required.

In conclusion, it can be said that the gradient method does not have the numerical stability problems such as those that may be encountered with both the Newton and the extrapolation methods for large scale systems. Furthermore, the three methods suffer from the same storage problem except that the gradient method would not have this problem if a conjugate gradients method (which does not involve matrices) could be used for the optimization.

CHAPTER III

A GENERALIZED APPROACH TO THE GRADIENT METHOD

3.1 Introduction

In Section 2.3 a gradient method for determining the periodic steady-state response, based on the state variable approach [10], [11], was outlined. Although an extension to the general algebraic-differential formulation of the system equations was also given [10], [11], it is still far from being directly implementable within most third generation time domain analysis programs which are based on the tableau formulation of the Network equations to take advantage of highly sophisticated sparse matrix techniques. In this chapter a more general and convenient formulation of the gradient method will be derived which can be implemented directly within most of the recent time domain analysis programs.

The derivation involves the use of the adjoint system concept. In Director's [6] treatment of the adjoint network, the parametric representation of branch types involved one dependency on a circuit variable as well as a set of design parameters, and the derivation of the adjoint network was carried out type-by-type. In the present approach to the derivation of the adjoint variational system, the equations describing the network are set in a general form, but with an assumed upper limit of two on the number of dependencies (linear or nonlinear). During the course of the derivation it is shown that dependencies within these limits are acceptable for representing the

adjoint system, with the implicit assumption that the dependency can be represented as an element in the original network.

The approach in the present work is similar to that of Director [8] but the equation formulation is more general. After a general adjoint system as well as a gradient form are developed, two useful objective functions, reflecting the state of the network relative to the steady-state are considered. One of these, based on normalizing the contribution of each reactive element into the weighted square of voltage, is selected. On the basis of this choice a more specific approach is taken, which leads to convenient forms of the equations for computing the gradient, as well as for the initial conditions of the adjoint system. The chapter ends with a computational algorithm suited to application within a general time domain analysis program such as SCAMPER [19], [20].

3.2 Derivation of the Gradient Equations Based on a Generalized Adjoint Variational System

In order to minimize an objective function Φ, which reflects how far a circuit is from the periodic steady-state, it is necessary to have an efficient procedure for computing the gradient $G = \partial \Phi / \partial q_0$ where q_0 is the initial condition vector of the differentiated variables representing the system. To this end, we formulate the original system of equations in the general tableau form.

$$h\,(u, q, t) = 0 \quad ; \text{Algebraic equations} \tag{3.1a}$$

$$g\,(u, q, \dot{q}, q_0, t) = 0 \; ; \text{Differential equations} \tag{3.1b}$$

where

u is an algebraic variables vector ,

q is a differentiated variables vector ,

$q_0 = q\,(t_0)$ is the initial condition vector ,

$\dot{q} = \dot{q}\,(q_0, t)$.

In recent time domain, circuit analysis programs the solution of the system of equations (3.1) at any time point is achieved by linearizing the nonlinear terms at a point (the last accepted point or a predicted one), and replacing the differentiated terms $\dot{q}$ by the slope of a polynomial of order $R \leq 6$ [28]. The linearized equations corresponding to (3.1) can be expressed in the form

$$J \begin{bmatrix} u \\ q \end{bmatrix} = E\,(t) \quad , \tag{3.2}$$

where J is the coefficient Jacobian matrix, and $E\,(t)$ is the forcing vector.

Before giving a detailed representation of J, it is important to specify limits on the acceptable range of branch types and dependencies. We shall use the following equation to describe an element or a part of a controlled (coupled) branch;

$$x_i = \alpha (x_r) \cdot x_j \qquad (3.3)$$

where x may be an algebraic (u) or differentiated (q) variable. In this way branches ranging from linear self-dependent to nonlinear with two dependencies will be considered within the scope of our analysis*.

Returning to the linearization of the system of equations, equation (3.1b) can be set in the form

$$M u + A \frac{d q}{d t} = 0 \qquad (3.4)$$

where M is a matrix of 0 or ± 1 entries,

and $A = [a_{ij}]$, where

$a_{ij} = 0, \pm 1$ for linear elements,

$= a_{ij} (q_j)$ for a nonlinear single dependency, (self or mutual),

$= a_{ij} (x_r)$, $r \neq j$ for nonlinear double dependency.

From equations (3.1a) and (3.4) the linearized equations of the original system can be expressed in the form

---

* For more details about the different types of branches under consideration, see Part A of Chapter V.

$$\begin{bmatrix} \frac{\partial h}{\partial u} & \frac{\partial h}{\partial q} \\ M & A\frac{d}{dt} \end{bmatrix} \begin{bmatrix} u \\ q \end{bmatrix} = E(t) \qquad (3.5)$$

In general, the objective function Φ, which reflects the deviation from the periodic steady-state, can be defined in an integral form

$$\Phi = \Phi(u, q, q_0, t_0) = \int_{t_0}^{t_0+T} \phi(u, q, q_0, t)\, dt \qquad (3.6)$$

If we choose Φ to be specifically the sum of the weighted squares of the differentiated variable differences, i.e.,

$$\Phi = [W(q(T) - q_0)]^t [q(T) - q_0] \quad , \qquad (3.7)$$

where W is a constant weighting coefficient vector, then the integral form for Φ can be written as follows:

$$\Phi = \int_0^T \phi\, dt = \int_0^T [(q(t) - q_0)^t \frac{dq}{dt}]\, dt \quad , \qquad (3.8)$$

where t_0 is taken to be zero for simplicity of notation.

Since the variables u and q must satisfy the network equations (3.5), by using Lagrangian multipliers [8] λ_1 and λ_2, equation (3.8) for Φ can be rewritten as

$$\Phi = \int_0^T [\phi\ (u, q, q_0, t) + \lambda_1^t\ h\ (u, q, t) + (\lambda_2^t\ M\ u + \lambda_2^t\ A\ \frac{d\ q}{d\ t}]\ d\ t - \int_0^T (\lambda_1^t, \lambda_2^t)\ E\ (t)\ dt \qquad (3.9)$$

where λ_1 and λ_2 are assumed to be independent of the initial conditions q_0.

The gradient G will then be

$$G = \frac{d\ \phi}{d\ q_0} = \frac{d}{d\ q_0} \int_0^T [\phi + \lambda_1^t\ h + \lambda_2^t\ M\ u + \lambda_2^t\ A \frac{d\ q}{d\ t}]\ dt$$

$$= G\ T_1 + G\ T_2 + G\ T_3 + G\ T_4 . \qquad (3.10)$$

$$G\ T_1 = \frac{d}{d\ q_0} \int_0^T \phi\ d\ t = \int_0^T [\frac{\partial\ \phi}{\partial\ u}\ \frac{\partial\ u}{\partial\ q_0} + \frac{\partial\ \phi}{\partial\ q}\ \frac{\partial\ q}{\partial\ q_0} + \frac{\partial\ \phi}{\partial\ q_0}]\ dt$$

If we use for ϕ the form given by (3.8) then

$$G\ T_1 = \int_0^T [W \frac{d\ q}{d\ t}]^t\ d\ t + \int_0^T [W\ (q\ (t) - q_0)]^t \cdot \frac{d}{d\ t}(\frac{\partial\ q}{\partial\ q_0})\ dt\ ,$$

which, after some straightforward manipulation, can be simplified to:

$$G\ T_1 = [W\ (q\ (T) - q_0)]^t \cdot \frac{\partial\ q\ (T)}{\partial\ q_0} - [W\ (q\ (T) - q_0)]^t \qquad (3.11a)$$

The second term $G\ T_2$ can be expanded as follows

$$G T_2 = \frac{d}{d q_0} \int_0^T \lambda_1^t h \, d t = \int_0^T \lambda_1^t \left[\frac{\partial h}{\partial u} \frac{\partial u}{\partial q_0} + \frac{\partial h}{\partial q} \frac{\partial q}{\partial q_0} + \frac{\partial h}{\partial q_0} \right] dt .$$

But since h is independent of q_0, therefore, $\partial h / \partial q = 0$, whence $G T_2$ becomes

$$G T_2 = \int_0^T \lambda_1^t \left[\frac{\partial h}{\partial u} \frac{\partial u}{\partial q_0} + \frac{\partial h}{\partial q} \frac{\partial q}{\partial q_0} \right] dt \qquad (3.11b)$$

The third term $G T_3$ is

$$G T_3 = \frac{d}{d q_0} \int_0^T \lambda_2^t M u \, d t = \int_0^T \lambda_2^t M \frac{\partial u}{\partial q_0} dt \qquad (3.11c)$$

In examining the fourth term $G T_4$, which depends on A, we will have to consider the acceptability of different types of nonlinear dependencies. We therefore proceed to reformulate $G T_4$ component-wise, i.e., the inner products will be explicitly shown.

$$G T_4 = \frac{d}{d q_0} \int_0^T \left[\lambda_2^t A \frac{d q}{d t} \right] d t = (I_1, I_2, \ldots, I_k, \ldots I_N) \qquad (3.12)$$

where I_k is the kth entry of $G T_4$, $k = 1, 2, \ldots N$.

$$I_k = \frac{d}{d\, q_{0_k}} \int_0^T [\lambda_2^t \; A \; \frac{d\, q}{d\, t}] \; dt$$

$$= \frac{d}{d\, q_{0_k}} \int_0^T [\sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_{2_i} \; a_{ij}\,(x_r) \; \frac{d\, q_j}{d\, t}] \; dt$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \{ \int_0^T [\lambda_{2_i} \; a_{ij}\,(x_r) \; \frac{d}{d\, t} \; \frac{\partial\, q_j}{\partial\, q_{0_k}}] \; dt$$

$$+ \int_0^T [\lambda_{2_i} \; \frac{\partial\, a_{ij}\,(x_r)}{\partial\, x_r} \; \frac{\partial\, x_r}{\partial\, q_{0_k}} \; \frac{\partial\, q_j}{\partial\, t}] \; dt\}$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} (I_{k1} + I_{k2}) \quad , \qquad (3.13)$$

where x_r may be an algebraic (u_r) or a differential (q_r) variable.

$$I_{k1} = \int_0^T [\lambda_{2_i} \; a_{ij}\,(x_r) \; \frac{d}{d\, t} \; \frac{\partial\, q_j}{\partial\, q_{0_k}}] \; dt$$

$$= \lambda_{2_i} \; a_{ij}\,(x_r) \; \frac{\partial\, q_j}{\partial\, q_{0_k}} \Big|_0^T$$

$$- \int_0^T \frac{d\, \lambda_{2_i}}{d\, t} \; a_{ij}\,(x_r) \; \frac{\partial\, q_j}{\partial\, q_{0_k}} \; dt$$

$$- \int_0^T \lambda_{2_i} \; \frac{\partial\, a_{ij}\,(x_r)}{\partial\, x_r} \; \frac{\partial\, x_r}{\partial\, t} \; \frac{\partial\, q_j}{\partial\, q_{0_k}} \; dt$$

substituting for I_{k1} in equation (3.13)

$$I_k = \sum_{i=1}^{N} \sum_{j=1}^{N} \Big[\lambda_{2_i} a_{ij}(x_r) \frac{\partial q_j}{\partial q_{0_k}} \Big|_0^T - \int_0^T \frac{d\lambda_{2_i}}{dt} a_{ij}(x_r) \frac{dq_j}{dq_{0_k}} dt + \int_0^T \lambda_{2_1} \frac{\partial a_{ij}(x_r)}{\partial x_r} \Big(\frac{\partial x_r}{\partial q_{0_k}} \frac{\partial q_j}{\partial t} - \frac{\partial x_r}{\partial t} \frac{\partial q_j}{\partial q_{0_k}} \Big) dt \Big] \qquad (3.14)$$

In the case of a single dependency, i.e., $x_r \to q_r$ and $r \to j$, the term

$$\frac{\partial x_r}{\partial q_{0_k}} \frac{\partial q_j}{\partial t} - \frac{\partial x_r}{\partial t} \frac{\partial q_j}{\partial q_{0_k}}$$

is identically zero. For a double dependency, where $x_r \neq q_j$, it is shown in Appendix I that this term is identically zero for a class of systems with a unique periodic steady-state solution. Hence, using this fact, and equations (3.12) and (3.14)

$$G T_4 = \lambda_2^t(T) A(T) \frac{\partial q(T)}{\partial q_0} - \lambda_2^t(0) A(0) - \int_0^T \Big[\frac{d\lambda_2^t}{dt} A \frac{\partial q}{\partial q_0} \Big] dt \qquad (3.11d)$$

From equations (3.11a) - (3.11d), the gradient can take the form

$$G = \int_0^T [\lambda_1^t \frac{\partial h}{\partial u} + \lambda_2^t M] \frac{\partial u}{\partial q_0} dt$$

$$+ \int_0^T [\lambda_1^t \frac{\partial h}{\partial q} - \frac{d \lambda_2^t}{d t} A] \frac{\partial q}{\partial q_0} dt$$

$$+ [\lambda_2^t (T) A (T) - (W (q_0 - q (T)))^t] \frac{\partial q (T)}{\partial q_0}$$

$$+ [W (q_0 - q (T))]^t - \lambda_0^t (0) A (0) \qquad (3.15)$$

Since the Lagrangian multipliers can be chosen arbitrarily, they can, in particular, be specified in such a way as to greatly simplify the form of the gradient equation, e.g., take λ_1 and λ_2 such that both the first and second terms in (3.15) are identically zero.

$$\lambda_1^t \frac{\partial h}{\partial u} + \lambda_2^t M = 0$$

and

$$\lambda_1^t \frac{\partial h}{\partial q} - \frac{d \lambda_2^t}{d t} A = 0$$

In matrix form, it can be replaced by

$$\begin{bmatrix} \frac{\partial h}{\partial u} & \frac{\partial h}{\partial q} \\ M & -\frac{d}{d t} A \end{bmatrix}^t \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = 0 \quad . \qquad (3.16)$$

Furthermore, and to avoid the computation of the term $\partial q(T) / \partial q_0$ (which is, in general, a full matrix), the initial conditions of λ_2 can be chosen so that the third term of equation (3.15) is identically zero, i.e.,

$$\lambda_2^t (T) A (T) - [W (q_0 - q (T))]^t = 0 \qquad (3.17)$$

Accordingly, the gradient takes the simplified form

$$G = [W (q_0 - q (T))]^t - \lambda_2^t (0) A (0) \qquad (3.18)$$

Equations (3.16) - (3.18) are the results that we sought. If we compare equations (3.5) and (3.16), the coefficient Jacobian matrix of the latter is just the transpose of the Jacobian of (3.5) with a change in the sign of the time derivative term. System (3.16) is called the adjoint variational system of the original one described by (3.5). Relation (3.17) defines the initial conditions of that adjoint system for computing G. Note that these conditions are given at $t = T$, which means that to satisfy equation (3.18), the integration of the adjoint system has to be carried out over one full period from $t = T$ to $t = 0$, i.e., in the backward direction. It should also be noted that, due to this reversal of the direction of integration, the sign of the differentiated terms in (3.16) will be restored to the original sign. Let us define a new time base τ, for the adjoint system where $\tau = T - t$, so that $d / dt = - d / d \tau$.

In that case (3.16) takes the form

$$-\begin{bmatrix} \frac{\partial h}{\partial u} & \frac{\partial h}{\partial q} \\ M & A\frac{d}{d\tau} \end{bmatrix}^t \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad (3.19)$$

In summary, (3.19) defines the adjoint system which, when integrated over one full period, starting with initial conditions at $\tau = 0$ (i.e., $t = T$) given by equation (3.17), has a final solution at $\tau = T$ (i.e., $t = 0$) in which λ_2 defines the second term of the gradient equation (3.18).

3.3 Choice of the Objective Function Φ

If the objective function Φ is to be a meaningful measure of the circuit's deviation from the periodic steady-state, then the components of Φ, corresponding to the contributions of the individual differentiated variables, should, in general, be of the same order of magnitude. However, the representation of the branch variables (i, v, q, f) in the Kirchhoff's laws equations in SCAMPER depends on the type of dependency and on whether the branch is nonlinear. Thus, the current in a q-dependency (capacitive) branch is replaced by dq / dt if the element is linear, and by c dv / dt if it is nonlinear. Because the units of the differentiated variables differ, their values

may also differ by many orders of magnitude. Therefore, it is necessary to weight the contribution of each storage element to the performance function Φ.

One particularly convenient way of achieving this weighting is to convert all the branch variables to voltages when computing Φ. Thus, for example, the variable associated with a linear capacitive branch, namely q_i, would be converted to a voltage v_i by defining $v_i \equiv q_i / C_i$. An objective function which is based on this approach and which has the same form as equation (3.7) is

$$\Phi = \sum_{i=1}^{N} \phi_i = \sum_{i=1}^{N} \omega_i \cdot \frac{1}{2} (v(T) - v_0)_i^2 \tag{3.20}$$

where N is the system order, and ω_i is an additional weighting factor for reactive element i.

An alternative form of Φ, which also tends to reduce the spread in the contributions of the storage elements to the objective function, is the sum over the stored energy, i.e.,

$$\Phi = \sum_{i=1}^{N} \phi_i = \sum_{\substack{\text{over} \\ \text{all C}}} \omega_i \cdot \frac{1}{2} (v(T) - v(0))_i \, (q(T) - q(0))_i$$

$$+ \sum_{\substack{\text{over} \\ \text{all L}}} \omega_j \cdot \frac{1}{2} (i(T) - i(0))_j \, (f(T) - f(0))_j \,, \tag{3.21}$$

where f is the inductor flux.

It can, in fact, be shown that (3.21) and (3.20) can be made equivalent by a suitable choice of the ω's. Nevertheless, equation (3.20) is more convenient to use and was selected for the work reported here.

3.4 Detailed Computations of the Gradient and the Initial Conditions of the Adjoint System

The initial conditions $\lambda(T)$ of the adjoint system are defined by equation (3.17), while the gradient G can be computed through equation (3.18). Both equations are given in matrix form, and appear to be difficult to handle. In order to see how easy it is, in fact, to compute $\lambda_2(T)$ and G, it is helpful to review the actual representation of the system variables and the order in which the equations are set in the dc and transient analysis program SCAMPER [19], [20] which was used for this work. (For more details see Part A of Chapter V).

Reactive Variable Representation:

The current through a q-dependency (capacitive) element in Kirchhoff's current law equations is replaced by:

(i) $\frac{dq}{dt}$ if the element is linear (self or mutually dependent), i.e., the differentiated variable is charge q.

(ii) $c \frac{d v}{d t}$ if the branch is nonlinear, where v is often a voltage, i.e., the differentiated variable is a voltage.

The voltage across an f-dependency (inductive) element in Kirchhoff's voltage law is replaced by:

(i) $\frac{d f}{d t}$ if the element is linear.

(ii) $L \frac{d i}{d t}$ if the element is nonlinear, where i is often a current.

The Order of the System Variables and Equations

Consider the number of elements is NE, and the number of nodes NN. In the tableau representation there are (2NE + NN - 1) equations and variables. The equations are ordered in the following way: first, the branch Kirchhoff's voltage law equations KVL (1 to NE) ; second, the nodal Kirchhoff's current law equations KCL (NE + 1 to NE + NN - 1) ; third, the branch constitutive relations B.C. (NE + NN to 2NE + NN - 1). The variables appear in the order of : first, branch current or charge (1 to NE) ; second, branch voltage or flux (NE + 1 to 2 NE); third, node voltages (2NE + 1 to 2 NE + NN - 1) .

Now, a close examination of equations (3.17) and (3.18), in the light of the above comments, reveals a very simple procedure for computing the different components of the gradient G of the performance

function defined by equation (3.20) as well as the initial conditions λ (T) . The following relations for λ (T) and G can be proved to be equivalent to those of equations (3.17) and (3.18).

For a linear capacitor C_ℓ :

$$\phi_{c_\ell} = \frac{1}{2} \cdot \omega_{c_\ell} \cdot \frac{1}{c_\ell^2} \left(q_0 - q^{(T)}\right)^2_{c_\ell} \tag{3.21a}$$

The initial conditions λ (T) in the solution of the adjoint system will be

$$\lambda_{NE+N1}(T) - \lambda_{NE+N2}(T) = \omega_{c_\ell} \frac{1}{c_\ell^2} \left(q_0 - q(T)\right)_{c_\ell} \tag{3.22a}$$

where N1 and N2 are the numbers of nodes to which the branch is connected.

The corresponding gradient G_{c_ℓ}

$$G_{c_\ell} = C_\ell \frac{d\phi}{d q_{0_{c_\ell}}} = \omega_{c_\ell} \frac{1}{C_\ell} \left(q_0 - q(T)\right)_{c_\ell} - C_\ell \left(\lambda_{NE+N1}(0) - \lambda_{NE+N2}(0)\right) \tag{3.23a}$$

while the argument passed to the optimization routine x_{c_ℓ}

$$x_{c_\ell} = \frac{1}{C_\ell} q_{0_{c_\ell}} \tag{3.24a}$$

For a nonlinear capacitor $C_n(\cdot)$:

$$\phi_{c_n} = \frac{1}{2}\,\omega_{c_n}\,(v_0 - v\,(T))^2_{c_n}\,, \tag{3.21b}$$

$$\lambda_{NE+N1}\,(T) = \lambda_{NE+N2}\,(T) = \omega_{c_n}\,\frac{1}{C_n\,(T)}\,(v_0 - v\,(T))_{c_n}\,, \tag{3.22b}$$

where $C_n\,(T)$ is the value of C_n at $t = T + t_0$.

$$G_{c_n} = \frac{d\,\phi}{d\,v_{0_{c_n}}} = \omega_{c_n}\,(v_0 - v\,(T))_{c_n} - C_n\,(0)\,(\lambda_{NE+N1}\,(0) - \lambda_{NE+N2}\,(0)) \tag{3.23b}$$

where $C_n\,(0)$ is the value of C_n at $t = t_0$,

and

$$x_{c_n} = v_{0_{c_n}} \tag{3.24b}$$

For a linear inductor L_ℓ :

$$\phi_{l_\ell} = \frac{1}{2} \cdot \omega_{l_\ell} \cdot \left(\frac{R}{L_\ell}\right)^2 (f_0 - f\,(T))^2_{l_\ell}\,, \tag{3.21c}$$

where R is a weighting resistance.

The initial condition $\lambda\,(T)$ in the solution of the adjoint system

$$\lambda_i\,(T) = \omega_{l_\ell}\,\left(\frac{R}{L_\ell}\right)^2 (f_0 - f\,(T))_{l_\ell}\,, \tag{3.22c}$$

where i is the branch number $(1 \le i \le NE)$.

The corresponding gradient G_{l_ℓ}

$$G_{l_\ell} = \frac{L_\ell}{R} \frac{d\phi}{d f_{0_{l_\ell}}} = \omega_{l_\ell} \left(\frac{R}{L_\ell}\right) (f_0 - f(T))_{l_\ell} - \frac{L_\ell}{R} \lambda_i(0) \tag{3.23c}$$

while the argument passed to the optimization routine X_{l_ℓ}

$$X_{l_\ell} = \frac{R}{L_\ell} f_{0_{l_\ell}} \tag{3.24c}$$

For a nonlinear inductor $L_n(\cdot)$:

$$\phi_{l_n} = \frac{1}{2} \omega_{l_n} R^2 (i_0 - i(T))^2_{l_n} , \tag{3.21d}$$

$$\lambda_i(T) = \omega_{l_n} \frac{R^2}{L_n(T)} (i_0 - i(T))_{l_n} , \tag{3.22d}$$

where $L_n(T)$ is the value of L_n at $t = T + t_0$

$$G_{l_n} = \frac{1}{R} \frac{d\phi}{d i_{0_{l_n}}} = \omega_{l_n} R (i_0 - i(T))_{l_n} - \frac{L_n(0)}{R} \lambda_i(0) \tag{3.23d}$$

where $L_n(0)$ is the value of L_n at $t = t_0$,

and

$$X_{l_n} = R\, i_{0_{l_n}} \tag{3.24d}$$

3.5 A Computational Technique

The proceeding formulation of the gradient method has been implemented in the dc and transient analysis program SCAMPER by means of the algorithm outlined below.

Algorithm 3.1

1. Initialization Step.

   (i) Define constants (minimum limit of Φ (Φ_{min}) and G (G_{min}) and maximum number of iterations NIT (NIT$_{max}$)) .

   (ii) Integrate the original system of equations (3.5) until a predefined time t_0, yielding a suitable set of initial conditions for the steady-state analysis.

   (iii) At $t = t_0$ save the initial conditions of the reactive variables q, the solution vector (u, q) and the values of the nonlinear reactive elements (for computing matrix A (0)).

   (iv) NIT = 0 .

2. Forward Integration.

   (i) NIT = NIT + 1 .

(ii) Integrate the original system (3.5) over one full period T from $t = t_0$. Save the necessary nonlinear entries of the Jacobian J as well as the step size at a sequence of time points.

(iii) At $t = t_0 + T$ compute $q(T)$ and the values of nonlinear reactive elements (for computing $A(T)$).

(iv) Compute the performance function Φ defined by equation (3.20), using equations (3.21a, b, c, and d).

(v) If $\Phi \leq \Phi_{min}$, a steady-state solution has been found. STOP.

(vi) IF $NIT > NIT_{max}$. STOP.

(vii) Compute the initial conditions $\lambda_2(T)$ for the adjoint system defined by equation (3.17) using equations (3.22a, b, c, and d). Also compute the first term of the gradient defined by equation (3.18) using equations (2.23a, b, c, and d).

3. Backward Integration.

(i) With the initial conditions $\lambda(T)$ computed in

step (2. vii), integrate the adjoint system of equations (3.19) over one full period using the stored values of the nonlinear entries of the Jacobian as well as the corresponding step sizes.

(ii) At the end of the period $\tau = T$ (i.e., $t = t_0$) use the solution of the adjoint system λ (0) and A (0) for completing the computation of the gradient (second term) using equations (3.23a, b, c, and d).

4. Optimization.

(i) Scale the argument vector X according to equations (3.24a, b, c, and d).

(ii) Call the optimization routine.

(iii) Reset the initial conditions of the original system (u, q) to the values saved from step (1. iii).

(iv) Update the reactive element variables q with the values q_0 returned by the optimization routine.

(v) Go to step 2.1.

Remarks

(1) The optimization routine returns a new set of initial conditions q_0. Due to the nonlinearity of the system, and the change in q_0 (which may be large relative to the previous values of q_0) the system may fail completely to satisfy the nonlinear error criteria at the starting point of integrating the original system. A way to overcome this problem is to start the integration with values of q_0 which satisfy the system equations at t_0. This is the reason for step (1. iii). In fact, the only values which it is necessary to save for this step are the components of (u, q) corresponding to the nonlinear branches.

(2) Steps (2. iv) and (2. vii) can be computed simultaneously.

(3) Some optimization techniques consider a minimum to have been found (which in our application means a steady-state solution is found) if the gradient

vector $|G|$ (or its norm $||G||_2^2$) is less than a certain limit. Hence another termination criterion can be added in this case.

(4) For more details about resetting the initial conditions as well as the integration process, see Part B of Chapter V.

CHAPTER IV

NUMERICAL CONSIDERATIONS AND RESULTS

4.1 Introduction

The periodic steady-state analysis algorithms 3.1 was implemented in the dc and transient analysis program SCAMPER. For the optimization step, two different optimization routines were made available to the user. These routines are Fletcher's [22] variable metric routine (VM), which requires full matrix storage, and an automatic restart, conjugate gradient routine (ARCG) by Powell [23] which requires less storage.

The convergence of the algorithm as well as the execution time are greatly influenced by the scaling of the different arguments. In Section 4.2 the scaling features are discussed briefly and some suggestions are made, based on our limited experience with the program, regarding the selection of a proper scheme for scaling the arguments.

Three examples are presented in Section 4.3. These were solved using both the VM and the ARCG optimization routines, however, since the VM routine gave better convergence, most of the detailed results are given for that routine and only some final results are shown for ARCG for comparison. An important result that comes out of these examples is that the time required for integrating the adjoint system of equations can be controlled and is usually less than half the time required for integrating the original system of equations describing the network.

Although these same examples were also analyzed by Nakhla [11] using a similar gradient method and the same VM routine, a detailed comparison of results is, in general, not meaningful because of the different strategies used for the transient analysis. In our case, the transient analysis starts after the dc solution is found, while in Nakhla's program the transient analysis starts from zero initial conditions.

4.2 Scaling

The results obtained for several problems demonstrate clearly that the optimization step of the algorithm is the most critical one. The available optimization routines, VM and ARCG, may require a large number of iterations to reach the region of quadratic convergence, depending on the starting point. The convergence rate as well as the execution time can be greatly affected by the choice of the scaling parameters, so that it is important to understand their respective functions in order to use them properly. The four kinds of scaling which are available are discussed below and some guide-lines are given for their proper use.

I - The equivalent resistance, RLS, for the inductive elements:

The initial conditions, q_0, corresponding to the different types (linear and nonlinear) of inductive elements are made in-

ternally to be equivalent to current sources I_{0_ℓ}, while those corresponding to the capacitive elements are made equivalent to voltage sources V_{0_c}. The equivalent resistance RLS is introduced to scale the inductive variables by converting them to voltages $V_{0_\ell} = RLS \cdot I_{0_\ell}$. A choice of the value of RLS other than unity may be necessary in cases where there are order of magnitude differences between the V's and I's of the C's and L's, respectively. From the computational point of view, the equivalent resistance RLS scales the objective functions of the inductive elements ϕ_ℓ by $(RLS)^2$ and the variables passed to the optimization routine, X_ℓ, by RLS, while the gradient will be scaled indirectly by RLS and $(1 / RLS)$. (See equations 3.23c and d).

II - Weighting of the objective function ϕ_i :

The weighting factor ω_i of the objective function ϕ_i given by equation (3.20) is specified via the parameter $FCTR_i = \omega_i$. This kind of weighting is useful for controlling the influence of each storage element's deviation from the periodic steady-state on the conversion process. For example, the $FCTR_k$ of ϕ_k corresponding to element k associated with a longer time constant, and yet probably having a relatively small initial difference between q_0 and $q(T)$, may be increased to advantage w. r. t. the other $FCTR_i$.

This kind of weighting affects both the objective function ϕ_i, and therefore Φ, and the gradient vector G.

III - Scaling the optimization variables X :

$$SCLX(i) = SCLX_i \quad , \quad i = L, 2, \ldots N .$$

In the steady-state analysis of circuits, cases are often encountered where the values of the variables differ by orders of magnitude. This may result in poor convergence due to the search for a minimum over an elongated contour of the performance function Φ with respect to these variables. Convergence may be improved by scaling the variables to be of the same order of magnitude. Care must be taken in using this type of scaling as the relative magnitudes of the variables may change drastically as the search for a minimum Φ proceeds. Four options are available in using SCLX.

(i) Individual scaling of each element. In this case M values have to be provided. If M < N, the first M variables are scaled in the given order of their corresponding branches, by the specified M values, while the remaining N - M variables will be scaled by unity.

(ii) Common scaling for all C's and L's, respectively. In this case two values, $SCLX_c$ and $SCLX_\ell$ must be provided where all the capacitive variables will be scaled by the first value, $SCLX_c$, and all the inductive variables by the

second, $SCLX_\ell$. This particular form of scaling is simple to use and yet quite important. If, for example, inductor currents are of the order of milliamps while the capacitor voltages are of the order of volts, the ratio $SCLX_\ell$ / $SCLX_c$ could be, say, 1000.

(iii) All the variables to be scaled in such a way as to cause their scaled values to be equal at the first steady-state iteration. In this case a parameter, XREF, must be supplied whereupon the variables will be scaled internally to be equal to XREF for the first iteration of the steady-state analysis.

(iv) No scaling at all.

IV - Scaling the initial conditions of the adjoint system SCLD :

If this scaling is required all the initial conditions of the adjoint system, λ_2 (T), will be scaled equally in such a way as to cause the initial condition with the largest equivalent voltage to be equal to SCLD at the beginning of each integration of the adjoint system. The reason for introducing this type of scaling is that the gradient may not have to be computed as accurately during the

first few iterations as within the quadratic region of convergence. Hence, if the parameter SCLD is chosen to be small enough relative to the largest equivalent voltage of λ_2 (T) at the first iteration and large enough relative to that same component of λ_2 (T) at the last iteration (which is normally within the specified error of computation), the average time for integrating the adjoint system may be reduced. Furthermore, the higher accuracy during the last few iterations may improve the convergence.

On the basis of our admittedly limited experience with the program, the following guidelines are proposed for using the steady-state analysis effectively.

(i) Perform a transient analysis over few periods. Examine the reactive (differentiated) variables as well as those corresponding to sharp nonlinearities, such as exponentials, over the last period and choose the starting point t_0 in such a way that (1) regions where the sign of the time derivatives is changing rapidly are avoided, (2) it does not correspond to the region of conduction of junctions shunted by reactive elements (if any), and (3) as many variables as possible are near their largest absolute values.

(ii) Choose the equivalent resistance RLS, corresponding to the inductive elements (if any), in such a way as to cause the average equivalent voltage of the inductive elements to be comparable to the average voltage of the capacitive elements.

(iii) Weight each objective function ϕ_i according to its expected influence on the transient response of the circuit (taking into account the effect of RLS).

(iv) Scale the arguments passed to the optimization routine in such a way as to make them all approximately equal (taking into account the effect of RLS).

4.3 Numerical Results

Three examples are presented to illustrate the effectiveness of the proposed steady-state algorithm 3.1, and to demonstrate the importance of using scaling. Besides the scaling parameters introduced in the preceding section, the following notations will be used in these examples

TF, TB — the time required for one full period integration of the original and the adjoint system, respectively ;

NP — the equivalent total number of forward integrations required for completing the steady-state analysis, i.e., NP = TT / TF , where TT is the total time to convergence.

Example 1

The importance of using a steady-state analysis algorithm rather than using continuous transient analysis ("brute force" method) is demonstrated by this simple example which was also analysed by Nakhla [11]. The clipper circuit shown in Figure 4.1 was analysed twice, once using the gradient method and once by the "brute force" technique. In fact, with the dc source EC = 5 volts, the diode never conducts so that the circuit is effectively a linear one. The circuit has two time constants T_1 and T_2 which, with the diode off, can be simply shown to be given by : $T_1 \approx C_1 (R_1 R_2 / (R_1 + R_2))$ and $T_2 \approx C_2 (R_1 + R_2)$. The ratio T_0 / T_1 is approximately 50, while the ratio of the longer time constant T_2 to the period of the input signal is approximately 100 . Hence, using the brute force method, the steady-state condition is not expected to be reached before a few hundred full periods of integration. This is confirmed by the results

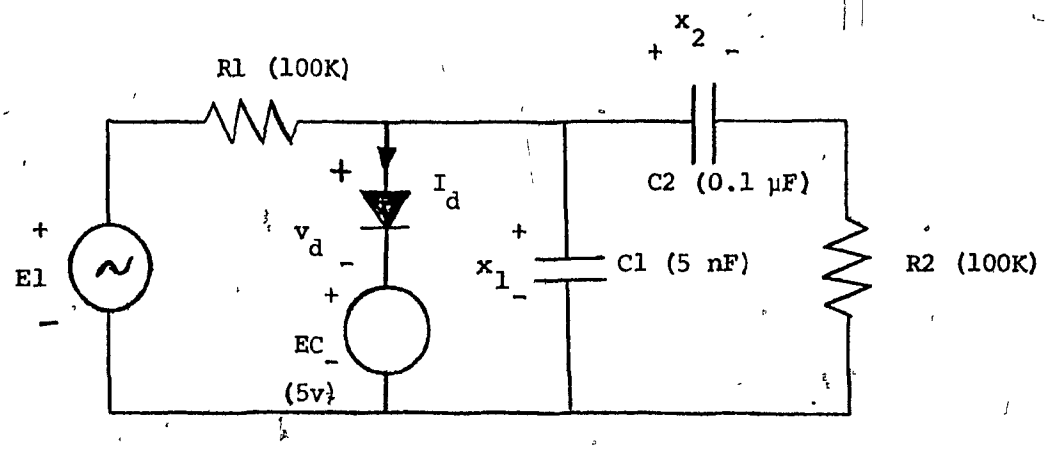


Figure 4.1. Clipper circuit.

$E1 = 20 \sin (\pi \times 10^4 t)$

$I_d = 10^{-6} (e^{40 v_d} - 1)$

shown in the first two columns of Table 4.1. Note that with the steady-state algorithm, Φ is reduced to less than 10^{-9} in two iterations (equivalent to approximately 2.64 periods of forward integration) whereas 49 periods by the brute force method reduces Φ to only 0.234×10^{-7}.

Example 2

The circuit shown in Figure 4.2 has been considered by both Nakhla [11] and by Trick et al [1] who showed that the system was still far from steady-state after 75 full-period integrations. With the gradient algorithm we obtained convergence to a Φ of 10^{-12} in 16 iterations with $t_0 = 3.5$ T. The only scaling used was that of the adjoint system initial conditions, SCLD. With SCLD equal to 0.01 the convergence within the quadratic region was improved and the execution time was reduced by about 10%. In Figure 4.3 the ratio (TB / TF) for both the scaled and the unscaled cases are plotted against the iteration number. With scaling the average (TB / TF) was 46% while without scaling it was 51% .

Example 3

The importance of applying the different types of scaling in using the gradient method is shown in the analysis of the class C amplifier of Figure 4.4. Starting the analysis after 4.9 periods, and with scaling applied as shown in Table 4.3, the steady-state

TABLE 4.1

RESULTS OF EXAMPLE 1

Brute Force Method		Gradient Method			
Period	Φ	Iteration	Φ	x_1	x_2
0	0.236	0	0.236	0.1	0.1
1	0.467×10^{-1}	1	0.992×10^{-8}	-1.1513	0.10156
2	0.925×10^{-2}	2	0.477×10^{-9}	-1.1515	0.10156
10	1.00×10^{-7}				
20	0.415×10^{-7}				
30	0.34×10^{-7}				
40	0.28×10^{-7}				
49	0.234×10^{-7}				

For the gradient method :-

SCLX 0,5 10.

$(x_1 , x_2)_{final}$: (- 1.1515, 0.10156)

Execution time = 5.37 sec.

$(TB / TF)_{average}$ = 32 % .

NP = 2.64 .

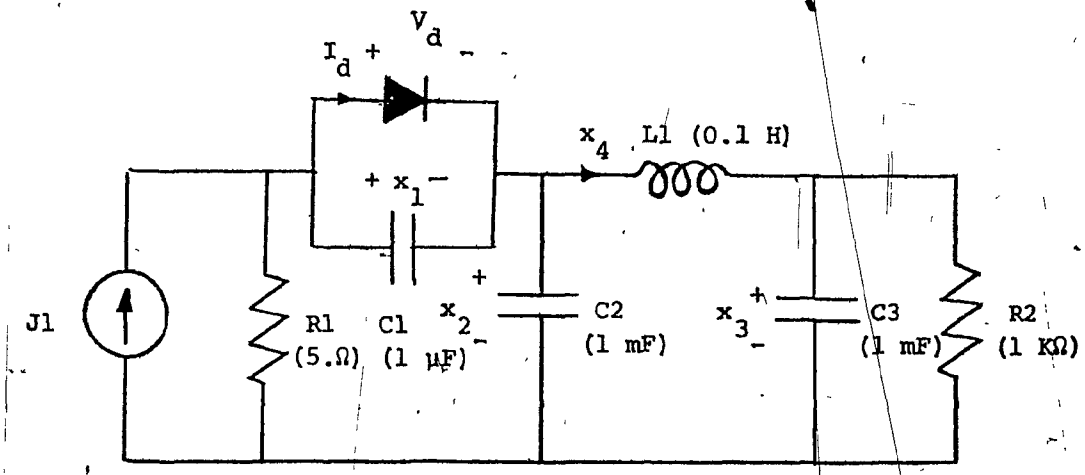


Figure 4.2. Power source.

$$J1 = 2 \sin (2\pi \times 60\ t)$$

$$I_d = 10^{-6} (e^{40\ v_d} - 1)$$

TABLE 4.2

RESULTS OF EXAMPLE 2

Scaling :

SCLD = 0.01.

Iteration	Φ	Iteration	Φ
0	4.26	9	0.995×10^{-3}
1	0.227	10	0.490×10^{-4}
2	0.226	11	0.153×10^{-5}
3	0.222	12	0.173×10^{-6}
4	0.212	13	0.122×10^{-7}
5	0.118	14	0.479×10^{-9}
6	0.111	15	0.489×10^{-11}
7	0.254×10^{-1}	16	0.232×10^{-12}
8	0.892×10^{-2}		

$[x_1, x_2, x_3, x_4]$ = [-9.07535, 9,05648, 9.10251, 9.029×10^{-3}]

Execution time = 45.7 sec.

$(TB / TF)_{average}$ = 46 % .

NP = 27 .

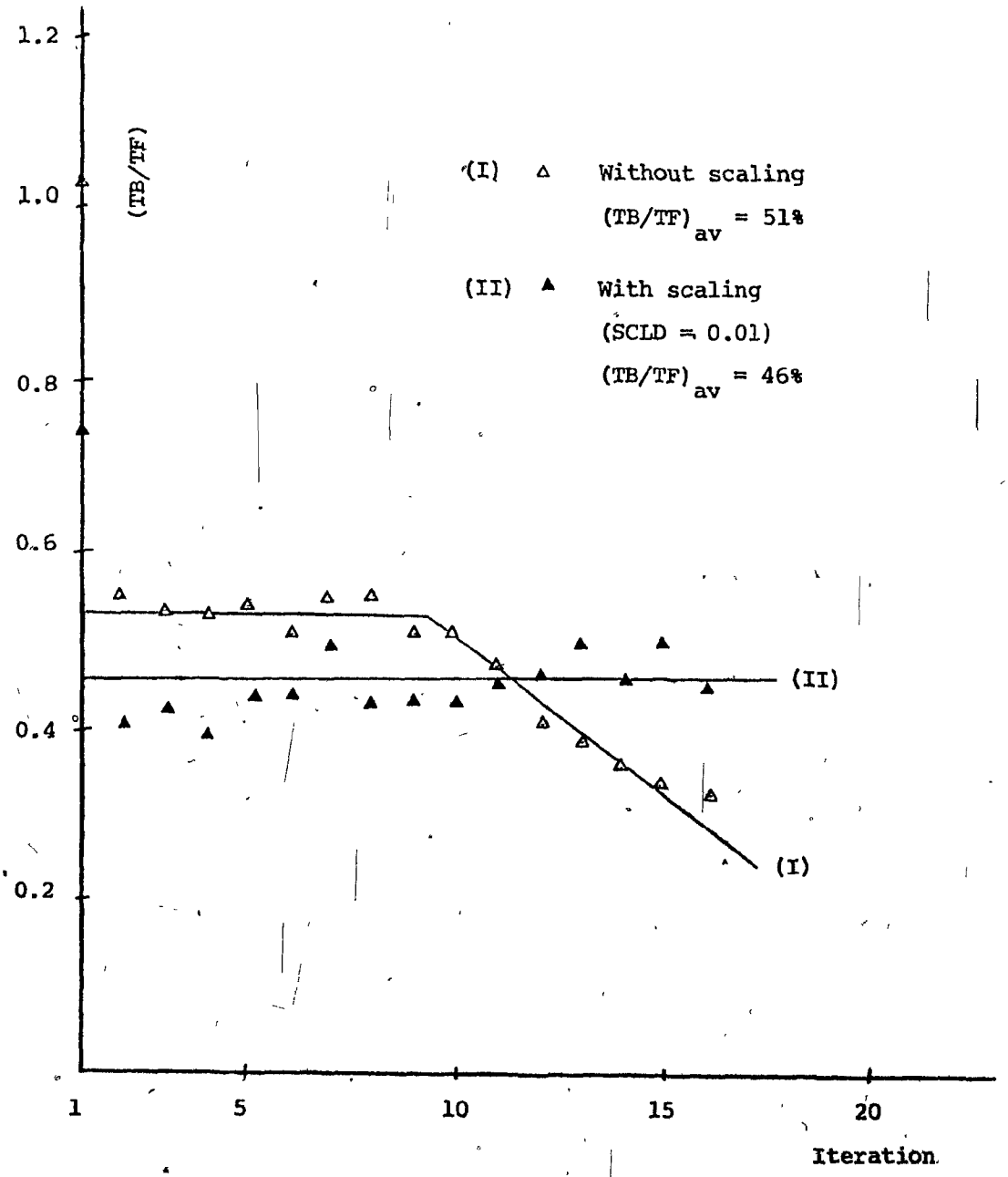
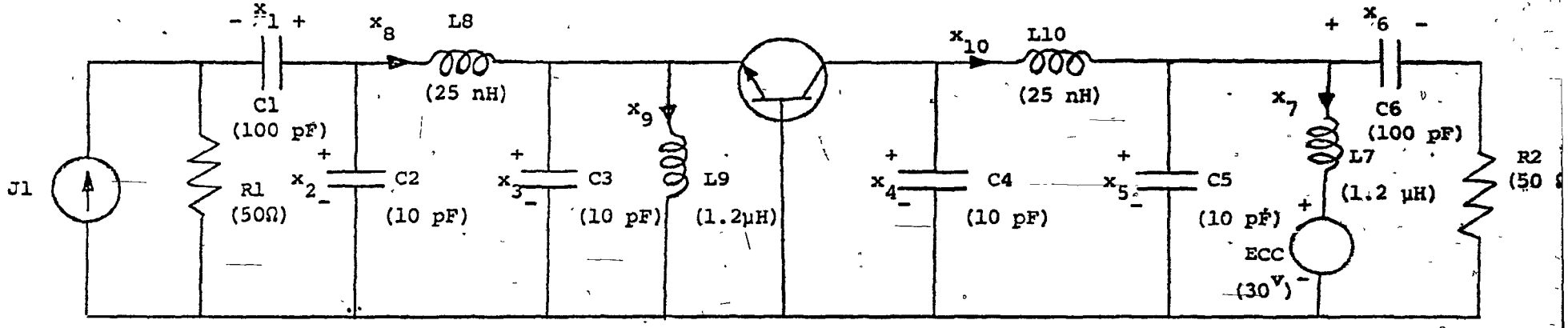


Figure 4.3. The relation between (TB/TF) vs iteration number for power supply example 2.

conditions were found after 23 iterations accurate to six digits with $\Phi < 10^{-12}$. The two sets of values shown for x_i in Table 4.3 correspond to the final steady-state solution at $t = t_0 = 4.9T$ and also at $t = 5.T$. The latter are given to permit comparison with the results of Trick et al [1] and Nakhla et al [10], [11].

When this problem was solved with the same RLS, FCTR and SCLX but without SCLD, the method converged to $\Phi = 0.819 \times 10^{-11}$ after 24 iterations with 118.4 sec execution time. This corresponds to an increase of about 15 % in the execution time due to the omission of SCLD. Figure 4.5 shows (TB / TF) as a function of iteration number without SCLD and with SCLD equal to 0.01. For the unscaled case, the increase of the execution time during the first few iterations was not balanced by the reduction during the last iterations resulting in an increase of the average (TB / TF) over that for the case where SCLD was used properly. In the absence of any kind of scaling, the method was not able to converge within the same number of iterations. After 28 iterations, and 150 secs. of exectuion time the objective function Φ was reduced to only 0.45×10^{-4}. It is worth noting that when the same problem was solved, for comparison, using the conjugate gradient optimization routine, the algorithm was not able to converge and, after 28 iterations, the objective function reduced only to 0.19×10^{-2} which is very far from the steady-state condition.



(a) Class C - Amplifier circuit

$J1 = 0.1 \sin (2\pi \times 10^8 t)$

(b) Transistor Model

$\alpha_f = 0.99$

$\alpha_r = 0.5$

$I_f = 10^{-6} (e^{38.9 v_{be}} - 1)$

$I_r = 10^{-6} (e^{38.9 v_{bc}} - 1)$

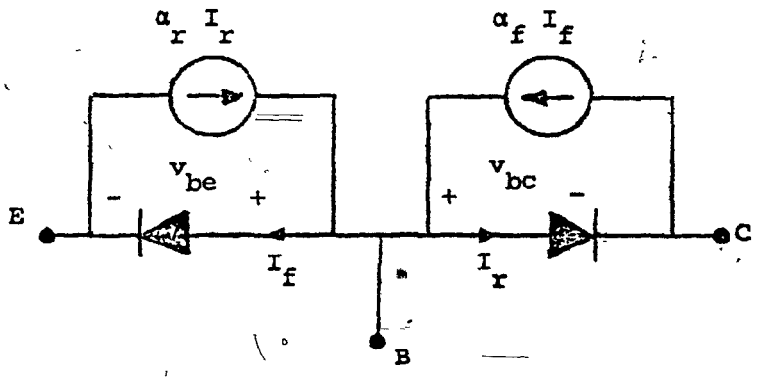


Figure 4.4.

TABLE 4.3

RESULTS OF EXAMPLE 3

Scaling :

RLS = 50 .

$FCTR_i$, i = 1, 2, .., 10 : 1., 1., 1., 0.1, 0.1, 0.1, 1., 1., 1., 1.

$SCLX_i$, i = 1, 2, ... 10 : 0.5, 0.5, 1., 0.1, 0.1, 0.1, 1., 1., 1., 1.

SCLD : 0.01

Iteration	Φ	Iteration	Φ	Iteration	Φ
0	0.138	8	0.307×10^{-1}	16	0.345×10^{-4}
1	0.329	9	0.284×10^{-1}	17	0.114×10^{-4}
2	0.973×10^{-1}	10	0.272×10^{-1}	18	0.496×10^{-5}
3	0.718×10^{-1}	11	0.262×10^{-1}	19	0.546×10^{-6}
4	0.532×10^{-1}	12	0.180×10^{-1}	20	0.331×10^{-7}
5	0.416×10^{-1}	13	0.133×10^{-1}	21	0.691×10^{-9}
6	0.338×10^{-1}	14	0.463×10^{-2}	22	0.105×10^{-10}
7	0.315×10^{-1}	15	0.637×10^{-3}	23	0.234×10^{-12}

$(x_i, i = 1, 2, \ldots, 10)_{4.9}$: 1.16458 , 1.04044 , -0.304503 , 26.5495 ,

5.0 1.4530 1.41695 -0.288972 28.3779

25.5844 , 29.4632 , -0.076572 , -0.062961 , -0.076349 , -0.149948

26.8711 28.8556 -0.079830 0.000141 -0.076101 -0.098802

Execution time = 104.73 sec.

$(TB / TF)_{average}$ = 48 %

NP = 39.5 .

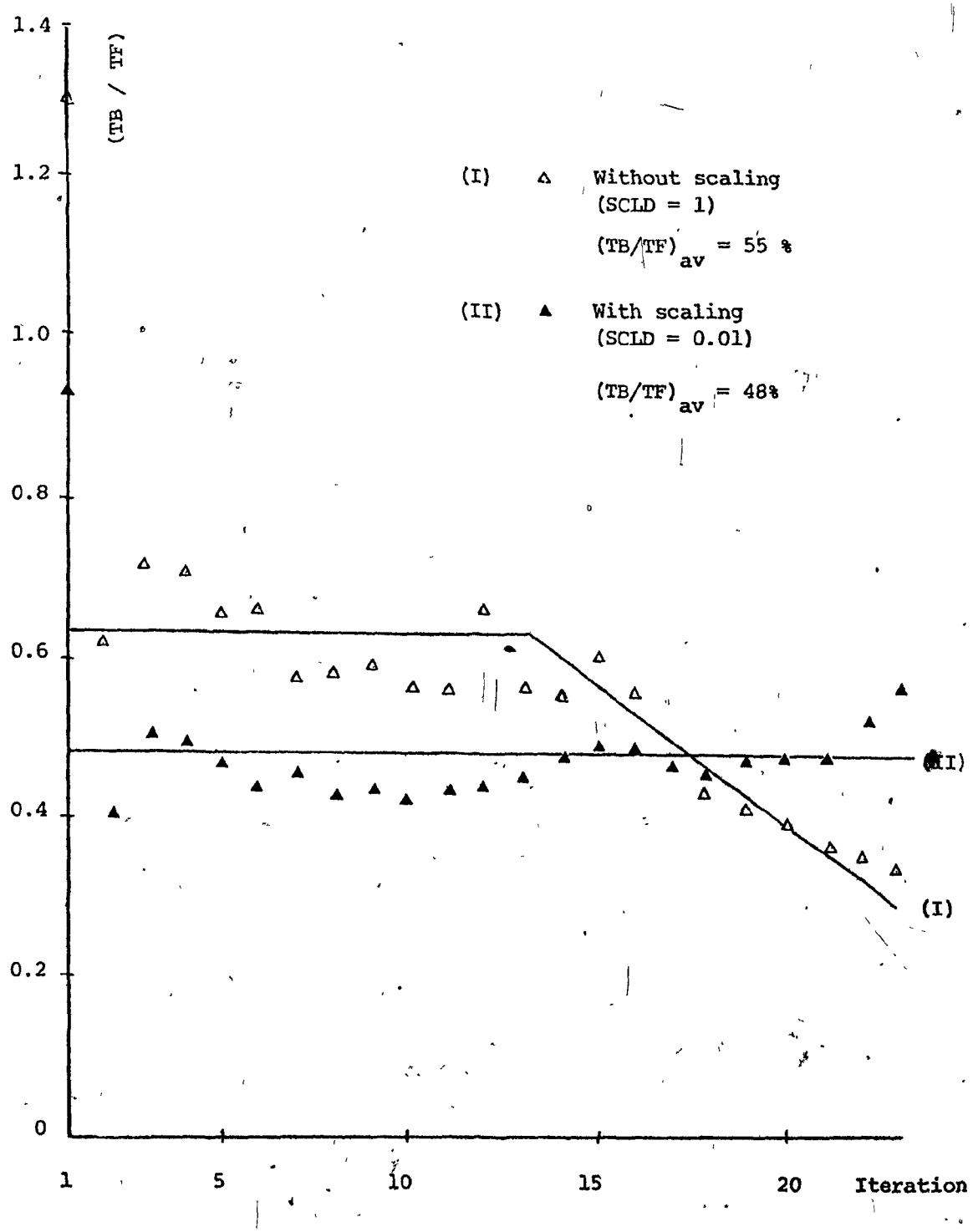


Figure 4.5. The relation between (TB/TF) vs the iteration number for Class C - amplifier, Example 3.

CHAPTER V. PROGRAMMING

PART A

DC AND TRANSIENT ANALYSIS PROGRAM SCAMPER

The transient analysis program SCAMPER, to which the periodic steady-state analysis has been added was written by Millar and Blostein [19], [20]. Its main features are accuracy, high speed and a capability of handling a wide range of branch types. Circuits of about 350 branches (which is roughly equivalent to a circuit of 20 transistors) can be analyzed.

The analysis starts by generating the dc solution. All the independent sources are set to their values at zero time and all the time derivatives are removed from the network equations, (which is equivalent to short circuiting all the inductors and open circuiting all the capacitors). Then, if required, a transient analysis is carried out over a specified time interval.

Techniques such as high order implicit integration with variable step size, the use of sparse matrices and the generation of executable machine code account for SCAMPER's high speed and accuracy. The program is written mainly in FORTRAN IV.

5.1 Acceptable Element Types

There are two general types of branches which are accepted by SCAMPER.

(i) Sources:

For current source

$$J = J(t) \tag{5.1a}$$

For voltage source

$$E = E(t) \tag{5.1b}$$

where the time dependent function can be any of the following functions. (1) Constant (dc source). (2) Sine wave (magnitude A_0, starting time t_0, frequency f). (3) Pulse (general trapezoidal periodic source). (4) Tabulated source (general piece-wise linearization of non-periodic source).

(ii) Controlled Branches

These are the branches that can be described by either a relation between network variables (voltage, current, etc.), or by a relation between element values (resistance, etc.) and network variables.

Using the conventional network definitions, where X is a network variable (voltage V, current I, charge Q, flux F), the general acceptable relations are

$$X_1 = X_1(X_2) \qquad \text{e.g.} \qquad I_b = I_b(V_a) \tag{5.2a}$$

$$\left.\begin{aligned} R &= R(X_1, X_2) \\ G &= G(X_1, X_2) \\ C &= C(X_1, X_2) \\ L &= L(X_1, X_2) \end{aligned}\right\} \quad \text{i.e.} \quad \left\{\begin{aligned} V &= X_1 \cdot R(X_2) && (5.2b) \\ I &= X_1 \cdot G(X_2) && (5.2c) \\ Q &= X_1 \cdot C(X_2) && (5.2d) \\ F &= X_1 \cdot L(X_2) && (5.2e) \end{aligned}\right.$$

Through this general representation, types ranging from constant, linear, self-dependent elements up to cross coupled non-linear elements are acceptable. The only restrictions on the type of dependency are that dependency is not permitted on non-representable variables such as a current through a capacitor or a voltage across an inductor, because these are not treated as equation variables. Also the nonlinearities are restricted to be of the piece-wise linear form, with the exception of a current-or charge-controlled branch which can be of the exponential type as well.

5.2 Order of Setting the Equations and Variables

The tableau representation of the network is set in the following order. First, the branch Kirchhoff's voltage law equations (KVL), in order of increasing branch number. Second, the nodal Kirchhoff's current law equations (KCL), in order of increasing node number.

Third, the branch constitutive equations (B.C), in order of increasing branch number.

The variables represented in the network equations, X, are not the network variables themselves, but the negated differences between their values at a new point (i) and the last accepted point (i-1), i.e.

$$X = Y_{i-1} - Y_i \quad , \tag{5.3}$$

where X is the equation variable, and Y is the corresponding network variable.

The arrangement of variables is in the following order. First, branch current or charge (I or Q), in order of increasing branch number. Second, branch voltage or flux (V or F), in order of increasing branch number. Third, node voltage (VN), in order of increasing node number.

5.3 Solution Procedures and Numerical Techniques

The transient analysis of the network is done by first linearizing the branch equations at a point, this being the last accepted point or a predicted one. The differentiated variables are replaced by a backward difference formula (integration formula) of a

user-specified order k. The linearized system of equations may take the form

$$J X = E \tag{5.4}$$

where J is the Jacobian coefficient matrix, E is the forcing vector, and X is the equation variable vector. Equation (5.4) is solved using the Gaussian LU factorization,

$$J \rightarrow L U$$

$$\therefore \quad L (UX) = E \quad , \tag{5.5}$$

followed by backward and forward substitutions.

Thus, substituting forward in

$$L Y = E \tag{5.6}$$

yields Y, whence, substituting back in

$$U X = Y \quad , \tag{5.7}$$

gives X .

As the system of equations is set in tableau form, the Jacobian $\mathcal{J}$ may be highly sparse and of large dimensions. Also, it can be shown that most of the Jacobian entries are simply ± 1 . More-

over, for each mode of analysis (dc or transient), the same system of equations, with a fixed sparsity of the Jacobian J, is required to be solved repeatedly with different values of J and E. In order to derive the most benefit from these properties, a sparse technique was developed which generates non-looping executable machine instructions (code) through Fortran statements. The code is generated twice, once for the dc analysis and once for the transient. When this code is executed, it performs only necessary operations, avoiding any trivial or unnecessary operations (e.g., multiplication and division by or storage of ± 1).

The linkage and accounting vectors used with the code generation are quite large. However, they are overlaid on other vectors, so that no extra storage space has to be specially reserved for these vectors.

In the following sections, the main techniques used in SCAMPER are reviewed.

5.4 Code Generation

There are two types of machine instructions used in SCAMPER [19], register-to-register (RR) and register-to-core (RX) instructions. The required RR instructions are defined in DATA statements, so that they can be added to the code stream when needed. The

RX instructions consist of an operation code, operand register number, index register number, base register number and displacement field. The operation code, operand register number and index register number occupy the first half word of the instruction, and are defined in DATA statements for every required combination. The index register is used to specify which vector is being operated on, and it contains the address of the start of this vector. The base register number and the displacement field occupy the second half of the instruction word, and define the operand in the specified vector. This is done by setting the second half word to

$$8 \cdot (I - 1) \quad ,$$

where I is the subscript value of the operand. The multiplication by 8 is done because vectors are of double precision. In this way vectors of up to 512 entries (= 4096 / 8) can be assessed. To increase the accessible length of each vector, some registers (1 through 11) are loaded by values, 4096. K, where K is the register number. This allows vectors of entries up to 6144 to be accessed correctly.

Considering equations (5.4) to (5.7) we can see that the code string has to consist of four basic sections, each with a specific function.

(i) J-code, to fill in the constant (real) entries of J. Also to fill in the integration formula corresponding to the linear reactive elements.

(ii) E-code, to compute the r.h.s. E of equation (5.3).

(iii) F-code, to perform a Gaussian LU factorization of Jacobian J, with complete pivoting to ensure numerical stability and high accuracy.

(iv) S-code to perform the solution with the factorized matrix via a forward substitution followed by a backward one.

In addition to these four main segments of the code, some control statements are inserted between segments (such as reloading index registers and transfer of control), as well as some constants to be initialized at the starting locations of the code.

5.5 Sparse Matrix Technique

The sparse matrix technique is used in conjunction with the code generation. It may seem quite complicated due to the large number of linkage and counting vectors used, but it should be noticed that this technique is executed only twice, once for the dc analysis and once for the transient one. Accordingly, a large portion of the time that would be consumed in generating and updating these vectors is saved due to the execution of the generated code. This is not the place for a detailed analysis of this technique, but the main idea and the description of two basic vectors will be given.

During the setting of the linearized network equations, two vectors, IC and ISGN, are constructed to link and identify the non-zero entries of Jacobian J.

(i) Linkage vector IC (IJ), IJ = 1, 2, ... etc., is a list by rows of the non-zero Jacobian entry column numbers. The entries for each row are separated by terminators, so that entries of each row can be identified. This vector is effectively a two dimensional (I, J) vector, where I represents the row number and J represents the column number.

(ii) Identity vector ISGN (IJ), IJ = 1, 2, ... This is a marker vector for the non-zero Jacobian entries and corresponds to IC. Entries which are + 1 or - 1 are identified by 0 and - 1, respectively. Non-zero entries other than ± 1 are identified by their location in the vector VAL where they are stored. Note that ± 1 are not stored in VAL.

Using these two basic vectors IC, ISGN, some other linkage and accounting vectors are constructed and updated during the step of Gaussian LU factorization to facilitate the process of generating the machine code.

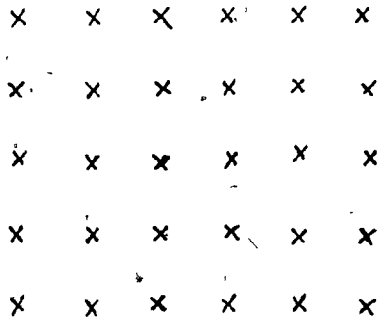
The process of LU factorization is illustrated in Figure 5.1. Assume that we have a full matrix as in Figure 5.1a. Also assume that the diagonal elements have been chosen as pivots. At

the beginning, the whole matrix is linked as in Figure 5.1b. As the process of factorization proceeds the linkage is split into three different paths: (i) the upper trapezoidal, (ii) the lower triangular, and, (iii) the unreduced matrix (Figure 5.1c). The start and the end of these paths are marked by pointers. At the end of the LU factorization both the upper U and the lower L triangular matrices are linked and identified. In general, the matrix is sparse and the pivot elements may not be the diagonals, so that the linkage paths will be overlapped, but the same principle remains.

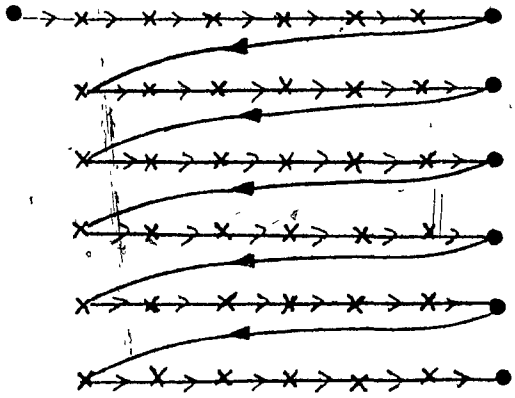
5.6 Pivoting Procedure

The pivoting process is almost a complete pivoting, except for some minor constraints which, from experience [21], are found to give better numerical stability. The algorithm used starts by eliminating the rows corresponding to the B.C. equations, then the KVL equations and the KCL (if possible). Also the choice of the pivot column (if possible) is forced not to correspond first to the node voltage variables. During each step of the factorization, the next pivot row is chosen to avoid a search procedure.

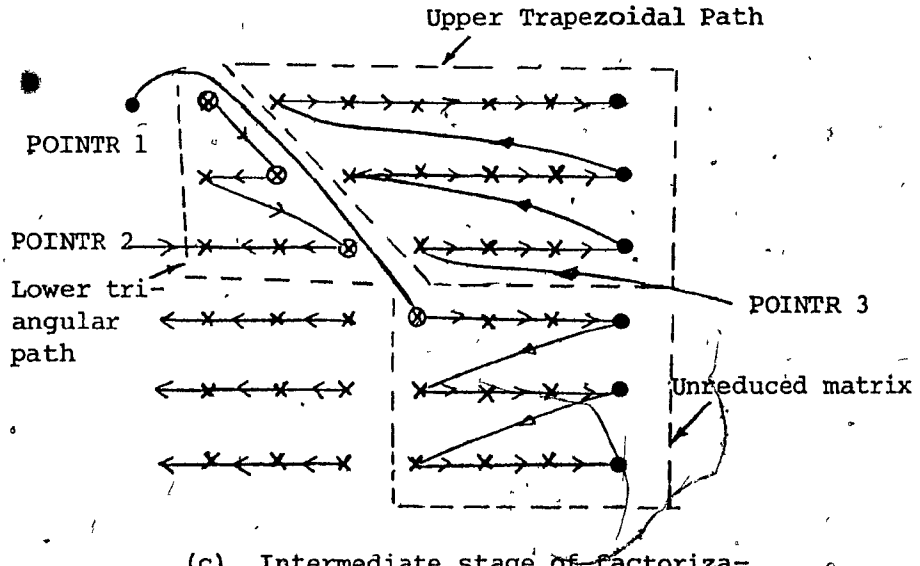
Taking into consideration the above remarks the following is a review of the pivoting algorithm.



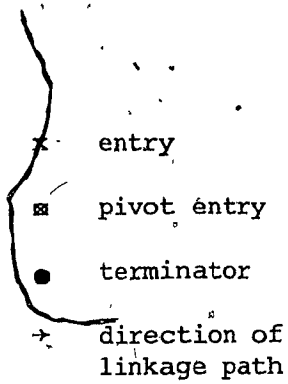
(a) Original matrix.

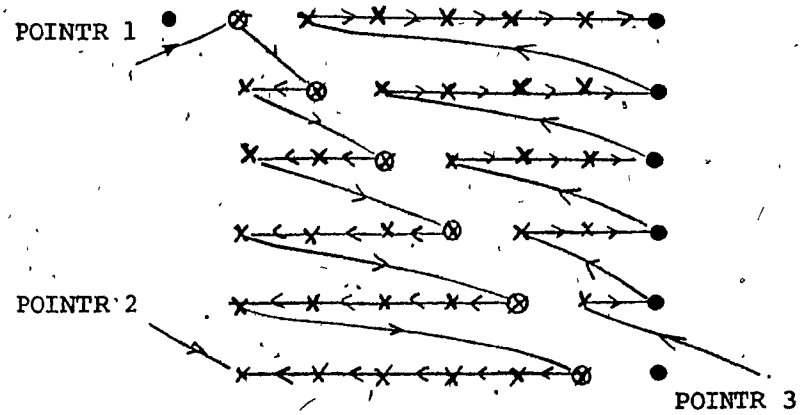


(b) Original linkage of the Jacobian J.



(c) Intermediate stage of factorization linkage.





(d) The final stage of factorization linkage.

Figure 5.1. The process of updating the linkage paths during the LU Factorization.

(1) Set an upper bound KREF on the number of entries in a row to be accepted as a pivot row. (At the start, this limit is set during the construction of the linkage vectors). This acceptance limit might be modified at any step.

(2) During each subsequent step of factorization, compare the number of entries of each row of the unreduced matrix with KREF. If a row of entries equal to KREF or less is found, terminate the search. If this condition is not satisfied after a complete scan choose the row with the smallest number of entries as the pivot row and reset KREF.

(3) From the selected row, choose the pivot column as follows:-

(i) If the first entry is ± 1 (corresponding to ISGN (•) = 0, -1), pick it.

(ii) If not, pick the first unit magnitude (± 1) before the entries corresponding to the node voltage variables.

(iii) If none, choose the largest entry which is greater than a certain lower limit (10^{-60}).

(iv) If none, choose any entry of unit magnitude (± 1) corresponding to a node voltage variable.

As soon as a point is chosen, a step of factorization is carried out: (1) processing the entries of the coefficient Jacobian vector VAL, (2) generating the necessary machine instructions for this step of factorization, (3) updating the linkage and accounting vectors. In order to ensure numerical stability and accuracy the representative Jacobian entries are chosen to correspond to the worst case (e.g., the coefficient of an exponential branch is set to zero).

5.7 Numerical Considerations

A variable $x = x(t)$ can, in general, be represented by a polynomial having an infinite number of terms. Practical considerations limit the number of terms k to $1 \leq k \leq 6$ [28]. The differentiated variable $\dot{x}$ is replaced by the slope of this polynomial at the new time point, which is computed using a "corrector" formula, while the error estimation is done using a "predictor" formula [19]. The error introduced by this "truncation" of the polynomial is called the truncation error. Moreover, when the nonlinear equations are replaced at a given time point by a linear model, the error introduced by this linearization is called the nonlinear error. Accordingly, there are two types of error control, the truncation error control and the nonlinear error control. The step size is chosen to limit the largest error to be less than or equal to a certain user-specified limit.

5.7.1 Truncation Error

The predictor-corrector algorithm replaces the differentiated variable $\dot{x}$ by the slope of the truncated polynomial of order k at the new time point t_n. The associated local truncation error E_{tr} is defined by [18].

$$h\,(\dot{x}_{n\ \text{true}} - \dot{x}_{n\ \text{approx}}) = E_{tr} + 0\,(h^{k+2}) \quad , \tag{5.8}$$

where $\dot{x}_{n\ \text{approx}}$ denotes the value yielded by the corrector formula. Neglecting the term $0\,(h^{k+2})$ it can be proved [18] that

$$E_{tr} = \frac{h}{h_t} \left| x_n - x_n^p \right| \tag{5.9}$$

where $h_t = t_n - t_{n-k-1}$, $h = t_n - t_{n-1}$, and x_n^p is a predicted value of x_n.

5.7.2 Nonlinear Error

This error is considered only for the exponential type of nonlinearity. If the equation for, say, an exponential capacitor

$$q = a\,(e^{bv} - 1) \quad ,$$

is linearized at a predicted value v^p, it can be proved [19], that the nonlinear error E_n can be approximated by

$$E_n = \frac{b}{2} (v - v^p)^2 \qquad (5.10)$$

5.7.3 Step Size Control

Step size h, is determined according to two criteria: the truncation error criterion (with a lower limit h = H0.), and the nonlinear error criterion (with a lower limit h = H00 where H00 is related to H0 and k) .

Let us consider

X , the normalized maximum truncation error,

Y , the normalized maximum nonlinear error,

h_n , the last (accepted) step size,

h_{n+1} , is the new step size.

In SCAMPER, the step size is chosen such that the maximum error is less than or equal to half a user-specified error.

Accordingly, the step size determined by the truncation error criterion is

$$h_{n+1} = h_n \left(\frac{0.5}{X}\right)^{\frac{1}{k+1}} \qquad (5.11)$$

and, that determined by the nonlinear error criterion is

$$h_{n+1} = h_n \left(\frac{0.5}{Y}\right)^{\frac{1}{k+1}} \qquad (5.12)$$

The smaller of h_{n+1} and h'_{n+1} is taken as the new step size. There are some other constraints on the step size such as: (1) avoiding wide change of h_{new} / h_{old} (say, $< 10, 0.1 >$) to achieve better numerical stability ; (2) choosing h_{new} to yield a point just after the cusp change of independent sources ; (3) choosing h_{new} to yield points at specified times to output results at these points.

PART B

STEADY-STATE ANALYSIS ROUTINE

5.8 Code Generation for the Adjoint System

One of the main reasons for the superiority of the dc and transient analysis algorithms used in the program SCAMPER is the successful marriage between a highly sophisticated sparse matrix technique and a compiler-like routine for generating machine instruction (code) strings. Once the code is generated it is used repeatedly in constructing and solving the network equations. When executed, this code does only the necessary operations, avoiding any trivial work such as multiplication or division by unity or the storage thereof. Since the steady-state analysis method considered here involves the proposed gradient algorithm 3.1, the adjoint system has to be integrated as well as the original one. To achieve a performance consistent with that of SCAMPER it was necessary to modify the compiler-like routine to generate another code for integrating the adjoint system in an efficient way, exploiting the common properties of both systems.

Let us first consider the actual representation of the original system. The linearized form of the network relations is

$$J\ X = E_f \tag{5.13}$$

Substituting the definitions given in Part A of this chapter, this becomes

$$
\begin{array}{c}
\qquad\quad \overbrace{\qquad\quad}^{(I,\,Q)} \qquad \overbrace{\qquad\quad}^{(V,\,F)} \qquad \overbrace{\qquad\quad}^{VN} \\
\begin{array}{c} \\ KVL \\ \\ \\ KCL \\ \\ \\ BC \end{array}
\left[\begin{array}{ccc}
D\,(n) & \begin{array}{c}\text{Alg.}(\pm\, I) \\ D\,(\ell,\, n)\end{array} & \text{Alg.}(\pm\, 1) \\
\text{Alg.}(\pm\, 1) & D\,(n) & 0 \\
D\,(\ell,\, n) & & \\
\text{Alg.} & \text{Alg.} & 0
\end{array}\right]
\left[\begin{array}{c} (I,\,Q) \\ (V,\,F) \\ VN \end{array}\right]
= E_f
\end{array}
$$

where $D\,(\cdot)$ symbolizes the entries of a block of the Jacobian J which are differentiated, while ℓ and n represent, respectively, linear and nonlinear terms.

This system of linearized equations is solved by using Gaussian LU factorization followed by forward and backward substitution. The pivoting procedure [19] forces the pivot row to be taken in the following order: first, rows corresponding to the branch constitutive relations, second KVL, third the KCL (if possible). Also, for the pivot columns, those corresponding to the node voltage variables VN are forced to be the last (if possible).

Consider now the adjoint system,

$$J^t \lambda = E_b \tag{5.14}$$

or

$$
\begin{array}{c}
\\
(I,Q) \\
(V,F) \\
(V,N)
\end{array}
\begin{array}{c}
\overbrace{\qquad\qquad}^{\text{KVL}}\ \overbrace{\qquad\qquad}^{\text{KCL}}\ \overbrace{\qquad\qquad}^{\text{BC}} \\
\left[\begin{array}{c|c|c}
 & \text{Alg.}(\pm 1) & \\
\text{D (n)} & \text{D } (\ell, \text{n}) & \text{Alg.} \\
\hline
\text{Alg.}(\pm 1) & & \\
\text{D } (\ell, \text{n}) & \text{D (n)} & \text{Alg.} \\
\hline
\text{Alg.}(\pm 1) & 0 & 0
\end{array}\right]
\end{array}
\left[\begin{array}{c} \lambda_v \\ \lambda_c \\ \lambda_{bc} \end{array}\right]
= E_b \ .
$$

This adjoint system may be solved using procedures similar to those for the original one, e.g., by using Gaussian LU factorization, and forward and backward substitution. The additional requirements for generating a solution code are:

(i) to manipulate the transposition of the matrix in such a way as to use the same J-code (which fills in the linear entries of the Jacobian) for both the original and the adjoint system;

(ii) to make use of the relation between the two systems to simplify the procedure for generating the E-code (which computes the r.h.s. vector E_b) ;

(iii) to modify the pivot selection algorithm in order to achieve requirements similar to those imposed on the pivoting order of the original system of equations, and also to use the same routine which generates the F- and S- codes for both systems.

The way in which these requirements were satisfied is outlined in the following sections.

5.8.1 Transposition of the Jacobian

As discussed in Part A, the Jacobian is mainly described by three vectors:

(i) VAL, the coefficient Jacobian vector which holds the real values of J (not including $\pm$ 1) ;

(ii) IC, which links the whole matrix by row in order of increasing column number, with rows identified by terminator entries ;

(iii) ISGN, the identity vector, which describes the type of entries located by IC (real, $\pm$ 1).

A typical sketch of these vectors is shown in Figure 5.2 for a 9 x 9 matrix.

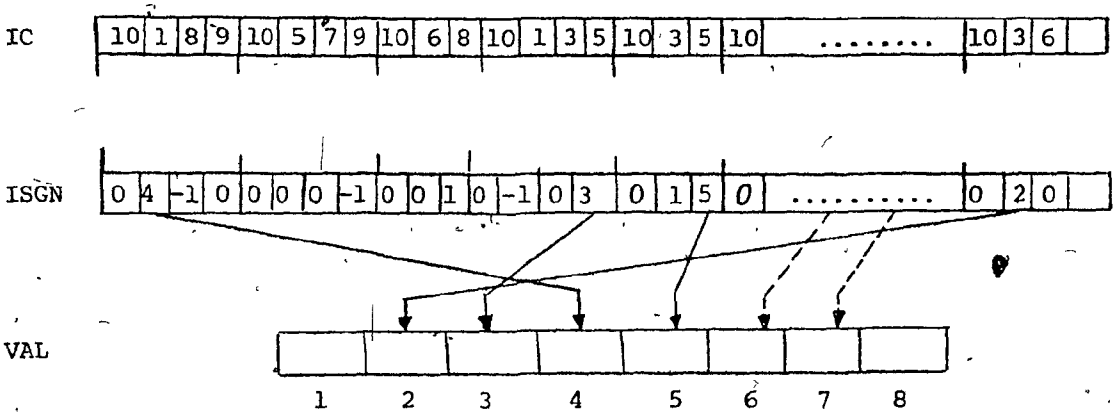


Figure 5.2. Typical representation of some entries of vectors IC, ISGN and VAL.

Since the execution of the J-code fills in the linear entries of VAL (with the nonlinear entries to be filled via Fortran statements), hence, if IC and ISGN are rearranged in such a way as to keep the order of VAL unchanged, the J-code can serve both the original and the adjoint system. This is achieved by means of the following steps :

(i) Count the number of entries in each row of the adjoint system using IC ;

(ii) use the above counting vector to set the terminators of each row ;

(iii) scan the linkage vector IC to fill the entries of each row of the corresponding linkage vector

ICJ in order of increasing column number and, at the same time, for each entry of IC fill in the corresponding identity vector ISGNJ, i.e., if IC (NL) → ICJ (NLJ), hence take ISGN (NL) → ISGNJ (NLJ).

In the actual program, not only ICJ and ISGNJ are constructed, but the other linkage and counting vectors required for the LU factorization are generated simultaneously.

5.8.2 E-Code Generation

For the original system, the equation variables are taken as the (negated) differences between the current network variables x_n and the last accepted solution x_{n-1}, i.e., $\Delta x = x_{n-1} - x_n$. Furthermore, the differentiated variable $\dot{x}_n$ is replaced by an integration formula (polynomial) of order k, $1 \leq k \leq 6$,

$$\dot{x}_n = C_0 \Delta x + BDF \,, \tag{5.15}$$

where $\Delta x = x_{n-1} - x_n$ and BDF is the explicit part of $\dot{x}_n$ (backward difference formula) which depends on the last k differences as well as on the step sizes. The term BDF will be added to the r.h.s of the equation.

Now, for the adjoint system a general form of the equation is

$$\sum a_i \lambda_i + \sum b_j \dot{\lambda}_j = 0 \qquad (5.16)$$

If we use the difference $\Delta\lambda$ as variable, $\Delta\lambda = \lambda_{old} - \lambda_{new}$, equation (5.16) takes the form

$$\sum a_i \Delta\lambda_i + \sum b_j c_0 \Delta\lambda_j = \sum a_i \lambda_i + \sum b_j BDF_j \qquad (5.17)$$

Equation (5.17) is used to compute the r.h.s. entries of equation (5.14) corresponding to the (I, Q) and (V, F) rows, while for the equations corresponding to VN the corresponding entries are set to zero. The special functional structure of the Jacobian is exploited to significantly reduce the complexity of the E-code generator.

5.8.3 F- and S-Code Generation and Pivoting

To formulate the Gaussian LU factorization, it was decided to use a pivoting procedure similar to that used in the forward integration. Thus, for the adjoint system, the rows corresponding to node voltages VN and the columns corresponding to the KCL variables are forced to be the last (if possible). This has been achieved by modifying the pivoting algorithm in such a way that it can handle both systems, the original and its adjoint. By doing this, one routine

SGEN is used for generating the F- and S- codes of the original system and its adjoint as well.

In summary, the original Jacobian J is transposed by changing the linkage and the identity vectors IC, ISGN in such a way as to keep the coefficient Jacobian vector VAL unchanged. Thus, the J-code generated for the original system serves the adjoint system as well. The functional structure of the Jacobian is utilized to simplify the E-code generating routine. The pivoting procedure has been modified so that it can handle both the original and the adjoint system in a similar manner and so that the same routine for generating the F- and S-codes can be used for both systems.

5.9 Data Management

Dealing with the integration of two different systems requires some kind of data management (storing, retrieving and processing). Some of these data are :

(i) the necessary values of the nonlinear entries of the Jacobian J as well as the corresponding step size (or the time) at different time points over one full period ;

(ii) the two machine instruction strings (codes) generated for solving both the original and the adjoint systems ;

(iii) vector of $(1/C, 1/L, C(0), L(0))$, in the order of reactive branch number, where $C(0)$ and $L(0)$ are the values of the nonlinear capacitive and inductive elements respectively at the starting time point t_0. Even though these values can be retrieved from the data in item (i), it is found more efficient to have this vector separately and arranged in a particular way;

(iv) the initial condition vector of the reactive variables q_0 as well as the solution of the network variables at the beginning of the first steady-state iteration.

Items (ii) to (iv) are each treated as one unit (sequential) and their manipulation is a trivial matter. This is not the case for data item (i). The nonlinear entries and the step sizes are evaluated at a series of time points over one full period during the integration of the original system. For the integration of the adjoint system, this data has to be retrieved in the opposite order and, hence, care must be taken to minimize both storage and the time for retrieving data from disk. We found it most efficient to define a temporary storage vector AREA with suitable pointers and linkage vectors to be used with a particular scheme for handling this problem. The size of AREA was chosen large enough to ensure that in the worst case* it can hold data of at least three time points.

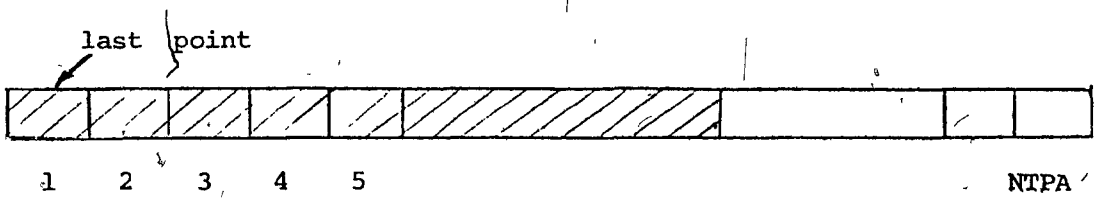
---

* By worst case we mean the largest problem that can be solved by the program (380 branches) with most of the branches nonlinear.

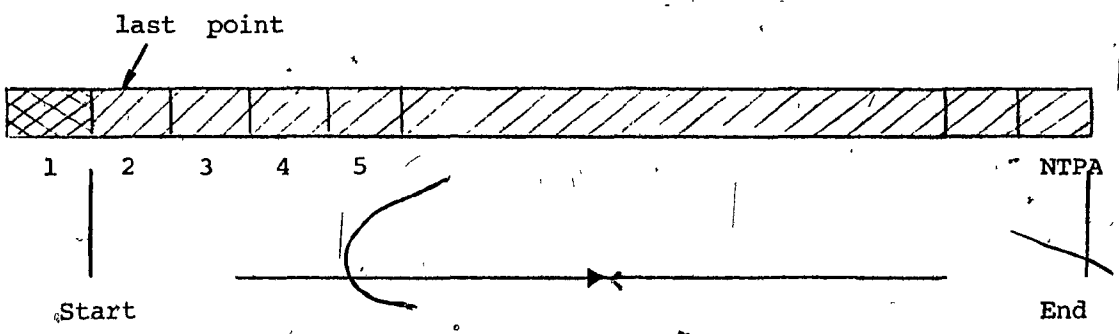
Suppose that the temporary storage vector AREA has NT entries and the number of entries per time point is NETP (the number of nonlinear entries plus one for step size). Hence, the maximum number of time points that can be held in AREA is NTPA = NT/NETP (rounded to the least integer number). Since interpolation between time points may be required for computing the Jacobian J, it is necessary to keep the data of the last time point in AREA before reloading it from disk with a new record. Accordingly, the "dynamic" size of AREA is (NTPA-1) time points. That means the transfer of data from and into disk takes place every (NTPA-1) points. If the data is retrieved from disk more than once, and because we have to keep the last point of the previous record, the starting (and the end) location of the newly retrieved record will move in a loop. A sketch of this situation is given in Figure 5.3. This process is controlled by defining pointers that change in a closed loop.

5.10 Some Aspects of the Backward Integration of the Adjoint System

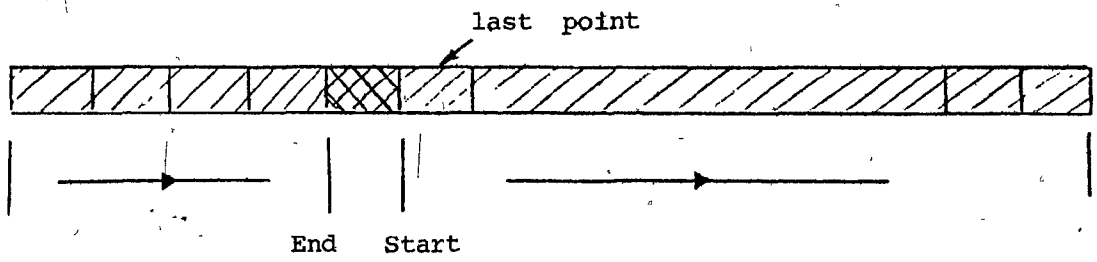
Figure 5.4 is a sketch of the adjoint system integration loop. It represents the integration of equation (3.19) over a period in the opposite order to that of integrating the original system given by equation (3.5). Requirements for the adjoint system integration are quite different from those for the original system.



(a) The first record.



(b) The second record (first retrieved from disk).



(c) A general record.

New data.

Old point from the previous record.

Sequence of loading.

Figure 5.3. Sketch to show different cases of the temporary storage Vector AREA.

(i) For the original system the required data are computed at each step using the currently available values of the variables, while for the adjoint system these data already exist but may need some manipulation (retrieval, interpolation).

(ii) The number of differentiated variables of the adjoint system λ_2 may be different from that of the original system. Accordingly, the truncation error computation as well as setting the initial conditions for both systems will be different.

(iii) In the original system, there are two error criteria to be applied : the truncation and nonlinear error criteria. For the adjoint system the only criterion is the truncation error control since the system is linear (time varying if the original system is nonlinear).

(iv) For the original system the step size is determined by the error control as well as the "cusp" changes (if any) of the independent sources, while for the adjoint the step size is determined by the error control as well as by the interpolation process.

5.10.1 Setting the Initial Conditions of the Adjoint System

As the initial conditions of the algebraic variables λ_1 can be chosen arbitrarily we simply put $\lambda_1(T) = 0$. The initial conditions of the differentiated variables $\lambda_2(T)$, are defined by equation (3.17), with the detailed equations for the different types of

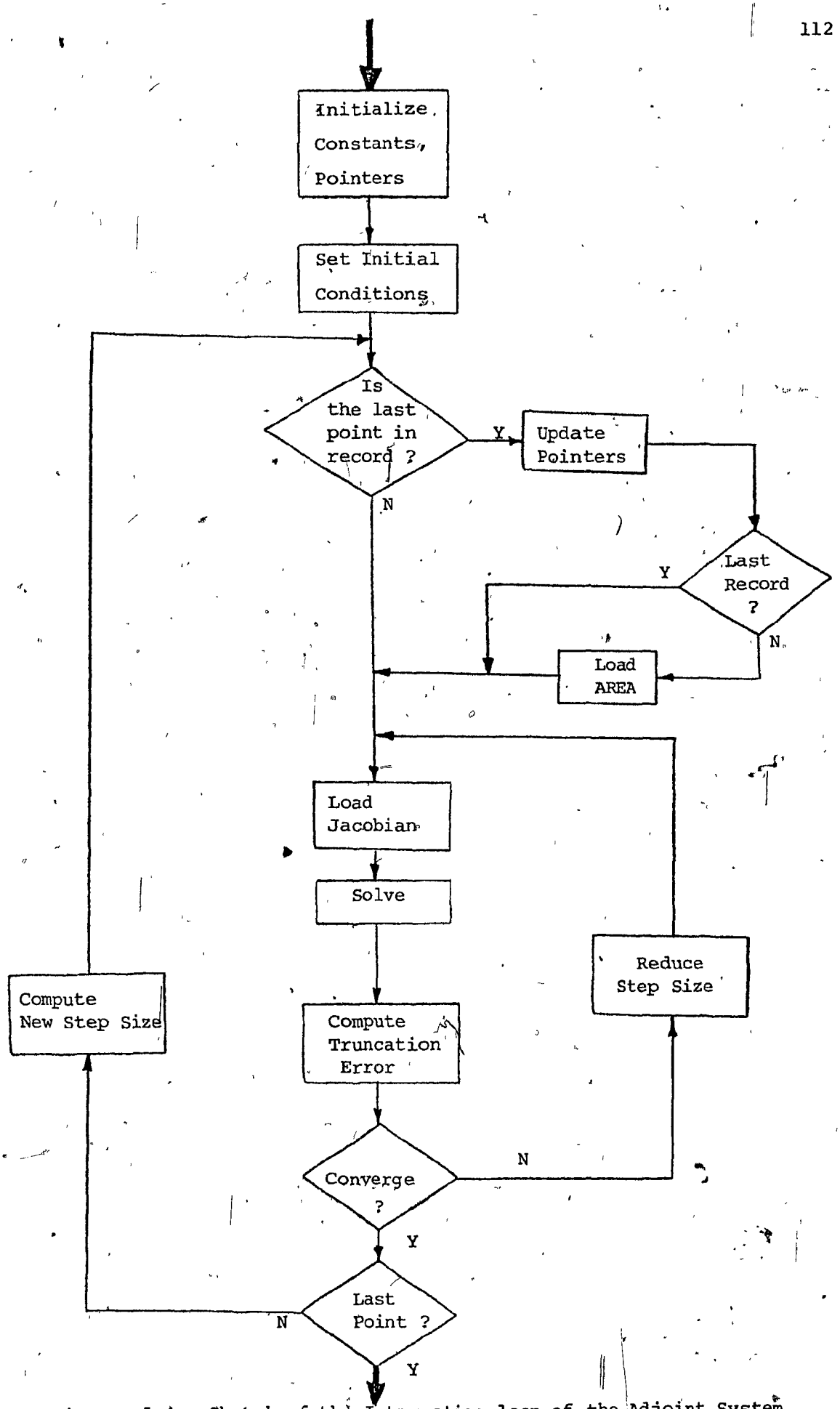


Figure 5.4. Sketch of the Integration loop of the Adjoint System.

reactive elements given by equations (3.22a, b, c, d). Corresponding to each inductive branch there is only one differentiated variable. In the case of capacitive branches we are faced with two problems: (1) corresponding to each branch there may be two differentiated variables (one for each node); (2) there may be loops of only capacitors and/or independent voltage sources and/or short circuit branches. In order to overcome the second problem an algorithm was formed to identify such loops and to ignore one of the capacitors in such a loop in computing the initial conditions. To get around the first problem an algorithm was designed to construct linkage vectors which are used to set the initial conditions in the correct and proper way. The outline of this algorithm is as follows.

Algorithm 5.1

1. Count the number of trees that consist of only capacitors and/or independent voltage sources and/or short circuits.

2. Count the number of capacitors connected to each node in these trees.

3. If there is a tree which contains the ground node (if the ground node is specified) consider it to be the first tree and consider the ground node the first node in this tree. For any other tree consider the first node to be the one which connects the largest number of capacitors.

Put the variable corresponding to this node equal to zero and proceed.

4. Following from the previous node NI choose the next node NJ separated from NI by one branch, C_j and set the initial condition of this node, λ_{NI} according to the relation

$$\lambda_{NJ} = \pm \left(\frac{\omega_j}{C_j} \left(v_j(0) - v_j(T) \pm \lambda_{NI}\right)\right) \qquad (5.18)$$

5.10.2 Truncation Error Computation

The computation of the truncation error of variables related to inductive branches is done in a similar manner to that for the original system. Corresponding to each capacitive branch, the difference between the two node variables is defined and the initial conditions are set such that this difference is equal to $(\Delta v_0 / C)$. Since the capacitors in the circuit may differ by orders of magnitude, the control of the absolute values of the variables is meaningless. Hence, the control is carried out on the differences corresponding to each capacitive branch. Consider a capacitor connected between two nodes NI and N2. Define a variable

$$z = \lambda_{N1} - \lambda_{N2} \qquad (5.19)$$

Hence

$$\dot{z} = \dot{\lambda}_{N1} - \dot{\lambda}_{N2} \tag{5.20}$$

If the differentiated variable $\dot{z}$ is replaced by an integration formula (polynomial) of order k, the predicted value will be

$$z^p = \lambda_{N1}^p - \lambda_{N2}^p \tag{5.21}$$

Accordingly, the truncation error control of the differences corresponding to each capacitive element is done according to equations (5.19) and (5.21) i.e., on the difference $| z - z^p |$, and not on $| \lambda_{N1} - \lambda_{N1}^p |$ or $| \lambda_{N2} - \lambda_{N2}^p |$.

5.10.3 Interpolation and Step Size Control

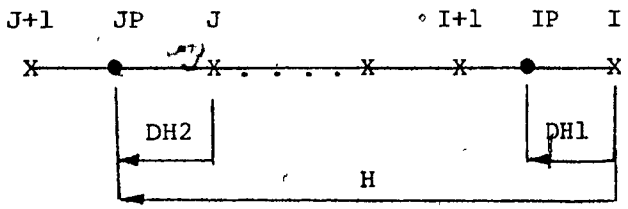
Suppose the largest normalized truncation error computed at a time point is X. If the last step size is h_n, the new step size h_{n+1} is chosen to produce half the normalized truncation error (the same as for the original system integration) i.e.,

$$h_{n+1} = h_n \left(\frac{0.5}{X}\right)^{\frac{1}{k+1}}$$

The step size is subject to other restrictions such as the ratio h_{n+1} / h_n having to be within certain limits $< \varepsilon_1, \varepsilon_2 >$ (say $<10, 0.1>$). Also, if the last point was rejected the step size is not allowed to increase.

Another restriction is set by the interpolation between points and by the end-of-record condition as will be shown next.

For each time point the step size h_n is stored along with the nonlinear entries. Considering the backward integration, suppose the precomputed step size is H, and IP defines the location of the last accepted point, where the time difference between IP and the preceding point is DH1 as shown below. A pointer J



is moved starting from value I and scanning step sizes h_{I+1}, h_{I+2}, ... etc., accumulating a step h

$$h = (h_I + h_{I+1} + \ldots + h_{J-1}) - DH1$$

The accumulation of h terminates when

$$h + h_J > H \ .$$

A new point JP will be taken between (or coincident with) points J and J+1. If JP is within a certain limit (say $|H - h| / H < 0.01$) of points J or J+1, the nearest point will be taken as the new point. If the new point is accepted, pointers will be reset to I = J, IP = JP and DH1 = DH2.

For the special case when the last accepted point is equal to or just before the last point I of the temporary storage vector AREA, and

$$h = h_I - DH1 < H ,$$

there are two possibilities:

(1) if $h > HO$, where HO is a lowest limit on the step size, H will be set to $H = h$ and IP to I ;

(2) if $h < HO$, this means that the last point I in AREA precedes any new acceptable point. In this case new data will be retrieved from disk in place of the previous data, except for the last point I. Pointer IP will be set to I and $DH1 = DH1 - h_{I-1}$. The process of interpolation will then continue normally.

It should be mentioned that we use a linear interpolation process. For example, if a point JP is chosen between two points J and J+1 with corresponding values X_J and X_{J+1} respectively, the value corresponding to JP, X_{JP} is given by

$$X_{JP} = X_J + \frac{DH2}{h_J} (X_{J+1} - X_J)$$

5.11 Resetting the Initial Conditions of the Original System

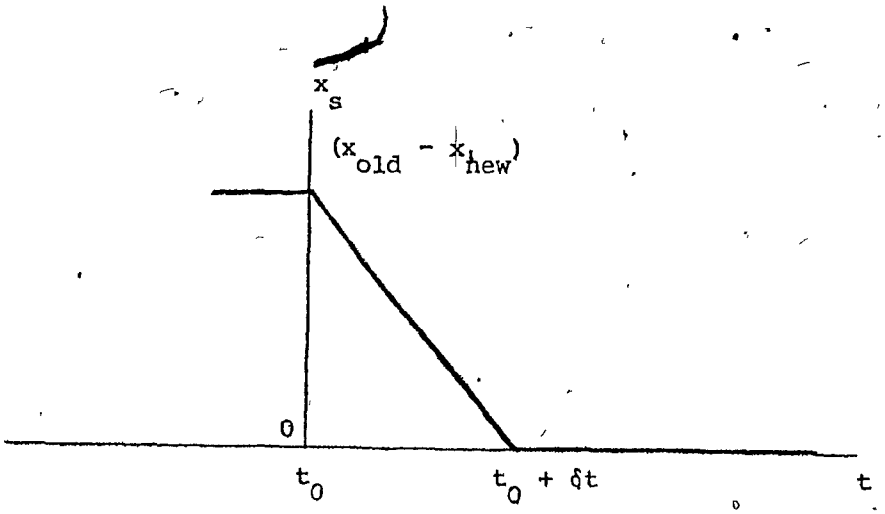
The values of the differentiated variables of the original system corresponding to the reactive elements, before and after the call of the optimization routine, may differ considerably. Due to the nonlinearity of the network, the integration procedure may fail completely to converge at the first step of integration if there are such big changes. To avoid this problem, the solution may proceed as follows.

Corresponding to each reactive element an auxiliary source is added: an independent voltage source E_c in series with a capacitive branch and an independent current source J_ℓ in parallel with an inductive branch. Suppose the new value of an initial condition is x_{new} and an old value is x_{old} (in our case the old value is that corresponding to the starting point of the first steady-state iteration). Assume that this previous value satisfies the system equations, and that the solution vector V_{old} at that point is available (or at least the variables corresponding to the nonlinear elements). We set the value of the corresponding source x_s at the starting point t_0 to be

$$x_s(t_0) = x_{old} - x_{new},$$

and at $t_0 + \delta t$ to be

$x_s\ (t_0 + \delta t) = 0$, as shown below, where δt



is chosen small, relative to the period of integration. However, the initial conditions of the corresponding reactive element is set equal to x_{new}. As a result, the terminal variable, x of each reactive element (including the auxiliary source) start with $x = x_{old}$ at $t = t_0$ and approaches x_{new} near $t = t_0 + \delta t$. In this way, the integration of the network equations starts with initial conditions which satisfy the nonlinear error criteria and which can then change to the new initial conditions x_{new} in several steps within the interval from t_0 to $t_0 + \delta t$.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

The gradient method for computing the periodic steady-state response of nonlinear networks has been successfully implemented with the dc and transient analysis program SCAMPER which can be used to analyze large networks by virtue of its being based on the sparse tableau formulation of network equations. This periodic response solver has been tested with several circuits including those which were used by others working on this problem, in particular, Nakhla and Branin [10], [11] who first proposed the gradient method but used the state variable approach to solve the network equations, which is only practical for small systems.

A by product of the derivation of the equations for computing the gradient vector was the proof that they are valid for circuits having nonlinear elements involving not only single [6] but also double dependencies. This has been verified for several examples [Appendix II].

As predicted, the time per iteration never exceeded twice the time for one forward integration over one period. The rate of convergence depends very much on the characteristics of the optimization routine, the scaling of the variables and the associated gradients, and on the choice of the starting time t_0.

The high standards set by the authors of SCAMPER were adhered to in coding the gradient algorithm. Thus, the setting up and solution of the adjoint system of equations is also achieved by a routine which generates executable machine instructions, and which was designed to exploit features common to both the original and the adjoint systems. Furthermore a large number of the subroutines used in the integration are common to both systems.

Several important and potentially fruitful areas of investigation present themselves as natural extensions of the present work. Various improvements that can be made in the existing program are also apparent. These are discussed below in an arbitrary order.

(1) To reduce the computation time, the error control should be modified. Assume a user supplies an error limit E_{max} which, correspondingly, causes the results to be accurate to, say, n digits. During the first few iterations lower accuracy may be satisfactory, hence the error limit can be larger which, in turn, causes a reduction of the integration time. Then, as the solution converges to the steady-state condition, the error limit can be reduced in steps to the user-specified limit. If this would be done, there would be no need for the extra control imposed on the adjoint system initial conditions for changing the accuracy of computation.

(2) The initial conditions from which the steady-state analysis starts are determined by integrating the system equations up to a

specified time point t_0. In some cases it may be more advantageous to start the analysis with a set of user-defined initial conditions. Work on this feature is now in progress.

(3) A linear interpolation process is used for computing the Jacobian of the adjoint system. According to the tested problems, this process works quite well, but cases may be encountered where a higher order of interpolation (2nd, 3rd, ... etc.) may be necessary.

(4) In Chapter IV some comments are made concerning the scaling problem in the light of our admittedly limited experience with the program. Further work is needed to achieve a more systematic scheme for using the scaling facilities that are made available to the user.

(5) Considerable work remains to be done on the problem of the design of an efficient function minimization routine. Although the variable metric routine was more successful in solving the test problems than the conjugate gradients one, its initial convergence was, nevertheless, quite slow in some cases. Moreover, the variable metric algorithm will present storage problems when large networks are tackled.

(6) The present investigation was concerned with the periodic steady-state analysis with periodic input (nonautonomous). The starting time t_0 of the analysis was regarded as a constant, in fact, the gradient depends on the choice of the starting point and, accordingly, the convergence criteria of the gradient method depends on that choice. In

other words, by letting the starting point t_0 change, a point can be found at which the computed gradient yield better convergence [11]. To do so, we redefine the objective function and the variable space as well as the gradient to include t_0 as a variable, i.e.,

$$\Phi = \Phi (X_0 , t_0) .$$

Hence the gradient with respect to (X_0, t_0) will be

$$G = \left(\frac{\partial \Phi}{\partial X_0} , \frac{\partial \Phi}{\partial t_0} \right) .$$

By passing the objective function Φ, gradient G and the variables (X_0, t_0) to the optimization routine not only the variables X_0 are updated but also t_0 is updated towards an "optimal" point. With this modification the gradient method is expected to give better results. Also, an extension of the method to the oscillatory (autonomous) case can be achieved with little effort, especially after including optimization of the starting point.

APPENDIX I

ACCEPTABILITY OF DOUBLE DEPENDENCY REACTIVE ELEMENT

To show that the term in equation (3.14)

$$\frac{\partial u}{\partial t} \cdot \frac{\partial x}{\partial p} - \frac{\partial u}{\partial p} \frac{\partial x}{\partial t}$$

is identically zero for a class of systems that have a unique solution over (t) for a given range of parameter {p} . Let the system variables (x, u) can be represented as functions of the parameters p, t, i.e.,

$$x = x(p, t) \quad ,$$

$$u = u(p, t) \quad . \tag{I.1}$$

The parametric relation between x and u may take any of two implicit forms

$$\psi(x, u, t) = 0 \quad , \tag{I.2}$$

where t is taken as the running parameter, or

$$\varepsilon(x, u, p) = 0 \quad , \tag{I.3}$$

where p is the running parameter.

We are interested in practical problems where both the variables x and u, and the p are within a finite range. Let us

consider a point in the (x, u) plane where $p = \hat{p}$ and $t = \hat{t}$. The two functions

$$\psi (x, u, \hat{t}) = 0 \tag{I.4}$$

$$\varepsilon (x, u, \hat{p}) = 0 \tag{I.5}$$

will, in general, intersect at one or more points $a_1, a_2, \ldots,$ as shown in Figure I.1. The interpretation of this figure is that, if the two curves intersect at more than one point the system has more than one solution defined by points $a_1, a_2, \ldots$. Hence, for our proof, we consider that class of systems which can be represented by Figure I.2 where both curves are tangent at a single point. In other words, for a given value of p, $p = \hat{p}$ a group of curves (I), corresponding to equation (I.2) can be found such that curve (II), representing equation (I.3) for a given value of $p = \hat{p}$, is the envelope of this group.

If the group of curves $\psi = 0$ is expressed parametrically in the form of equation (I.1), then equation (I.2) can be replaced by a function $\phi (p, t)$ where

$$\phi (p, t) = \psi (x (p, t), u (p, t), t) = 0 \tag{I.6}$$

Hence, at any point on any curve of the group $\psi = 0$

$$\frac{\partial \phi}{\partial p} = \frac{\partial \psi}{\partial x} \frac{\partial x}{\partial p} + \frac{\partial \psi}{\partial u} \frac{\partial u}{\partial p} = 0 \tag{I.7}$$

and

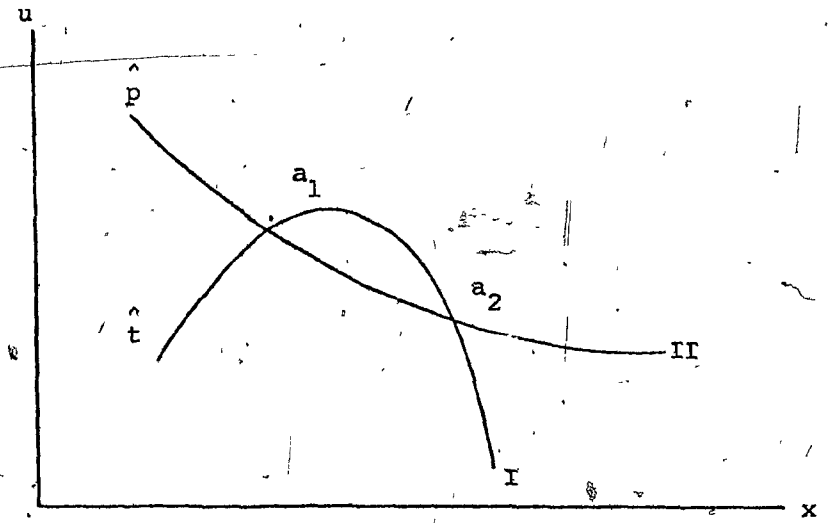


Figure I.1. General representation of equation (I.4) by curve I , and equation (I.5) by curve II.

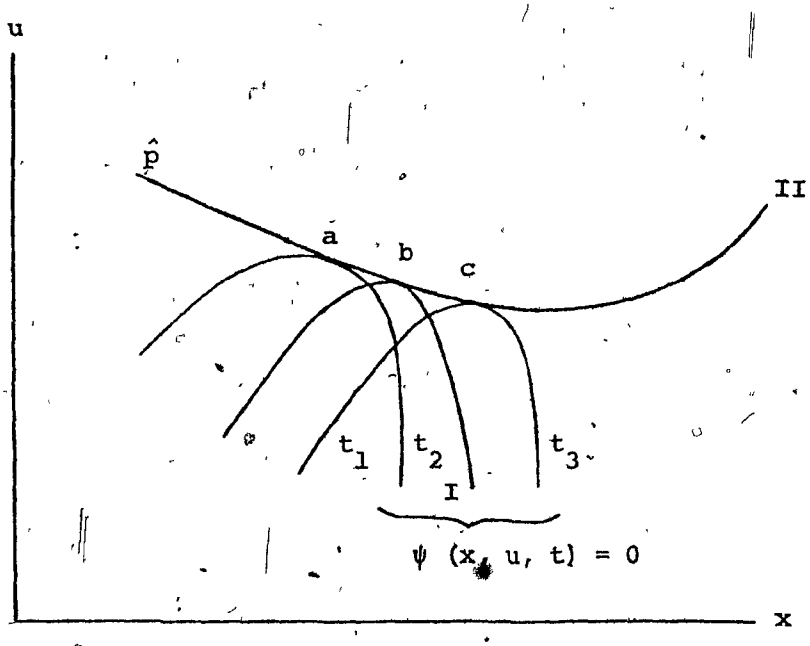


Figure I.2. Representation of family of curves I given by equation (I.2) for different values of t, and equation (I.3) for a corresponding p̂ (II).

$$\frac{\partial \phi}{\partial t} = \frac{\partial \psi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \psi}{\partial u}\frac{\partial u}{\partial t} + \frac{\partial \psi}{\partial t} = 0 \qquad \text{(I.8)}$$

For a given value $p = \hat{p}$, the solution of the system (x, u) for t as a parameter will be given by curve (II), i.e., by the envelope of group (I). At any point on the envelope [24]

$$\frac{\partial \psi}{\partial t} = 0 \,. \qquad \text{(I.9)}$$

hence

$$\frac{\partial \phi}{\partial t} = \frac{\partial \psi}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial \psi}{\partial u}\frac{\partial u}{\partial p} \qquad \text{(I.10)}$$

If we multiply equation (I.7) by $\frac{\partial x}{\partial t}$ and equation (I.10) by $\frac{\partial x}{\partial p}$ and subtract, we obtain

$$\frac{\partial \psi}{\partial u}\left[\frac{\partial u}{\partial t}\frac{\partial x}{\partial p} - \frac{\partial u}{\partial p}\frac{\partial x}{\partial t}\right] = 0$$

But $\frac{\partial \psi}{\partial u}$ may not equal zero .

Hence

$$\frac{\partial u}{\partial t}\frac{\partial x}{\partial p} - \frac{\partial u}{\partial p}\frac{\partial x}{\partial t} = 0$$

APPENDIX II

VERIFICATION OF ACCEPTABILITY OF DOUBLE DEPENDENCY

Several nonlinear and time varying differential equations which require the use of nonlinear circuit elements with single or double dependencies when simulated on SCAMPER, and for which solutions can be obtained in closed form, were used to test the validity of the gradient derivation in Chapter III. Results of direct hand calculations and of the use of the program described in this thesis are compared below. It should be noted that, due to the inherent nature of the SCAMPER simulation, such as the restrictions imposed by the set up routine of this program, exact agreement can not be expected. Moreover, lower accuracy should be expected in computing the gradient G because it is obtained after two steps of integration, the forward integration and the backward one.

The results will be compared on the basis of the following definitions.

Let the differential equation be

$$\dot{x} = f(x, t), \tag{II.1}$$

the starting time point t_0 and the period T.

We define

$$x_0 = x(t_0) \quad ,$$
$$x_T = x(t_0 + T) \quad . \tag{II.2}$$

From the definition of the objective function ϕ

$$\phi = \frac{1}{2} (x_0 - x_T)^2 \qquad \text{(II.3)}$$

the gradient is

$$G = \frac{\partial \phi}{\partial x_0} = (x_0 - x_T) \left(1 - \frac{\partial x_T}{\partial x_0}\right) \qquad \text{(II.4)}$$

Example 1

Consider the differential equation

$$\dot{x} = - t (x - 1) \qquad \text{(II.5)}$$

solving we get

$$x = 1 + (x_0 - 1) e^{-\frac{1}{2} (t^2 - t_0^2)} \qquad \text{(II.6)}$$

The problem was simulated by simulating

$$\dot{x} + t x - t = 0$$

twice, once as a node equation (Figure II.1a) and once by using its dual loop equation. In the simulation, t_0 and T were taken to be $t_0 = 1$, $T = 2$. Step size control, in fact, forced the starting point to be $t_0 = 1.0062$. Substituting these values and choosing x_0 such that $x(0) = 0$ we obtain

$$x_0 = 0.3974 ,$$

$$x_T = 0.9891 ,$$

$$G = 0.5805 . \qquad \text{(II.7)}$$

The results from the simulation were

$$x_0 = 0.3972 ,$$

$$x_T = 0.9891 ,$$

$$G = 0.5811 . \qquad \text{(II.8)}$$

Thus, the gradients are in agreement to three digits (< 0.1 %). In Figure II.1 the representation of the nonlinear resistive element G1 was

$$I_{G1} = G1 \ (I_{J1}) \ V_{G1}$$

In another simulation the equation was rewritten as

$$(\frac{1}{t}) \ x^{\cdot} + x - 1 = 0 , \qquad \text{(II.9)}$$

where the derivative term was represented as in Figure II.2 by a nonlinear inductor L (or C in the dual network) of value equal to 1 / t . More specifically, the reactive element is modelled through a double dependency

$$F_{L1} = L1 \ (I_{R2}) \ . \ I_{L1}$$

In this case agreement is only to two significant figures, primarily due to the modelling compromises that must be made with $1 / t$ at $t = 0$.

Example 2

Solution of the differential equation

$$\dot{x} = - x^2 t \qquad \text{(II.10)}$$

is

$$x = [\frac{1}{2} t^2 + (\frac{1}{x_0} - \frac{1}{2} t_0^2)]^{-1} \qquad \text{(II.11)}$$

The simulation of this problem was done on the basis of the following form of (II.10)

$$\dot{x} + (x . t) x = 0 \quad ,$$

where the nonlinear coefficient was represented by a nonlinear conductance Gl (or its corresponding element in the dual network) where, $Gl = x . t$. Taking $t_0 \simeq 0$ ($= 0.2 \times 10^{-7}$) , $T = \sqrt{2}$ and $x_0 = 1$, the hand computations yielded

$$x_0 = 1. \quad ,$$

$$x_T = 0.5 \quad ,$$

$$G = 0.375 \qquad \text{(II.12)}$$

The results of simulation as in Figure II.3 gave

$$x_0 = 1. \quad ,$$

$$x_T = 0.5 \quad ,$$

$$G = 0.3734 \; . \qquad \text{(II.13)}$$

Both results of the gradient are in agreement to within $< 0.4\ \%$. The representation of the nonlinear element G1 and R2 is done according to the relations

$$I_{G1} = G1\ (V_{R2})\ V_{G1}$$

$$V_{R2} = R_2\ (V_{01})\ I_{R2}$$

where 01 is an open circuit branch across the capacitor C and the independent source (initial condition) E1 .

Example 3

Consider the differential equation

$$\dot{x} = t / x \qquad \text{(II.14)}$$

Solving we get

$$x = [t^2 + (x_0^2 - t_0^2)]^{1/2} \qquad \text{(II.15)}$$

The simulation was done on the basis of

$$(x) \cdot \dot{x} - t = 0$$

where the derivative term was simulated by a nonlinear capacitor C (or inductor L in the dual network) where $c = x$.

Taking $t_0 = 1$, $T = 1$, $x(0) = 0$.

The hand computation gives

$$x_0 = 1.$$

$$x_T = 2.$$

$$G = -0.5 . \qquad \text{(II.16)}$$

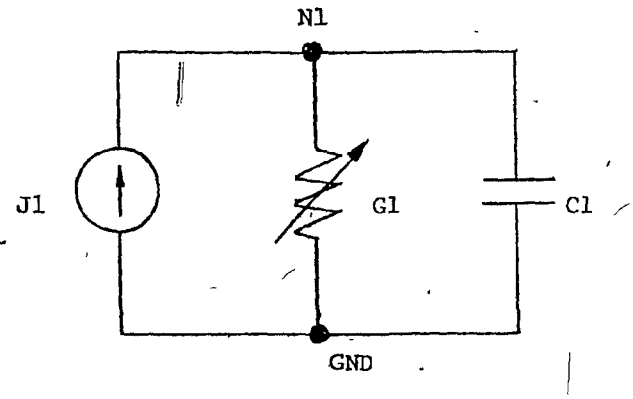
while the results of simulation as in Figure II.4 yielded

$$x_0 = 1.002$$

$$x_T = 2.003$$

$$G = -0.5009 \qquad \text{(II.17)}$$

The results for the gradient show an agreement between the simulation and the hand computations within three digits.



(a)

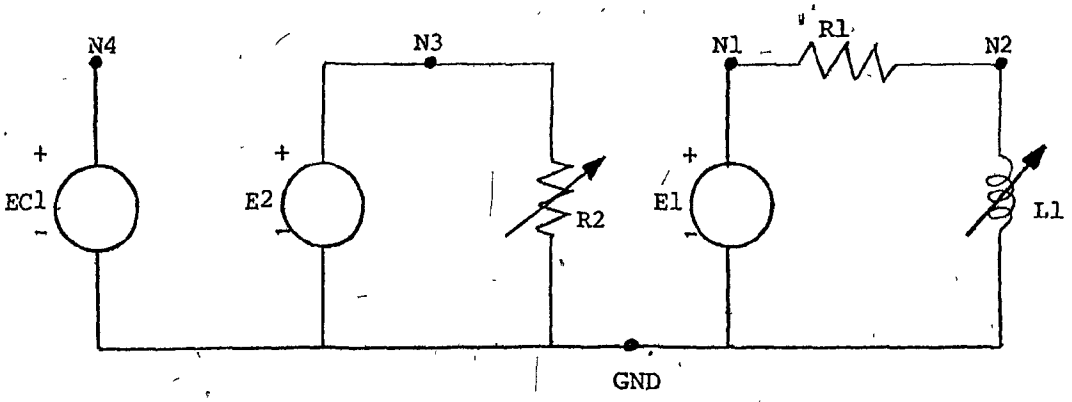
```
J1   GND   N1    TABLE   0.   0.   10.   10

G1   N1    GND   TABLE   V (G1)   I (J1)   0.   I.E - 8   10.   10

C1   N1    GND   1
```

(b)

Figure II.1. Simulation of equation $\dot{x} = -t(x-1)$ using nonlinear resistance.

(a) The simulating circuit.

(b) The input circuit description.



(a)

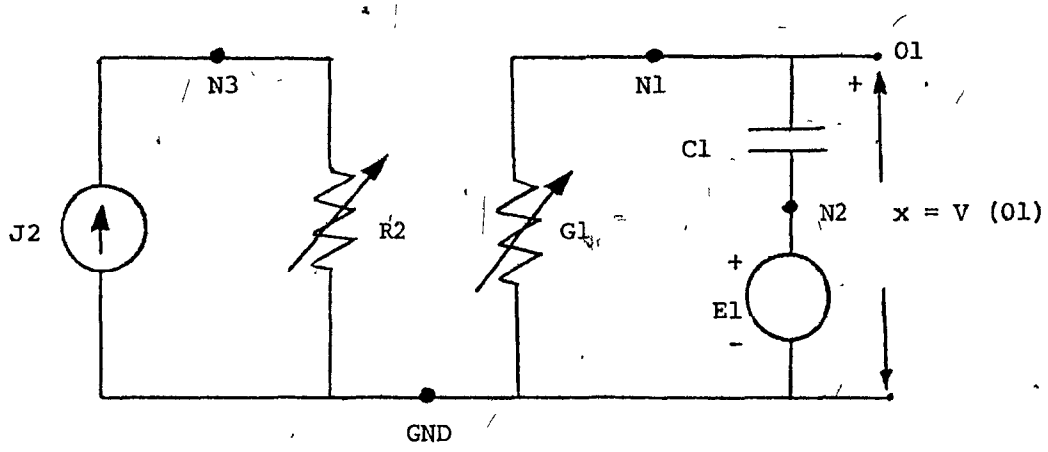
```
EC1    N4   GND    TABLE  0.   0.   10.   10.
E2     N3   GND    1.
R2     N3   GND    TABLE  I (R2)  V (EC1)  0 . 1.E-8.  10.  10.
E1     N1   GND    TABLE  0.   0.   1.E-7   1.
R1     N1   N2     1.
L1     N2   GND    TABLE  I(L1)  I (R2)   0.   0.   1.E8.  1.E8.
```

(b)

Figure II.2. Simulation of equation $\dot{x} = -t(x-1)$ using nonlinear reactive (L) circuit.

(a) Simulation circuit.

(b) Input description.



(a)

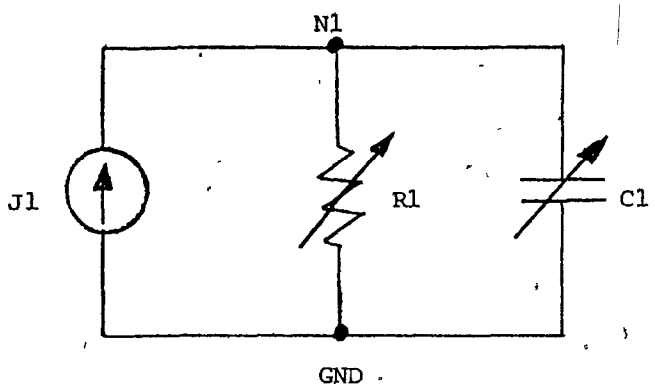
```
C1    N1    N2    1.
E1    N2    GND   TABLE   0.   0.   1.E-8   1 .
01    N1    GND .
G1    N1    GND   TABLE   V (G1)   V (R2)   0.   1.E-8.   10.   10.
R2    N3    GND   TABLE   I (R2)   V (01)   0.   0.   10.   10.
J2    GND   N3    TABLE   0.   0.   10.   10.
```

(b)

Figure II.3. Simulation of equation $\dot{x} = -x^2 t$.

(a) Simulation circuit.

(b) Circuit description.



(a)

```
J1    GND    N1    TABLE    0.    0.    10.    10.

R1    N1     GND   TABLE    0.    1.E5  1.E-6.  1.E10.

C1    N1     GND   TABLE    V (C1)  0.  1.E-5.  10.  10.
```

(b)

Figure II.4. Simulation of equation $\dot{x} = t / x$.

(a) Simulation circuit.

(b) Circuit description.

REFERENCES

[1] T.J. Aprille, Jr., and T.N. Trick, "Steady-State Analysis of Nonlinear Circuits with Periodic Inputs," Proceedings of the IEEE, vol. 60, pp. 108-114, January 1972.

[2] ——— , "A Computer Algorithm to Determine the Steady-State Response of Nonlinear Oscillators," IEEE Trans., Circuit Theory, vol. CT-19, pp. 354-360, July 1972.

[3] F.R. Colon, and T.N. Trick, "Fast Periodic Steady-State Analysis for Large-Signal Electronic Circuits," IEEE J. Solid-State Circuits, vol. SC-8, pp. 260-69, August 1973.

[4] T.N. Trick, F.R. Colon, and S.P. Fan, "Computation of Capacitor Voltage and Inductor Current Sensitivities with Respect to Initial Conditions for the Steady-State Analysis of Nonlinear Periodic Circuits," IEEE Trans. Circuits and Systems, vol. CAS-22, pp. 391 -396, May 1975.

[5] S.W. Director, A.J. Brodersen, and D.A. Wayne, "A Method for Quick Determination of the Periodic Steady-State in Nonlinear Networks," Proceedings of the Ninth Allerton Conference on Circuit and System Theory, pp. 131-139, October 1971.

[6] 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.

[7] R.K. Brayton, and S.W. Director, "Computation of Delay Time Sensitivities for Use in Time Domain Optimization," IEEE Trans. Circuits and Systems, vol. CAS-22, pp. 910-920, December 1975.

[8] S.W. Director, and K.W. Current, "Optimization of Forced Nonlinear Periodic Circuits," IEEE Trans. Circuits and Systems, vol. CAS-23, pp. 329-334, June 1976.

[9] K.W. Current, "Optimization of Nonlinear Periodic Circuits," Ph.D. Dissertation, University of Florida, December 1974.

[10] M.S. Nakhla and F.H. Branin, Jr., "Determining the Periodic Response of Nonlinear Systems by a Gradient Method," Int. J. Cir. Theor. Appl., vol. 5, no. 3, pp. 255-273, July 1977.

[11] M.S. Nakhla, "Steady-State Analysis of Nonlinear Periodic Systems," Ph.D. Dissertation, University of Waterloo, 1975.

[12] C. Brezinski, and A.C. Rieu, "The Solution of Systems of Equations Using the ε-Algorithm, and an Application to Boundary-Value Problems," Math. Comp., vol. 28, pp. 731-741, 1974.

[13] E. Gekeler, "On the Solution of Systems of Equations by the Epsilon Algorithm of Wynn," Math. Comp., vol. 26, pp. 427-436, 1972.

[14] J.B. McLeod, "A Note on the ε-Algorithm," Computing, vol. 7, pp. 17-24, 1971.

[15] P. Wynn, "Acceleration Techniques for Iterated Vector and Matrix Problems," Math. Comp., vol. 16, pp. 301-322, 1962.

[16] S. Skelboe, "Extrapolation Methods for Computation of the Periodic Steady-State Response of Nonlinear Circuits," 1977 IEEE Int. Symp. on Circuits and Systems, Pheonix, Arizona, pp. 64-67.

[17] G.D. Hachtel, R.K. Brayton, and F.G. Gustavson, "The Sparse Tableau Approach to Network Analysis and Design," IEEE Trans. Circuit Theory, vol. CT-18, pp. 101-113, January 1971.

[18] R.K. Brayton, F.G. Gustavson, and G.D. Hachtel, "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Backward Differentiation Formulas," Proc. IEEE, vol. 60, no.1, pp. 98, 108, January 1972.

[19] M.L. Blostein, and D.J. Millar, "SCAMPER: Information Manual," Electrical Engineering Report, McGill University, Montreal, Canada, 1976.

[20] ———— , "SCAMPER: User Manual," Electrical Engineering Report, McGill University, Montreal, Canada, 1976.

[21] M.L. Blostein, Private Communication.

[22] R. Fletcher, "New Approach to Variable Metric Algorithms," The Computer Journal, vol. 13, pp. 317-322, August 1970.

[23] M.J.D. Powell, "Restart Procedures for the Conjugate Gradient Method," Report C.S.S. 24 (A.E.R.A., Hartwell, November 1975).

[24] Carl E. Pearson, "Handbook of Applied Mathematics," Van Nostrand Reinhold, pp. 387-391, 1974.

[25] M.S. Nakhla, and J. Vlach, "A Piecewise Harmonic Balance Technique for Determination of Periodic Response of Nonlinear Systems," IEEE Trans. Circuits and Systems, vol. CAS-23, no. 2, pp. 85-91.

[26] R.S. Norin,and C. Pattle, "Effective Ordering of Sparse Matrices Arising from Nonlinear Networks," IEEE Trans. Circuit Theory, vol. CT-18, pp. 139-145, January 1971.

[27] C.A. Desoer and E.S. Kuh, "Basic Circuit Theory," vol. 2, New York : McGraw-Hill, pp. 292-300, 1967.

[28] C.W. Gear, "The Control of Parameters in the Automatic Integration of Ordinary Differential Equations," Department of Computer Science, University of Illinois, Int. Rep., May 1968.