DECOMPOSITION TECHNIQUES FOR LARGE SCALE CIRCUIT ANALYSIS AND SIMULATION

Giovanni De Micheli and Hsueh Y. Hsieh
IBM-T.J.Watson Research Center
Yorktown Heights, NY 10598

Ibrahim Hajj
Coordinated Science Laboratory
and the Department of Electrical and Computer Engineering
University of Illinois
Urbana Champaign, IL 61801

7.1 Introduction

Circuit analyzers, such as ASTAP [7.41] and SPICE[7.23], have proven to be essential tools for analyzing a variety of circuits. Such programs perform different kinds of analysis, such as *dc*, *ac* and transient analysis, and support a large variety of element models. With the advent of large scale integrated circuits, there has been an increasing demand for fast and reliable circuit analyzers. Since most large scale circuits are digital integrated circuits, there has been a wider interest in transient analysis. For this reason, several programs have been developed to perform mainly transient analysis and supporting primarily MOS technology devices.

General purpose circuit analyzers [7.41] [7.23] are based on Sparse Tableau Formulation [7.4] or the Modified Nodal Analysis [7.19] and on sparse Gaussian elimination. These techniques have been explained in detail in Part I of this book. It has been shown that the memory and computing time used by these programs grow super-linearly with the circuit size and realistically limit the programs to the domain of small to medium-sized circuits.

Gate and switch-level simulators [7.38] [7.3] [7.1] can provide first-order timing information more than three order of magnitude faster than circuit analyzers. However gate and switch-level simulators do not solve many problems inherent in integrated circuit design. Circuit designers are often interested in analyzing electrical waveforms with higher accuracy, for example to estimate critical-path delay or the effect of tightly-coupled feedback loops (e.g. memory sense amplifiers).

Several approaches have been followed to simulate the transient response of large scale circuits, while retaining a reasonable computational accuracy. A taxonomy of the major existing simulation techniques has been presented in [7.18], [7.26] and in [7.42]. We will consider in this chapter the transient analysis techniques that are based on incremental-time integration algorithms: i.e. the simulated waveforms are computed by discretizing the analysis time frame and by determining at each time-point the values of the requested circuit variables. In this respect, this chapter could be considered as an extension of the integration methods, described in Chapter 5 of Part I of this book, to the analysis of large scale circuits.

Circuit equations can be formulated as a system of a differential-algebraic equations:

$$f(\dot{x}, x, t) = 0 \tag{7.1.1}$$

$$x(0) = x_0$$

$$x \in R^n; \quad f(\cdot, \cdot, \cdot): R^n \times R^n \times R \rightarrow R^n$$

where x is a vector of n circuit variables. The transient analysis problem consists of integrating this set of equations over a time interval. The incremental-time numerical integration approach consist of discretizing the time interval into time-points t_k, $k = 1, 2, \dots, K$ and by replacing $\dot{x}$ at each time-point by an integration formula [7.14][7.7]. Here the Backward Differentiation Formulae [7.4] are used (see Chapter 5.3) , which approximate $\dot{x}(t_{k+1})$ as a function of x at the previous time-points. In particular:

$$\dot{x}_{k+1} = \frac{\sum_{p=0}^{P} \alpha_p x_{k+1-p}}{h} \tag{7.1.2}$$

where x_{k+1} is the computed value of $x(t_{k+1})$, $h \equiv t_{k+1} - t_k$ is the integration step-size at time t_k and P is the approximation order of the differentiation formula. The integration coefficient α_p are chosen so that $x_k = x(t_k)$ $\forall k = 1, 2, \dots, K$, when the solution $x(t_k)$ is a polynomial of degree P . For example the Backward Euler integration formula is obtained by choosing $P = 1$; $\alpha_0 = 1$; $\alpha_1 = 1$ and the second order Gear algorithm by setting: $P = 2$; $\alpha_0 = 3/2$; $\alpha_1 = -2$; $\alpha_2 = 1/2$ for fixed time steps.

By combining the formulae, the circuit equation can be expressed as:

$$g(x_{k+1}, t_{k+1}) = 0 \tag{7.1.3}$$

which is a set of nonlinear algebraic equations. This new set of equations must be solved at every time-step t_{k+1} for the computed solution x_{k+1} .

General purpose circuit analyzers solve the nonlinear algebraic system of equation by the Newton-Raphson algorithm at each time-point. The Newton-Raphson algorithm constructs a sequence of vectors x_{k+1}^j which eventually converges to x_{k+1} when the related assumptions are met. The vectors x_{k+1}^j are the solution of the set of linear equations:

$$A_{k+1}^j \, x_{k+1}^j = b_{k+1}^j \tag{7.1.4}$$

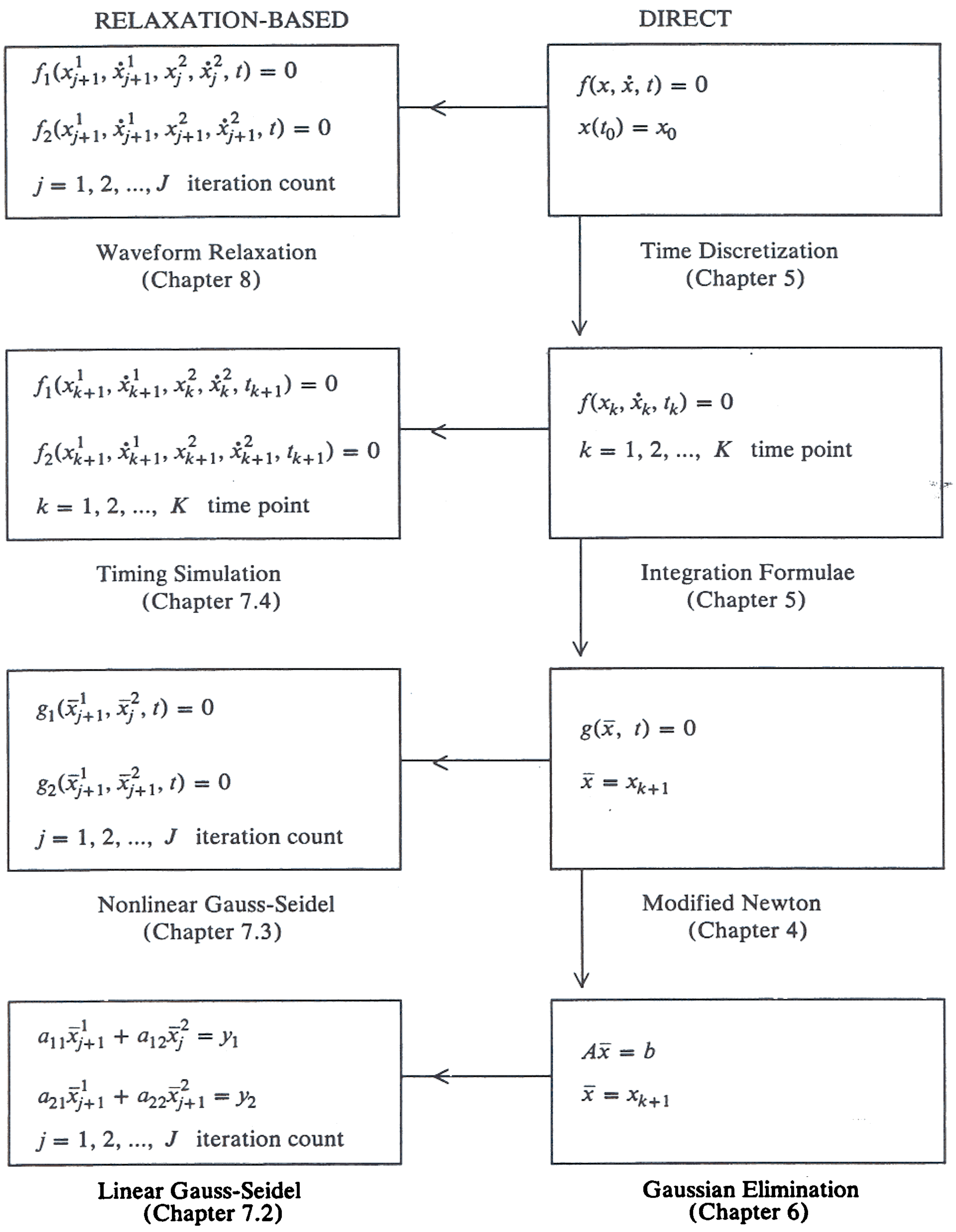
RELAXATION-BASED
DIRECT
$f_1(x_{j+1}^1, \dot{x}_{j+1}^1, x_j^2, \dot{x}_j^2, t) = 0$
$f_2(x_{j+1}^1, \dot{x}_{j+1}^1, x_{j+1}^2, \dot{x}_{j+1}^2, t) = 0$
$j = 1, 2, \ldots, J$ iteration count
$f(x, \dot{x}, t) = 0$
$x(t_0) = x_0$
Waveform Relaxation (Chapter 8)
Time Discretization (Chapter 5)
$f_1(x_{k+1}^1, \dot{x}_{k+1}^1, x_k^2, \dot{x}_k^2, t_{k+1}) = 0$
$f_2(x_{k+1}^1, \dot{x}_{k+1}^1, x_{k+1}^2, \dot{x}_{k+1}^2, t_{k+1}) = 0$
$k = 1, 2, \ldots, K$ time point
$f(x_k, \dot{x}_k, t_k) = 0$
$k = 1, 2, \ldots, K$ time point
Timing Simulation (Chapter 7.4)
Integration Formulae (Chapter 5)
$g_1(\bar{x}_{j+1}^1, \bar{x}_j^2, t) = 0$
$g_2(\bar{x}_{j+1}^1, \bar{x}_{j+1}^2, t) = 0$
$j = 1, 2, \ldots, J$ iteration count
$g(\bar{x}, t) = 0$
$\bar{x} = x_{k+1}$
Nonlinear Gauss-Seidel (Chapter 7.3)
Modified Newton (Chapter 4)
$a_{11}\bar{x}_{j+1}^1 + a_{12}\bar{x}_j^2 = y_1$
$a_{21}\bar{x}_{j+1}^1 + a_{22}\bar{x}_{j+1}^2 = y_2$
$j = 1, 2, \ldots, J$ iteration count
$A\bar{x} = b$
$\bar{x} = x_{k+1}$
Linear Gauss-Seidel (Chapter 7.2)
Gaussian Elimination (Chapter 6)

Figure 7.1.1 Circuit Formulations

where the coefficient matrix A_{k+1}^{j} and the vector b_{k+1}^{j} are related to the jth Newton-Raphson iteration at the time-point t_{k+1} [7.27]. Hence, numerical integration and the Newton-Raphson technique reduces the problem to the solution of a system of linear equations. This is the fundamental limitation of general purpose analyzers for large scale circuits.

In this chapter we present some techniques that exploit the structure of large scale circuits and enable time-effective integration of the circuit equations, while retaining considerable accuracy of the solution. Most of the techniques in this chapter are based on *decomposition* . Decomposition techniques are used to partition the circuit equations according to their structure. Smaller sets of equations are analyzed more efficiently due to their smaller size. Under appropriate assumptions, the values of the electrical variables of the entire circuit can be computed with controllable accuracy by knowing the values of the variables in the components.

The chapter is organized in a logical perspective, by presenting decomposition techniques at different levels of the solution of the circuit equations. First decomposition and solution of linear system of equations are presented. Then, the decomposition of nonlinear system of equations is described. Eventually decomposition at the level of the integration formulae is reported. This will help the reader to understand the relations between different techniques and bridging the gap between these techniques and waveform relaxation (Chapter 8) in which decomposition is applied directly on the system of differential equations. In fact this chapter and Chapter 8 are closely related, since they both deal with solving the same problem. However the logical perspective does not correspond to the chronological order in which these techniques have been discovered and implemented. Fig. 7.1.1 shows the different levels of the solution of the circuit equations and the use of decomposition and relaxation methods at each of these levels.

7.2 Decomposition at the linear equation solution level

In this section we consider the solution of (7.1.4), which is the last level in Fig. 7.1, by decomposition techniques. For convenience, the superscripts and the subscripts are dropped and the equations are written as:

$$Ax = b \tag{7.2.1}$$

The matrix A is sparse for the circuit applications we consider here. Sparse matrix techniques, as described in Chapter 6 are widely used in solving (7.2.1). For large-scale circuits, however, sparse matrix techniques alone are not cost-effective and decomposition techniques become necessary for reducing computation time.

As mentioned above, the solution of circuit equations by decomposition is an approach by which the circuit is partitioned or torn into an interconnection of subcircuits. The subcircuits are then solved separately (in parallel, if parallel processing is available) or in a predetermined sequence; their solutions are then combined together to obtain the solution of the entire circuit.

From the algebraic point of view, decomposition can be considered as a *reordering* of the circuit in a special way. It can be shown that the amount of computation required in solving linear equations by decomposition is in many cases greater than that needed for sparse matrix techniques to solve the circuit equations without decomposition. Nevertheless, there are many situations where decomposition becomes advantageous, if not necessary. This is true, for example, when a circuit is so large that its equations cannot be stored on an available computer even though sparse matrix solution techniques are being used. Decomposition and overlay schemes thus become a necessity. Decomposition techniques can also be employed in cases where the circuit contains repetitive identical subcircuits so that the equations of only a few subcircuits need to be stored. Further, decomposition also allows the application of parallel processing where the computation time could be reduced, although the amount of computation may increase. As we shall see below, decomposition becomes more effective when solving nonlinear circuits, where the concept of *latency* is used to reduce computation.

There are many ways of reordering or decomposing a system of equations, depending on the solution method to be used. Here, we will consider two types of matrix decomposition which have been employed in large-scale integrated circuit simulation. One is the bordered-block diagonal form (BBD) and the other is the 'nearly' lower-block triangular form (NLBT). We will also consider two general classes of solution methods: direct methods and indirect or relaxation methods. As will become clear in the sequel, the BBD form is employed to solve the circuit equations by direct methods, such as in SLATE [7.44], while the NLBT form is suitable for relaxation methods.

7.2.1 Bordered-block diagonal form

Consider a linear circuit which is to be analyzed by decomposing or partitioning its matrix into a bordered-block diagonal form. Let the circuit equations be constructed using a general formulation approach, such as the tableau approach [7.17] or the modified nodal approach [7.19] (see also Chapter 2 above). As mentioned above, decomposition can be viewed as ordering the circuit equations in a special way. From the circuit point of view, decomposition can be accomplished by removing or 'tearing' a set of branches [7.43] or nodes [7.36]. If the tearing set consists of branches, such that there is no coupling among the torn subcircuits and between the subcircuits and the tearing branches, and also if the currents in the tearing branches are declared as circuit variables, while each subcircuit variables are clustered together, then the partitioned equations will have the following form:

$$\begin{bmatrix} M_1 & & & & Y_{10} & A_1 \\ & M_2 & & & Y_{20} & A_2 \\ & & & & & \\ & & & M_m & Y_{m0} & A_m \\ Y_{01}^T & Y_{02}^T & & Y_{0m}^T & Y_{00} & A_t \\ A_1^T & A_2^T & & A_m^T & A_t^T & -Z_t \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_m \\ v_0 \\ i_t \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \\ b_m \\ j_0 \\ e_t \end{bmatrix} \qquad (7.2.2)$$

where x_i is an n_i-vector containing node-to-global datum voltages of subcircuit N_i together with possibly a subset of subcircuit branch currents, and b_i is the vector of sources in the subcircuit. The vector v_0 is the set of *local* datum nodes voltages, where each local datum node is selected from each floating subcircuit that is created by removing the tearing branches. The reason for ordering v_0 last with i_t is to prevent the block-diagonal matrices corresponding to the floating subcircuit from being singular. More formally, it can be shown that [7.43]: $Y_{00} = \sum_{i=1}^{m} Y_{0i}^T M_i^{-1} Y_{i0}$, where M_i is an $n_i x n_i$ nonsingular matrix, and Y_{i0} and Y_{0i} each contain at most one nonzero column. A_i is a topological matrix which contains exactly b_{ti} nonzero columns and n_{ti} nonzero rows, where b_{ti} is the number of tearing branches incident with subcircuit n_i at n_{ti} nodes other than the local datum node.

If the tearing set consists of node voltages only, such that no coupling exists neither among the torn subcircuits nor between the subcircuits and the tearing nodes, the circuit equations will have the following form:[1]

$$\begin{bmatrix} M_1 & & & & Y_{1c} \\ & M_2 & & & Y_{2c} \\ & & & & \vdots \\ & & & M_m & Y_{mc} \\ Y_{c1}^T & Y_{c2}^T & & Y_{cm}^T & Y_{cc} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \\ v_c \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \\ j_c \end{bmatrix} \qquad (7.2.3)$$

where Y_{ic} and Y_{ci} each now contains exactly n_{ci} nonzero columns; n_{ci} is the number of tearing nodes connected to subcircuits n_i ; v_c is the set of node-to-datum voltages at the tearing nodes. Note that in (2.3) if the entire circuit equation is nonsingular, $Y_{cc} = \sum_{i=1}^{m} Y_{ci}^T M_i^{-1} Y_{ic}$ will also be nonsingular. For easy reference we denote the partitioned form in (7.2.2) as branch tearing (BT) and in (7.2.3) as node tearing (NT). Also, the vertical and horizontal border submatrices in (7.2.2) and (7.2.3) is denoted by P_i and Q_i,

As long as there is no coupling among the subcircuits themselves, it is possible to have coupling between the subcircuits and the tearing variables, and (7.2.2) and (7.2.3) would still have Bordered Block Diagonal form. For simplicity, it is assumed that no coupling exists between the subcircuits and the tearing set.

respectively; i.e., $P_i = [Y_{io} \mid A_i]$ or $P_i = Y_{ic}$ and $Q_i = [Y_{oi} \quad A_i]$ or $Q_i = Y_{ci}$. The tearing matrix is $M_t = \begin{bmatrix} Y_{00} & A_t \\ A_t^T & -Z_t \end{bmatrix}$ or $M_t = Y_{cc}$.

The partitioning of a circuit matrix into BBD form can be done either manually, such as implied by nested models and subcircuits of input data description [7.44], or automatically. A heuristic algorithm for automatically decomposing a circuit matrix into BBD form based on node tearing is described in [7.37]. The algorithm produces blocks of 'nearly' equal sizes while minimizing the number of variables in the border. The algorithm constructs a dependency matrix which represents the zero-nonzero pattern of the coefficients of A, which is assumed to be structurally symmetric, and thus is represented by an undirected graph G. Three disjoint sets of nodes are then formed iteratively: Z is a set of nodes which are candidates to form a block of the decomposition, S is a set of nodes called the *separator* of Z with respect to G.; and W is the remainder of the nodes of G. The algorithm then determines a sequence of nodes of G that are added to Z. For each node in the sequence the cardinality of Z, $|Z|$, is increased by one and the separator S is updated. Let Z_j, be the jth candidate block in the sequence of length k such that $|Z_k| = n_{\max}$, where $n_{\max}$ is a preassigned maximum block size. Then Z_j is selected as a block which satisfies $n_{\min} \leq |Z_j| \leq n_{\max}$ such that $|S_j|$ is minimum, where $n_{\min}$ is a pre-assigned minimum block size Z_j and S_j are then both removed from G and the process repeated. A greedy algorithm is used in extending the sequence at every step, where the node selected to be added to Z is one which causes the increase in S to be minimum, preferably a negative increase. The success of the algorithm depends on the initial node chosen. The approach can also be applied to weighted cluster problems where a set of k nodes can be preassigned to be in one block by collapsing them into one 'super' node in G with weight equal to k.

7.2.2 Direct solution algorithms for BBD systems

This section describes how the sets of linear equations 7.2.2 and 7.2.3 can be solved by direct methods. Let systems (7.2.2) and (7.2.3) be written in the form:

$$\begin{bmatrix} M & P \\ Q^T & M_t \end{bmatrix} \begin{bmatrix} x \\ x_t \end{bmatrix} = \begin{bmatrix} b \\ b_t \end{bmatrix} \tag{7.2.4}$$

System (7.2.4) may be solved using one of three factorization procedures denoted by F_1, F_2 and F_3 and corresponding substitution procedures S_1, S_2 and S_3, respectively [7.15] [7.16]. The procedures are described in the algorithms below in the frames.

Note that the difference between the algorithms is in the solving of the subcircuit equations. The step that forms $\hat{M}_t$ is the same in all the factorization algorithms. It represents the solution of an interconnection circuit formed by *collapsing* or forming the n-port description of the subcircuits at the interconnection terminals.

```
ALGORITHM  Factorization $F_1$ :

BEGIN
  FOR  i := 1  TO  m  DO
  BEGIN
     $\begin{bmatrix} M_i & P_i \\ Q_i^T & O \end{bmatrix} \xrightarrow[\text{Factorization}]{\text{Partial LU}} \begin{bmatrix} L_iU_i & V_i \\ W_i^T & R_i \end{bmatrix}$
     where $V_i = L_i^{-1}P_i,\ W_i^T = Q_i^TU_i^{-1},\ R_i = -W_i^TV_i$
  END;
  Form        $\hat{M}_t = M_t + \sum_{i=1}^{m} R_i$
  Factorize   $\hat{M}_t = L_tU_t$
END.
```

```
ALGORITHM  Substitution $S_1$ :

BEGIN
  FOR  i := 1  TO  m  DO
  BEGIN
     $L_i\, a_i = b_i$
     $\bar{b}_i = W_i^T a_i$
  END;
  $L_t\, y_t = b_t - \sum_{i=1}^{m} \bar{b}_i$
  $U_t\, x_t = y_t$
  FOR  i := 1  TO  m  DO
  BEGIN
     $U_i\, x_i = a_i - V_i x_t$
  END;
END.
```

ALGORITHM *Factorization* F_2 **:**

```
BEGIN
  FOR  $i$ := 1  TO  $m$  DO
  BEGIN
    $\begin{bmatrix} M_i & P_i \\ Q_i^T & O \end{bmatrix} \xrightarrow[\text{Factorization}]{\text{Partial LU}} \begin{bmatrix} L_iU_i & \overline{V}_i \\ Q_i^T & R_i \end{bmatrix}$
    where $\overline{V}_i = U_i^{-1}V_i = U_i^{-1}L_i^{-1}P_i\,,\quad R_i = -Q_i^T\overline{V}_i$
  END;
  Form        $\hat{M}_t = M_t + \sum_{i=1}^{m} R_i$
  Factorize   $\hat{M}_t = L_tU_t$
END.
```

ALGORITHM *Substitution* S_2 **:**

```
BEGIN
  FOR  $i$ := 1  TO  $m$  DO
  BEGIN
    $L_i\, a_i = b_i$
    $U_i\, z_i = a_i$
    $\overline{b}_i = Q_i^T z_i$
  END;
  $L_t\, y_t = b_t - \sum_{i=1}^{m} \overline{b}_i$
  $U_t\, x_t = y_t$
  FOR  $i$ := 1  TO  $m$  DO
  BEGIN
    $x_i = z_i - \overline{V}_i\, x_t$
  END;
END.
```

ALGORITHM *Factorization* F_3 :

```
BEGIN
  FOR  i := 1  TO  m  DO
  BEGIN
    $\begin{bmatrix} M_i & P_i \\ Q_i^T & O \end{bmatrix} \xrightarrow[\text{Factorization}]{\text{Partial LU}} \begin{bmatrix} L_iU_i & P_i \\ \overline{W}_i^T & R_i \end{bmatrix}$
    where $\overline{W}_i^T = W_i^T L_i^{-1} = Q_i^T U_i^{-1} L_i^{-1}$ , $R_i = -\overline{W}_i^T P_i$
  END;
  Form        $\hat{M}_t = M_t + \sum_{i=1}^{m} R_i$
  Factorize   $\hat{M}_t = L_t U_t$
END.
```

ALGORITHM *Substitution* S_3 :

```
BEGIN
  FOR  i := 1  TO  m  DO
  BEGIN
    $\overline{b}_i = \overline{W}_i^T b_i$
  END;
  $L_t y_t = b_t - \sum_{i=1}^{m} \overline{b}_i$
  $U_t x_t = y_t$
  FOR  i := 1  TO  m  DO
  BEGIN
    $L_i y_i = b_i - P_i x_t$
    $U_i x_i = y_i$
  END;
END.
```

The structure (zero-nonzero pattern) of the matrix $\hat{M}_t$ can be determined ***a priori*** by forming an interconnection circuit as described in [7.16]. In general, when the number of subcicuits is large, the matrix $\hat{M}_t$ becomes large but sparse since the number of interconnections between each subcircuit and the rest of the system is relatively small when compared to the total number of interconnections, which means that P_i and Q_i^T are sparse. Sparsity consideration in implementing the above solution algorithms are studied in detail in [7.16]; factorization F_1 and substitution S_1 are implemented in SLATE [7.44]

7.2.3 Indirect solution methods and NLBT systems

In the indirect or relaxation methods the LU factorization is replaced by an iterative procedure where A is decomposed into a sum of matrices:

$$A = L + D + U \tag{7.2.5}$$

where L is a strictly lower triangular matrix and U a strictly upper triangular matrix and D a point- or block-diagonal matrix. Eq. (7.2.5) is then solved iteratively using either Gauss-Jacobi or Gauss-Seidel method. For the **Gauss-Jacobi** algorithm, the iteration can be written in matrix form:

$$Dx^{k+1} = b - (L + U)x^k; \tag{7.2.6a}$$

where x^{k+1} is the approximation of x produced at the k^{th} relaxation iteration. We write Eq. 7.2.6a as

$$\begin{aligned} x^{k+1} &= D^{-1}b - D^{-1}(L + U)x^k \\ &= D^{-1}b + M_{GJ}x^k \end{aligned} \tag{7.2.6b}$$

where M_{GJ}, the Gauss-Jacobi *companion matrix*, is defined as $M_{GJ} = -D^{-1}(L + U)$. Similarly, the **Gauss-Seidel** algorithm can be written in matrix form:

$$(L + D)x^{k+1} = b - Ux^k; \tag{7.2.7a}$$

or equivalently

$$\begin{aligned} x^{k+1} &= (L + D)^{-1}b - (L + D)^{-1}Ux^k \\ &= (L + D)^{-1}b + M_{GS}x^k \end{aligned} \tag{7.2.7b}$$

where $M_{GS} = -(L + D)^{-1}U$ is the companion matrix of the method. In both (7.2.6) and (7.2.7), D is assumed to be nonsingular, otherwise, the iterations are not defined. If D is diagonal, then the methods are referred to as point Gauss-Jacobi and point Gauss-Seidel; if D is block-diagonal, they are referred to as block Gauss-Jacobi and block Gauss-Seidel. The following theorems state conditions for convergence of the iterations.

Theorem 7.2.1

The necessary and sufficient condition for the iterates in (7.2.6) and (7.2.7) to converge to a solution for any initial guess is that the spectral radii of M_{GJ} and M_{GS} are less than one; i.e., all the eigenvalues are inside the unit disk in the complex plane. ■

Theorem 7.2.2

Sufficient conditions for Gauss-Jacobi and Gauss-Seidel both point and block methods to converge is that A be strictly diagonally dominant or be an M-matrix [7.40]. ■

Gauss-Seidel and Gauss-Jacobi methods can be applied to any system of equations as long as the matrix D in (7.2.5) is nonsingular and convergence conditions are satisfied. However, as can be seen from (7.2.6) and (7.2.7), convergence, as well as the speed of convergence, depends on the ordering of the equations and variables, i.e., on the ordering of the rows and columns of A. If A has a BBD form, indirect solution methods can be applied by relaxing the tearing variables. In many cases, for the same row and column ordering, the Gauss-Seidel method converges faster than the Gauss-Jacobi; but not always. One obvious case where the Gauss-Seidel method converges faster than the Gauss-Jacobi is when A can be permuted into a lower triangular matrix so that U is identically zero. In this case the Gauss-Seidel method converges in one iteration, but not the Gauss-Jacobi. In general, the speed of convergence of the Gauss-Seidel method improves if A is permuted into a nearly lower triangular or block triangular form (NLBT). Algorithms for permuting the circuit equations into NLBT form are essentially the same ones used when applying Gauss-Seidel methods at the nonlinear solution level, which are discussed in the next section, as well as in the waveform relaxation techniques described in Chapter 8. These algorithms are referred to as analysis sequencing, selective-trace or event-driven procedures.

7.3 Decomposition at the nonlinear equation solution level

7.3.1 Direct solution methods

A number of different strategies are used to achieve decomposition at the nonlinear equation level. The first approach we consider is to solve (7.1.3) at each time-point by direct methods (Newton's method or modifications of Newton's method) and to rely on BBD decomposition of (7.1.4). Thus the numerical properties of the integration formula used to discretize the differential equations as well as the quadratic convergence properties of the Newton's method are retained. Such an approach has been used in SLATE [7.44].

If the entire linearized circuit equations are to be solved at every iteration, then decomposition techniques in general do not provide any reduction in the computational effort as compared to a straight-forward sparse matrix solution technique without decomposition. In practice, however, particularly in digital circuits, a large portion of the circuit is inactive or *latent* at any given time. Latency means inactive or 'not changing', which could be numerically not changing during the Newton iterations at each time point (spatial) or over a period of time (temporal). Latency can be exploited to reduce computation when applying direct solution methods as well as indirect methods. It can be detected at three levels: device level, subcircuit level and interconnection level. By using decomposition and by exploiting latency, only a portion of the circuit equations have to be formulated and solved at an iteration point, thus reducing the computational effort and justifying the additional overhead required by decomposition methods.

At the device level latency is sometimes referred to as a bypass scheme. This scheme is done by monitoring the operating point of each nonlinear device. If the operating point remains unchanged, to within a prescribed tolerance, from one iteration to the next, the device characteristic equations are not re-evaluated, and the matrix entries at the previous iteration are used again. This bypass scheme may, of course, be applied when analyzing the circuit without decomposition such as in SPICE [7.23].

Latency at the subcircuit and the interconnection levels depend on the decomposition. Without loss of generality, we assume node tearing is being used. As mentioned above in solving the circuit equations in BBD form, when all the variables of a subcircuit are eliminated, a generalized Norton equivalent of the linearized subcircuit equations is generated. By combining the equivalent circuits of all the subcircuits with the rest of the circuit, the interconnection circuit is obtained (Step 2 of the factorization and solution algorithms of the previous section). If the values of all the subcircuit variables remain unchanged, to within a prescribed tolerance, the equivalent circuit of the subcircuit as seen by the external nodes remains unchanged, and the subcircuit is declared latent during the Newton iteration at that particular time point. This condition usually occurs when subcircuits converge before the others. This latency state remains in effect until a *significant* change is detected in the values of the external subcircuit variables.

The latency criteria employed in SLATE [7.44] are as follows. Consider a subcircuit N_k Let $v_{t\,sub\,k}$ denote the tearing node voltages of N_K, v_{sk} its internal node voltages, and v_{nlk} the voltages across the nonlinear elements of N_k Subcircuit N_k is declared latent at the ith iteration during the Newton-Raphson iteration loop if the following two conditions are satisfied:

$$(1) \quad v_{nlk_m}(i-1) - v_{nlk_m}(i-2) \leq \epsilon_a + \epsilon_r \max \left(\; v_{nlk_m}(i-1) \quad | v_{nlk_m}(i-2) | \right)$$

for $m = 1,2,$

$$(2) \quad v_{t_{km}}(i) - v_{t_{km}}(i-1) \leq \epsilon_a + \epsilon_r \max \left(| v_{t_{km}}(i) | \quad v_{t_{km}}(i-1) \; \right)$$

for $m = 1, 2, \dots , s$, where ϵ_a and ϵ_r are absolute and relative error constant, r and s respectively the number of nonlinear elements and tearing nodes in N_k. Subcircuit N_k will remain latent as long as:

$$v_{t_{km}}(i+j) - v_{t_{km}}(i-1) \quad \leq \epsilon_a + \epsilon_r \ \max \ (\ v_{t_{km}}(i+j) \qquad | \ v_{t_{km}}(i-1) \ | \ ),$$

for $m = 1, 2, \dots , s$, where j is an iteration number increment.

Once a subcircuit is declared latent during the Newton-Raphson iteration, then the linearization of the equations of the nonlinear element in N_k, the processing of the subcircuit matrix to obtain the partial contribution to the interconnection matrix, backward substitution to solve for the internal circuit variables, and convergence tests related to N_k, are all bypassed. It is only necessary to monitor the tearing node voltages to check when subcircuit N_k becomes active.

For latency in time, a subcircuit N_k is considered latent at time t_n if the following conditions are satisfied:

$$(1) \quad | \ v_{t_{km}}(t_n) - v_{t_{km}}(t_{n-1}) \ | \leq \epsilon_a + \epsilon_r \ \max \ (\ | \ v_{t_{km}}(t_n) \qquad | \ v_{t_{km}}(t_{n-1}) \ | \ , \quad m = 1, 2, \dots s$$

$$I_{k_m}(t_n) - I_{k_m}(t_{n-1}) \, | \ \leq \epsilon_c + \epsilon_r \ \max \ (\, | \, I_{k_m}(t_n) \, | \quad I_{k_m}(t_{n-1}) \, | \,), \quad m = 1, 2, \dots, b$$

$$(3) \ h_{n-1} \ \frac{| \ I_{k_m}(t_n) - I_{k_m}(t_{n-1}) \ |}{' \, Q_{k_m}(t_n) - Q_{k_m}(t_{n-1}) \, |} \quad \geq 1$$

provided $| \, I_{k_m}(t_n) - I_{k_m}(t_{n-1}) \quad \geq \epsilon$, ϵ is a very small constant chosen to be 10^{-12})

Where $I_k = (I_{k_1}, I_{k_2}, \dots, I_{k_b})$ is the set of capacitor currents and inductor voltages, $Q_k = (Q_{k_1}, Q_{k_2}, \dots, Q_{k_b})$ is the set of capacitor charges and inductor fluxes in N_k. Subcircuit N_k remains latent in time as long as

$$(4) \quad | \ v_{t_{km}}(t_{n+j}) - v_{t_{km}}(t_{n-1}) \ | \quad \leq \epsilon_a + \epsilon_r \ \max \ (\ | \ v_{t_{km}}(t_{n+j}) \ | \ , \quad | \ v_{t_{km}}(t_{n-1}) \ | \ )$$

for $m = 1, 2, \dots , s$. Condition (3) is used to avoid declaring a circuit with a slowly varying response to be latent when it is not. The condition is derived by analyzing linear RC circuit and relating the response to the circuit time constant and the capacitor current and charge. The reason for including condition (4) is that in applying a direct solution method all the subcircuits are solved with the same timestep, which is determined by the subcircuit with the fastest response. An alternative approach would be to let each subcircuit be analyzed with its own timestep control. But this would require waveform interpolation and extrapolation, which in turn would require some form of relaxation techniques, and thus would make the solution method indirect rather than direct.

7.3.2 Hierarchical Decomposition and Multilevel-Newton Algorithm.

The above decomposition technique can be extended to obtain *hierarchical* decomposition, where each subcircuit is further decomposed into an interconnection of smaller subcircuits, thus producing a hierarchy of decomposition levels. The hierarchy can be specified by the designer in the input-language specification to the analysis and take advantage of repetitive units (e.g. generalized logic gates) in a circuit. In general, each subcircuit interacts with the rest of the circuit at the higher level of hierarchy only at few terminal nodes.

A *numerical macromodel* of a subcircuit consists of a set of algebraic-differential equations simulating the input/output behavior of the subcircuit at the terminal nodes:

$$h(x, y, u) = 0 \qquad (7.3.1)$$

where x represents the variables internal to the macromodel, y and u are the variables related to the terminal nodes. Note that in (7.3.1) x, y and u represent a partition of the variable vector x of (7.1.3). We assume here that given u the interaction of the subcircuit is completely specified by y. If (7.3.1) has one and only one solution for each u, we denote the I/O representation:

$$y = g_y(u) \qquad (7.3.2)$$

an *exact macromodel*

By using the macromodel representation, a circuit can be described as either a set of circuit elements or an interconnection of subcircuits and numerical macromodels. Hierarchical representations have shown to be effective to manage the complexity of large-scale system analysis [7.31]. Ruehli et al. surveyed macromodelling techniques in [7.32] and [7.29]. Macromodelling techniques have been used in program MACRO [7.30] [7.33] and MEDUSA [7.12] among others.

We consider now hierarchical decomposition and macromodelling in conjunction with the solution of (7.1.3). For the sake of simplicity, let us represent (7.1.3) as

$$g(x) = 0 \qquad (7.1.3bis)$$

and consider a circuit with a 2-level hierarchy and only one macromodel (Figure 7.3.1).

In the hierarchy, there is an "upper" circuit which is represented by:

$$g(u, y, w) \quad 0 \qquad (7.3.3)$$

where $u, y,$ and , w are blocks of a partition of x of (7.1.3bis) so that u and y represent the circuit variables that interact with the macromodel and w those that do not interact with it. Furthermore a "lower" subcircuit is represented by a macromodel (7.3.1).

Rabbat, Sangiovanni and Hsieh introduced a *multi-level Newton algorithm* [7.31] for the solution of (7.3.3) and (7.3.1), that take advantage of the hierarchical representation. This algorithm is especially suited for large scale circuits.

The main concept of the multilevel Newton algorithm is to use the hierarchical representation to handle large-scale circuits. There are as many Newton loops as the levels. In our case, we have two levels and two loops: an "outer" loop to solve (7.3.3) and an "inner" loop to solve for (7.3.1). The outer Newton loop approximates the solution of $g(u, g_y(u), w) = 0$ by solving:

$$\left(\frac{\partial g}{\partial u} + \frac{\partial g}{\partial y}\frac{dg_y}{du}\right)\Delta u + \frac{\partial g}{\partial w}\Delta w + g = 0 \qquad (7.3.4)$$

where $g = g(u^i, g_y(u^i), w^i)$; $\Delta u \equiv u^{i+1} - u^i$; $\Delta w \equiv w^{i+1} - w^i$ and i is the "outer" loop iteration count. To evaluate (7.3.4) at each iteration i of the "outer" loop, we need to know $g_y(u)$ and $\frac{dg_y}{du}$. This is done using a second, "inner", Newton loop on the macromodel equations:

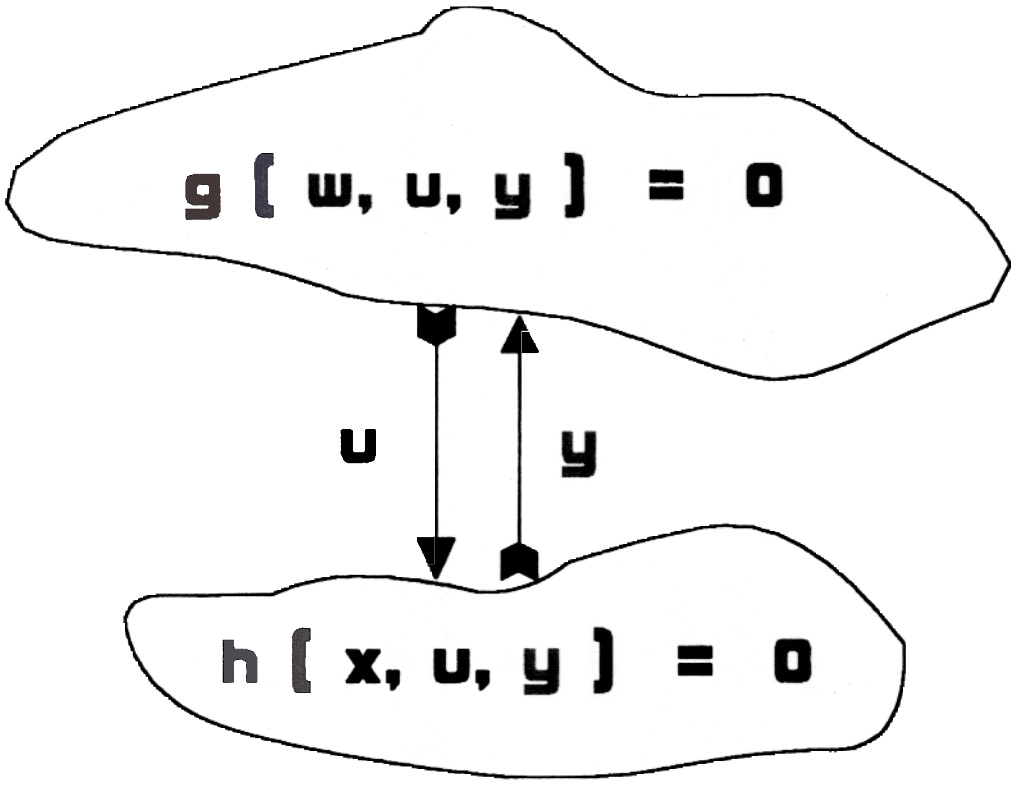

Figure 7.3.1 Hierarchical Circuit Model

$$\frac{\partial h}{\partial x}\Delta x + \frac{\partial h}{\partial y}\Delta y + h = 0 \qquad (7.3.5)$$

where $h = h(u^i, x^k, y^k)$; $\Delta x \equiv x^{k+1} - x^k$; $\Delta y \equiv y^{k+1} - y^k$ and k is the "inner" loop iteration count. The second Newton iteration is termed "inner" because u^i is determined by (7.3.4) and held fixed in (7.3.5). If (7.3.1) is solved exactly, then (7.3.4) is a true Newton iteration with local quadratic convergence. However, if the solution of (7.3.1) is approximated, the question of quadratic convergence is open. Rabbat et al. proved in [7.31] that by defining an appropriate stopping criterion for the "inner" loop, the multilevel Newton algorithm retains quadratic convergence rate.

The algorithm is described in detail in the frame. The parameters τ^i and η control the inner and outer loop convergence respectively. In the "inner" loop we solve the nonlinear equation $h(u^i, x^{i,k}, y^{i,k}) = 0$ for x and y with u^i as a known vector.

In the "outer" loop $g_y(u)$is approximated to $y(u)$ and the Jacobian $\frac{dg_y}{du}$ is approximated using the implicit function theorem. The approximation arises from the fact that (7.3.1) is in general not solved exactly.

The convergence of the "outer" loop is driven by parameter η usually set to the typical value of 10^{-4}. If the "outer" loop has converged, the algorithm stops. Else we compute dynamically the "inner" loop convergence tolerance τ^{i+1} for the next inner loop iteration. It should be noted that γ is a parameter which affects the convergence property of MLNA and is typically between 1 and 2. The overall rate of convergence of the algorithm is at least quadratic, if $\gamma = 2$

Theorem 7.3.1

Let $(\hat{u}, \hat{w})$ be such that $g(\hat{u}, g_y(\hat{u}), \hat{w}) = 0$. Assume that :

1) g is Lipschitz continuously differentiable;

2) $J(\hat{u}, \hat{w})^{-1}$ exists, where $J(u,w) = \left(\frac{\partial g}{\partial u} + \frac{\partial g}{\partial y}\cdot\frac{dg_y}{du}\ ,\ \frac{\partial g}{\partial w}\right)$

3) $h(x,y,u)$ is Lipschitz continuously differentiable and the Jacobian of h is uniformly bounded on the set $\Omega \equiv \{(x,y,u) \mid h(x,y,u) = 0\}$. The Jacobian with respect to x,y is nonsingular in Ω and its inverse is uniformly bounded in Ω

4) $\forall i$, $\exists$ x,y such that $h(u^i, x, y) = 0$

5) $\forall i$, $(x^{i,0}, y^{i,0})$ are such that the "inner" Newton loop converges.

then: *exists delta gt*0 such that $\forall (u^0, w^0) \in B((\hat{u}, \hat{w}), \delta)$; $\forall \tau^0 \in [0, \delta]$, the multi-level Newton algorithm converges to $(\hat{u}, \hat{w})$ with root convergence order greater than or equal to two.■

The proof of the theorem is reported in [7.31].

To achieve quadratic convergence, it is crucial to drive τ to zero as fast as $\| (\Delta u, \Delta w) \|^2$. The convergence rate of MLNA for the case $\gamma < 2$ has been investigated by Lin [7.22]. In particular it has been shown that convergence is pairwise quadratic [7.22]. Although $\gamma \geq 2$ is crucial to insure a quadratic convergence theoretically, the use of $\gamma < 2$ does not necessarily imply a slower convergence rate in practice.

```
ALGORITHM    (Multi-level Newton)

Parameter: $\tau^0; \eta; \gamma$

BEGIN
  Initialization of the upper level Newton algorithm
  $i = 0$
  Guess $u^0; w^0$.
  REPEAT                                        (* outer loop *)
    Initialization of the lower level Newton algorithm
    $k = 0$
    Guess $x^{i,0}; y^{i,0}$.
    REPEAT                                      (* inner loop *)
      Compute $(x^{i,k+1} y^{i,k}) = (x^{i,k} y^{i,k}) + (\Delta x, \Delta y)$ by solving:
      $\frac{\partial h}{\partial x} \Delta x + \frac{\partial h}{\partial y} \Delta y + h = 0.$   where   $h = h(u^i, x^{i,k} y^{i,k})$
      $k = k + 1$
    UNTIL   $\| (\Delta x, \Delta y) \| < \tau^i$
    $(x^{i+1}, y^{i+1}) = (x^{i,k+1}, y^{i,k+1})$
    $\frac{dg_y}{du} \cong - \left\{ \frac{\partial h(u^i, x^{i+1} y^{i+1})}{\partial xy} \right\}^{-1} \frac{\partial h(u^i, x^{i+1}, y^{i+1})}{\partial u}$
    Compute $(u^{i+1}, w^{i+1}) = (u^i, w^i) + (\Delta u, \Delta w)$   by solving
    $\left( \frac{\partial g}{\partial u} + \frac{\partial g}{\partial y} \frac{dg_y}{du} \right) \Delta u + \frac{\partial g}{\partial w} \Delta w + g = 0$
    where $g = g(u^{i+1}, g_y(u^{i+1}), w^{i+1})$
    $i = i + 1$
    $\tau^{i+1} = \min[\tau^0, \| (\Delta u, \Delta w) \|^\gamma]$
  UNTIL   $\| \Delta u, \Delta w \| < \eta$ .
END.
```

Again, the latency properties of circuits, as described in the first method above, can be exploited to reduce the computation in the multilevel-Newton method [7.31]. That is, a subcircuit, at any level in the decomposition hierarchy, does not have to be processed at a given time-point unless its internal variables or input signals are changing.

7.3.3 Indirect nonlinear solution methods

A second approach for achieving decomposition at the nonlinear level is using iterative methods such as nonlinear-Gauss-Seidel or nonlinear-Gauss-Jacobi. Let us consider (7.1.3) and let us drop the subscript related to the time-point for the sake of clarity. In this section, subscripts denote scalar variables and superscripts the iteration count.

Nonlinear-Gauss-Jacobi

Repeat until convergence:

Solve: $$g_k(x_1^j, \dots, x_k^{j+1}\quad x_n^j) = 0 \qquad \text{for } x_k^{j+1},\ k = 1, 2, \dots n \tag{7.3.6}$$

Nonlinear-Gauss-Seidel

Repeat until convergence:

Solve: $$g_k(x_1^{j+1}, \dots, x_k^{j+1}, x_{k+1}^j, \dots, x_n^j) = 0 \qquad \text{for } x_k^{j+1},\ k = 1, 2, \dots n \tag{7.3.7}$$

As in the case of liner Gauss-Jacobi and Gauss-Seidel methods, it is important to investigate the conditions of convergence of the methods.

Theorem 7.3.2

Let $g'(x)$ denote the Jacobian of g computed at x. Let g be continuously differentiable in an open neighborhood S^0 of $\hat{x}$, for which $g(\hat{x}) = 0$. Let $g'(\hat{x})$ be split as $D(\hat{x}) + L(\hat{x}) + U(\hat{x})$, where $D(\hat{x})$, $L(\hat{x})$and $U(\hat{x})$ are diagonal, strictly lower triangular and strictly upper triangular respectively. Let $D(\hat{x})$ be nonsingular, and let

$$M_{GJ}(\hat{x}) \equiv \quad D\hat{x}^{-1}(L\hat{x} + U\hat{x})$$

and:

$$M_{GS}(\hat{x}) \quad - (D\hat{x} + L\hat{x})^{-1}U\hat{x}$$

Let the eigenvalues of $M_{GJ}(\hat{x})$and $M_{GS}(\hat{x})$ be inside the unit disk. Then there exist an open ball $S \subseteq S^0$ such that the nonlinear Gauss-Jacobi and Gauss-Seidel iterations are well defined for each $x^0 \in S^0$ and the sequence generated by the iterations converges to $\hat{x}$. ■

The theorem relies on the assumption that (7.3.6) and (7.3.7) can be solved exactly. In practice, the Newton-Raphson method is used to approximate the solution of these equations. Note that each equation is solved for a scalar variable.

It is important to compare the nonlinear-Gauss methods in which the internal loop is Newton-Raphson (described above) to the Newton-Raphson method in which the internal loop is a linear-Gauss method (described in the previous section). In the nonlinear-Gauss method only one Newton Raphson iteration on (7.3.6) or (7.3.7) is sufficient to preserve the convergence properties of the relaxation method [7.27]. The overall rate of convergence of the nonlinear-Gauss methods is linear. For this reason, one Newton step only is taken at each iteration, which requires the computation of only one partial derivative and no matrix inversion. For this reason this scheme is efficient to solve nonlinear equation. If, on the other hand, the relaxation is carried-out at the linear level as an internal loop of a Newton iteration, the number of relaxation iterations affects the overall rate of convergence. If only one relaxation iteration is performed, the overall rate of convergence is linear. However the rate of convergence improves to be asymptotically quadratic, as more relaxation iterations are taken. In this case the entire Jacobian matrix of the system has to be stored.

The advantage of using nonlinear-Gauss methods for circuit simulation relies particularly in handling each node equation at a time and therefore on the possibility of using event-driven analysis. The disadvantages, with comparison to the use of direct methods, as in standard simulators, are in the slower rate of convergence and some (mild) assumptions on the nature of the circuit required for convergence. For this reason, relaxation-based simulation has shown to be effective especially for digital MOS circuits, where the quasi-unidirectionality of the circuits allow an acceptable rate of convergence and event-driven analysis exploits fully the latency of the circuit.

Iterated timing analysis [7.35] is based on solving (7.1.3) by means of a nonlinear Gauss-Seidel scheme. One Newton-Raphson iteration is used to approximate the solution of each nodal equation at each relaxation iteration. However, the relaxation iteration is carried to convergence. Note that previous simulators, called *timing simulators* and described in the next section used only one relaxation iteration. For this reason, simulators that use a nonlinear relaxation technique and carry the relaxation iteration to convergence are called iterated timing analyzers.

Iterated timing analyzers are not as fast as timing analyzers but are more robust, in the sense that the computed circuit waveforms converge to the analytical solution under mild assumptions for a large class of practical circuits. This is not true in general for timing analyzers that do not carry the relaxation iteration to convergence, as will be shown in the next section.

To be more specific, we consider now the numerical properties of iterated timing analysis. Since in iterated timing analysis the nonlinear circuit equations are solved by an iterative method until satisfactory convergence, the numerical properties of the integration formulae (7.1.2) are retained. As mentioned in Theorem 7.2.2, a sufficient condition for convergence of the the relaxation iteration is diagonal-dominance of the Jacobian. If we assume that circuit equations (7.1.1) are nodal equations and inductive effect are neglected, only the capacitive elements of the Jacobian are a function of the integration step-size and dominate when this tends to zero. For circuits having only two-terminal capacitive elements, strictly positive for any voltage, and having the property that each node has a capacitor to ground, the Jacobian is diagonally dominant for values of the integration step-size sufficiently small. Therefore:

Theorem 7.3.3

There exists a time-step $\hat{h} > 0$, such that $\forall h_k \leq \hat{h}$ the iterated timing analysis algorithm converges to the solution of the discretized circuit equation. ■

Iterated timing analysis is used in mixed-mode simulators SPLICE 1.6 [7.35] and in some version of MOTIS2 [7.6]. In SPLICE 1.6, iterated timing analysis is used in conjunction with event-driven selective-trace algorithms to exploit latency. The combination of iterated-timing analysis and event-driven scheduling outperforms general purpose circuit simulators in the analysis of large scale loosely-coupled circuits as far as computing-time and memory requirements are considered, while retaining reasonable accuracy of the solution. Event-driven analysis has been widely used in logic and circuit simulation. We refer the reader to [7.39] [7.38] [7.24] [7.33] [7.34] for further details.

7.4 Decomposition at the discrete-time circuit equation solution level

Timing simulators, introduced in the early seventies, were a major revolution in circuit analysis. Timing simulators [7.5] [7.13] [7.24] [7.8] [7.1] provide voltage and/or current waveforms as a function of time, where the accuracy of the solution is compromised to increase computation speed. Timing simulators are based on indirect methods, but relaxation is used in a innovative way.

MOTIS [7.5] was the first timing simulator and differed from standard simulators in two key aspects:

i) MOTIS could only simulate MOS circuits with a capacitor to ground at every node of the circuit. Inductive effects could not be included.

ii) MOTIS avoided both sparse Gaussian Elimination and the conventional Newton-Raphson iteration as solution methods.

In MOTIS the Backward Euler Formula is used to discretize the time derivative operator and a nonlinear Gauss-Jacobi like relaxation technique was adopted to decouple the node equations at the nonlinear equation level [7.5]. Moreover, the nonlinear relaxation iteration was not taken to convergence: only one sweep was taken. The accuracy of the computed solution was controlled by keeping the integration step-size small enough, so that the result of one-sweep relaxation could reasonably approximate the solution.

Later, two other timing simulators were introduced. Fan [7.13] perfected the MOTIS technique in MOTIS-C. The trapezoidal formula was used in conjunction with one-sweep Gauss-Seidel relaxation. The timing simulator SPLICE1 [7.24] used the Backward Euler integration formula in conjunction with one-sweep Gauss-Seidel relaxation. A selective-trace algorithm was used to both order the nodes of the circuit and exploit the 'latency' of large digital circuits. The selective-trace ordering made the one sweep relaxation faster and more accurate than in the previous simulators.

Timing simulators have proven successful when applied to simulating circuits from IC design styles, based on standard cells [7.28] or gate arrays, since they have well-defined circuit configurations, have not been as successful in the custom design environment [7.26]. Timing simulators failed to simulate circuits with feedback loops, as in the case of floating capacitors, with enough accuracy to satisfy designers [7.24b][7.10]. Simulators based on Iterated Timing Analysis provide more reliable solutions, as will be shown in the sequel. However, when timing simulators can be used, they provide over two order of magnitude speed improvement over conventional circuit simulators with reasonable waveform accuracy [7.26].

It is important to stress that timing simulators introduced a novelty by not carrying the relaxation iteration to convergence. In particular, the numerical properties of the numerical integration formulae no longer hold and these methods have to be considered as *new integration methods*. Hence a complete analysis of their numerical properties has to be carried out to characterize them. Historically timing simulators were developed and used before such an analysis was done. This explains why some of the difficulties of these methods were unexpected and why other techniques, such as Iterated Timing Analysis and Waveform Relaxation, were subsequently developed and implemented.

7.4.1 Timing simulation algorithms

Timing analysis programs (e.g. MOTIS and the simulation part of the mixed mode simulator SPLICE1) assemble the differential equations that describe the circuit by using nodal analysis. Nodal analysis limits the circuits to be analyzed to these that can be modeled by voltage-controlled current sources, voltage-controlled capacitors, voltage-controlled resistors and independent current sources. In timing simulators it is also assumed that each node in the circuit has a capacitor to ground or to a fixed voltage rail. These assumptions are usually satisfied by most practical MOS circuits, where these capacitors are used to model the time delay of a signal propagating through the circuit. With this formulation, the node equations are also state equations of the circuit and voltages are state variables. The circuit equation can be written as follows:

$$C(v)\dot{v} + f(v,t) = 0$$
$$v(0) = v_0 \qquad (7.4.1)$$
$$v \in R^n; \quad C(\,\bullet\,): R^n \to R^{n \times n} \quad f(\,\bullet\,,\bullet\,): R^n \times R \to R^n$$
$$f(v,t) = [f^1(v,t), f^2(v,t), \ldots, f^n(v,t)]^T$$

where v is a vector of n node voltages, $C(v)$ is the nonlinear nodal capacitance matrix and $f^i(v,t)$ represents the sum of the currents flowing out of the capacitors at the node i

The structure of the nodal capacitance matrix $C(v)$ depends on the modelling of inter-nodal capacitance. Timing simulator MOTIS and the early version of SPLICE1 did not allow *floating capacitors*, i.e. capacitors whose nodes are both not connected to a fixed voltage reference (or ground). Later, models and algorithms for floating capacitors were introduced.

The analysis of integration methods which exclude floating capacitors is considered first. In this case, the nodal capacitance matrix $C(v)$ is a diagonal matrix. Timing simulators assumed the existence of a non-zero capacitance to ground from each node for any value of v of interests. Therefore the nodal capacitance matrix has an inverse and the circuit equation can be written as:

$$\dot{v} + F(v,t) = 0 \qquad (7.4.2)$$
$$v(0) = v_0$$

where:

$$F(v,t) \equiv C(v)^{-1} f(v,t) \qquad (7.4.3)$$

Algorithms used for timing analysis often discretize the derivative operator by the Backward Euler or Trapezoidal integration formula. For the sake of simplicity, the backward Euler formula will be used:

$$\dot{v}_{k+1} = \frac{v_{k+1} - v_k}{h} \qquad (7.4.4)$$

where the time-step $h = t_{k+1} - t_k$ and v_k is the computed approximation of the nodal voltage vector v at time t_k. The resulting algebraic nonlinear system of equations is:

$$v_{k+1} - v_k + hF(v_{k+1}, t_{k+1}) = 0 \qquad (7.4.5)$$

This set of nonlinear algebraic equation must be solved at each time-point.

Example 7.4.1: Consider the circuit shown in Fig. 7.4.1

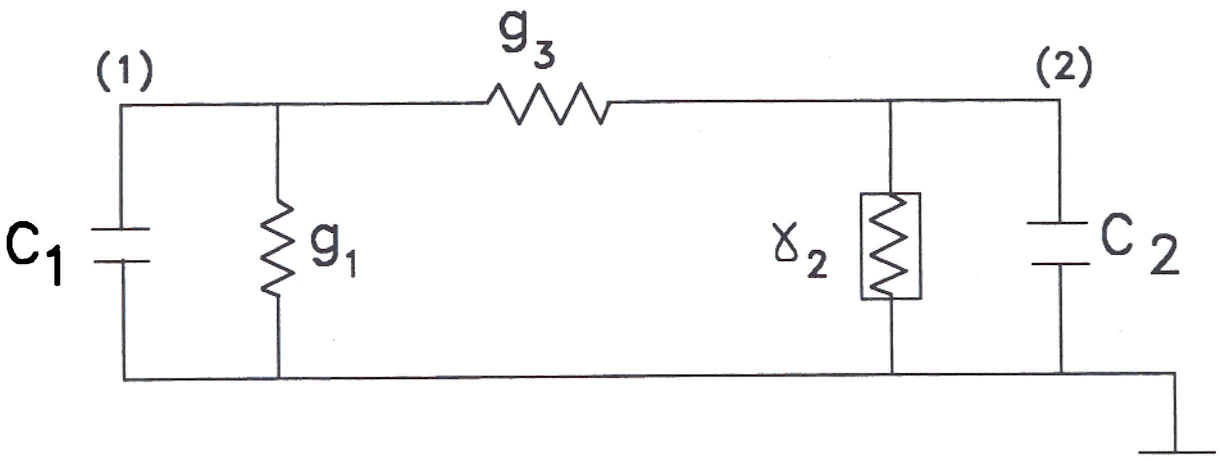

Figure 7.4.1 Nonlinear Sample Circuit

The circuit has two linear resistors g_1, g_3, two nonlinear capacitors $c_1(v^1)$, $c_2(v^2)$ and a a nonlinear voltage-controlled current source $\gamma_2(v^1)$. The node equation for the circuit can be written as:

$$\begin{bmatrix} c_1(v^1) & 0 \\ 0 & c_2(v^2) \end{bmatrix} \dot{v} + \begin{bmatrix} (g_1 + g_3)v^1 - g_3 v^2 \\ -g_1 v^1 + g_3 v^2 - \gamma_2(v^1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The nonlinear algebraic system of equation at time-point t_{k+1} is:

$$v^1_{k+1} - v^1_k + \frac{h}{c_1(v^1_{k+1})}\left((g_1 + g_3)v^1_{k+1} - g_3 v^2_{k+1}\right) = 0$$

$$v^2_{k+1} - v^2_k + \frac{h}{c_2(v^2_{k+1})}\left(-g_1 v^1_{k+1} + g_3 v^2_{k+1} + \gamma_2(v^1_{k+1})\right) = 0$$

The solution of this set of equations are the voltages at time-point t_{k+1}.

When the number of nodes n in the circuit is large, solving such a system of equations with a direct method is expensive in terms of computer time. Moreover the Newton-Raphson iteration would require the computation and storage of the Jacobian matrix of the system, which has n^2 elements. Even if sparse matrix techniques are used, the computing-time and memory requirements make direct methods unattractive for large circuits.

For this reason, timing simulators use indirect methods to approximate the solution of such a system. By using a nonlinear indirect method, the Jacobian is a diagonal matrix. Moreover there is no need to compute and store the Jacobian matrix as a whole, because the equations are decoupled and only one scalar entry of the Jacobian matrix is needed while solving each equation. The advantage in computation speed is due mainly to taking only one sweep of the nonlinear relaxation.

The program MOTIS uses a Gauss-Jacobi like technique which yields the following set of decoupled equations:

$$v_{k+1}^{i} - v_k^{i} + hF^{i}(v_k^{1}, \dots, v_k^{i-1}, v_{k+1}^{i}, v_k^{i+1}, \dots, v_k^{n}, t_{k+1} \quad = 0 \quad ; \; i = 1,2, \;\; . \,, n \qquad (7.4.6)$$

The solution of the decoupled linear equations is then approximated by taking a single step of a *regula falsi* iteration [7.21].

Example 7.4.2: Consider the circuit of Example 7.4.1. The decoupled equations at time-point t_k are:

$$v_{k+1}^{1} - v_k^{1} + \frac{h}{c_1(v_{k+1}^{1})}((g_1 + g_3)v_{k+1}^{1} - g_3 v_k^{2}) = 0$$

$$v_{k+1}^{2} - v_k^{2} + \frac{h}{c_2(v_{k+1}^{2})}(- g_1 v_k^{1} + g_3 v_{k+1}^{2} + \gamma_2(v_k^{1})) = 0$$

which can be solved for v_{k+1}^{1} and v_{k+1}^{2} Note that the two equations can be solved in any order. By applying one sweep of the *regula falsi* iteration:

$$v_{k+1}^{1} = v_k^{1} - \frac{1}{\Delta^{1}(v_k^{1})}\left\{ \frac{h}{c_1(v_k^{1})}((g_1 + g_3)v_k^{1} - g_3 v_k^{2}) \right\}$$

$$v_{k+1}^{2} = v_k^{2} - \frac{1}{\Delta^{2}(v_k^{2})}\left\{ \frac{h}{c_2(v_k^{2})}(- g_1 v_k^{1} + g_3 v_k^{2} + \gamma_2(v_k^{1})) \right\}$$

where

$$\frac{g_1 + g_3}{v_k^{1} - v_{k-1}^{1}}\left(\frac{v_k^{1}}{[illegible]} - \frac{v_{k-1}^{1}}{[illegible]} \right)$$

$$\Delta^{2} \equiv \frac{1 + h}{[illegible]}\left(\frac{1}{[illegible]}(- g_1 v_k^{1} + g_3 v_k^{2} + \gamma_2(v_k^{1})) + \right.$$

$$\left. (- g_1 v_k^{1} + g_3 v_{k-1}^{2} + \gamma_2(v_k^{1}))\right)$$

approximate the Jacobian of the set of nonlinear algebraic equations.

The MOTIS-C and SPLICE1 programs use a Gauss-Seidel like technique. When the equations are ordered as the flow of the signal through the circuit, one Gauss-Seidel iteration yields a better accuracy than Gauss-Jacobi, because the updated voltages at nodes $1, 2, \ldots, i-1$ are used to compute the voltage at node i. In SPLICE1 this technique yields:

$$v_{k+1}^i - v_k^i + hF^i(v_{k+1}^1, \ldots, v_{k+1}^{i-1}, v_{k+1}^i, v_k^{i+1}, \ldots, v_k^n, t_{k+1}) = 0; \; i = 1,2, \ldots, n \qquad (7.4.7)$$

The solution is then approximated by one step of the Newton-Raphson algorithm.

Example 7.4.3: Consider the circuit of Example 7.4.1. The decoupled equations at time-point t_k are:

$$v_{k+1}^1 - v_k^1 + \frac{h}{c_1(v_{k+1}^1)}((g_1 + g_3)v_{k+1}^1 \quad g_3 v_k^2) = 0$$

$$v_{k+1}^2 - v_k^2 \quad \frac{h}{c_2(v_{k+1}^2)}(-g_1 v_{k+}^1 + g_3 v_{k+1}^2 + \gamma_2(v_{k+1}^1)) = 0$$

which can be solved for v_{k+1}^1 and v_{k+1}^2 By applying one sweep of the Newton-Raphson iteration:

$$v_{k+}^1 = v_k^1 \quad \frac{1}{J^1(v_k^1)} \left\{ \frac{h}{c_1(v_k^1)}((g_1 + g_3)v_k^1 - g_3 v_k^2) \right\}$$

$$v_{k+1}^2 = v_k^2 - \frac{1}{J^2(v_k^2)} \left\{ \frac{h}{c_2(v_k^2)}(-g_1 v_k^1 + g_3 v_k^2 + \gamma_2(v_k^1)) \right\}$$

where:

$$J^1 \equiv 1 + \frac{h}{\cdot}(g_1 + \gamma_2) \quad \frac{h}{c_1^2(v_k^1)}(g_1 + g_3)v_k^1$$

$$J^2 \equiv 1 \quad \frac{h}{\;_{,\,2,}} g_3 + \frac{h}{2,\,1,}(\quad g_1 v_k^1 + g_3 v_k^2 + \gamma_2(v_{k+1}^1))$$

are the diagonal entries of the Jacobian matrix of the set of nonlinear algebraic equations.

The Gauss-Seidel like approach used in SPLICE1 does not take into account the feedbacks inside the circuit, because of the nature of the Gauss-Seidel method. For example, when two nodes i and j are tightly coupled and node i (j) is scheduled to be processed before node j (i) , the voltage at node i (j) is computed by not taking into account the updated voltages at node j (i). This approximation can cause inaccuracies and even instability of the integration method [7.24b] [7.10].

For this reason, a symmetric Gauss-Seidel scheme is considered. A modified symmetric Gauss-Seidel technique, proposed by W. Kahan for general set of ordinary differential equations, has been adapted to circuit simulation [7.9] and implemented in an experimental timing simulator [7.11]. The integration method consists of two half-steps. During the former a Gauss-Seidel like sweep is taken in the usual "forward" direction; during the latter the order of the nodes is reversed and the Gauss-Seidel sweep is taken in a backward direction. The intermediate result is then discarded. To achieve complete symmetry, the i^{th} unknown of F^i is evaluated at $\frac{v^i_{k+1/2} + v^i_k}{2}$. The forward step yields:

$$v^i_{k+1/2} - v^i_k + \frac{h}{2} F^i\left(v^1_{k+1/2}, \ldots, v^{i-1}_{k+1/2}, \frac{v^i_{k+1/2} + v_{k+1}}{2}, v^{i+1}_k, \ldots, v^n_k, t_{k+1/2}\right) = 0 \tag{7.4.8a}$$

for $i = 1,2, \ldots, n$ and the backward step:

$$v^i_{k+1} - v^i_{k+1/2} + \frac{h}{2} F^i\left(v^1_{k+1/2}, \ldots, v^{i-1}_{k+1/2}, \frac{v^i_{k+1/2} + v_{k+1}}{2}, v^{i+1}_{k+1}, \ldots, v^n_{k+1}, t_{k+1}\right); \tag{7.4.8b}$$

for $i = n, n-1, \ldots, 1$.

Example 7.4.4: Consider the circuit of Example 7.4.1. The time-step is divided into two half-steps. First a forward Gauss-Seidel sweep is taken, with time-step $t_{k+1/2}$ The forward step is:

$$v^1_{k+1/2} - v^1_k + \frac{h}{2c_1\left(\frac{v^1_{k+1/2} + v^1_k}{2}\right)}\left((g_1 + g_3)\frac{v^1_{k+1/2} + v^1_k}{2} - g_3 v^2_k\right) = 0$$

$$v^2_{k+1/2} - v^2_k + \frac{h}{2c_2\left(\frac{v^2_{k+1/2} + v^2_k}{2}\right)}\left(-g_1 v^1_{k+1/2} + g_3\frac{v^2_{k+1/2} + v^2_k}{2} + \gamma_2(v^1_{k+1/2})\right) = 0$$

which can be solved for $v^1_{k+1/2}$ and $v^2_{k+1/2}$
The backward step is:

$$v^2_{k+1} - v^2_{k+1/2} + \frac{h}{\frac{v^2_{k+1} + v^2_{k+1/2}}{2})}\left(-g_1 v^1_{k+1/2} + g_3\frac{v^2_{k+1} + v^2_{k+1/2}}{2} + \gamma_2(v^1_{k+1/2})\right) = 0$$

$$v^1_{k+1} - v^1_{k+1/2} + \frac{h}{\frac{v^1_{k+1} + v^1_{k+1/2}}{2})}\left((g_1 + g_3)\frac{v^1_{k+1} + v^1_{k+1/2}}{2} - g_3 v^2_{k+1}\right) = 0$$

The solution of the decoupled equations is then approximated by taking one step of the Newton-Raphson algorithm as in the previous example.

These methods have been called "time-advancement" algorithms, because time is incremented at each relaxation sweep. The new integration algorithms, obtained by combining the integration formula with one sweep of Gauss-Jacobi, Gauss-Seidel and modified symmetric Gauss-Seidel relaxation, will be referred to as Gauss-Jacobi (GJ), Gauss-Seidel (GS) and modified symmetric Gauss-Seidel (MSGS) time-advancement integration algorithm respectively.

As a final remark, note that these algorithms can be applied to any circuit whose equation can be written in normal form. However, since the main issue here is speed of computation, the assumptions on the nature of the circuit guarantee that the state equations can be assembled in normal form directly from the input description of the circuit. In general, formulating the circuit equations in normal form is a computationally intensive task.

7.4.2 Numerical properties of the time-advancement algorithms.

The numerical properties of an integration method, such as stability and accuracy, are studied on test problems [7.7] [7.14] which are simple enough to allow a theoretical analysis but still sufficiently general that some insight can be obtained about how the method will behave in general. For the widely used multistep methods, the test problem consists of a linear time-invariant asymptotically stable autonomous differential equation. Unfortunately this simple test problem cannot be used to evaluate relaxation based time-advancement algorithms. In fact, each variable of the system of differential equations is treated differently according to the ordering in which equations are processed. Hence a more complex test problem is needed. A test problem suitable for studying the numerical properties of the time advancement algorithms is a linear time-invariant asymptotically stable *system* of autonomous differential equations:

$$\dot{v} = Av \qquad (7.4.9)$$
$$v(0) = v_0$$

where $v \in R^n$ $A \in R^{n \times n}$ and the spectrum of A, $\sigma(A)$, is in the open left half complex plane, i.e. $\sigma(A) \in C^-$.

In circuit theoretic terms, linear circuits whose natural frequences are in the open left-half plane and which satisfy the assumptions of the previous section are considered as test circuits. Let $A = L + D + U$, where L is strictly lower triangular, D is diagonal and U is strictly upper triangular respectively. The analysis of the algorithms is carried out using a constant stepsize h. The time-advancement algorithms applied to the test system yield the following iterative relations:

i) Gauss-Jacobi time-advancement method:

$$[I - hD]v_{k+1} = [I + h(L + U)]v_k \qquad (7.4.10a)$$

which can be written as:

$$v_{k+} = M_{GJ}(h)v_k \qquad (7.4.10b)$$

$$M_{GJ}(h) \equiv [I - hD]^{-1}[I + h(L + U)] \qquad (7.4.10c)$$

ii) Gauss-Seidel time-advancement method:

$$[I \quad h(D + L)]v_{k+1} \qquad [I + hU]v_k \qquad (7.4.11a)$$

which can be written as:

$$v_{k+1} \qquad M_{GS}(h)v_k \qquad (7.4.11b)$$

$$M_{GS}(h) \qquad [I - h(D + L)]^{-1}[I + hU] \qquad (7.4.11c)$$

iii) Modified symmetric Gauss-Seidel time-advancement method: Let:

$$A_L \equiv L + \frac{1}{2}D \quad A_U \equiv U + \frac{1}{2}D \qquad (7.4.12a)$$

Forward half-step:

$$[I - \frac{h}{2}A_L]v_{k+1/2} = [I + \frac{h}{2}A_U]v_k \qquad (7.4.12b)$$

Backward half-step:

$$[I - \frac{h}{2}A_U]v_{k+1} = [I + \frac{h}{2}A_L]v_{k+1/2} \qquad (7.4.12c)$$

which can be written as:

$$v_{k+1} = M_S(h)v_k \qquad (7.4.12d)$$

$$M_S(h) \equiv [I \quad \frac{h}{2}A_U]^{-1}[I + \frac{h}{2}A_L][I - \frac{h}{2}A_L]^{-1}[I + \frac{h}{2}A_U] \qquad (7.4.12e)$$

The matrices $M_{GJ}(h)$, $M_{GS}(h)$, and $M_S(h)$ are the *companion matrices* of the methods. If $M(h)$ denotes the generic companion matrix of a method, then:

$$v_{k+1} = M(h)v_k$$

The companion matrices characterize the integration method and the numerical properties of the algorithms can be studied on the companion matrices. A complete analysis of the time-advancement algorithms for the equations in normal form is reported in [7.10]. The major results are summarized here. The numerical properties of the time-advancement integration algorithms are described following the outline of one-step integration methods applied to ordinary differential equations [7.14], and are related to the test problem 7.4.9.

Definition 7.4.1.

An integration algorithm is consistent if its companion matrix can be expanded in power series as a function of the step-size h as:

$$M(h) = I + hA + 0(h^2)$$

Theorem 7.4.1.

The Gauss-Jacobi, Gauss-Seidel and modified symmetric Gauss-Seidel time-advancement algorithms are consistent.■

Definition 7.4.2.

An integration algorithm is stable if $\exists \delta > 0$, $\exists N > 0$ such that $\forall v_0 \in R^n$, $\exists \bar{k} > 0$

$$\| v_k \| > N \quad \forall k \geq \bar{k} \quad \forall h \in [0, \delta)$$

where v_k is the sequence generated by the algorithm applied to the test problem.

Theorem 7.4.2.

The Gauss-Jacobi, Gauss-Seidel and modified symmetric Gauss-Seidel time-advancement algorithms are stable.■

Definition 7.4.3.

Let $v(\cdot)$ be the exact solution of the test problem. An integration algorithm is convergent if the sequence of the computed solutions converges uniformly to $v(\cdot)$ as the step-size h tends to zero.

Theorem 7.4.3.

The Gauss-Jacobi, Gauss-Seidel and modified symmetric Gauss-Seidel time-advancement algorithms are convergent.■

These results show that the time-advancement algorithms can be used to analyze circuits without floating capacitors. The numerical stability of the methods can be achieved by monitoring the step-size, as in the case of the explicit integration methods.

For computational efficiency, it would be highly desirable that the step-size be limited only by accuracy considerations, as in the case of some implicit methods, such as backward Euler and the trapezoidal. In the case of multi-step methods, the concepts of A-stability [DALQ71] and stiff-stability [7.14] have been introduced to test for unconditional stability. For the time-advancement algorithms, it would make sense to define a similar concept. The general results of unconditional stability are not available for the test problem previously defined, but only for a subclass, the subclass characterized by a symmetric A matrix. In circuit theoretic terms, only linear circuits are considered now, whose node equation yield a nodal admittance matrix, when only the resistive part of the circuit is considered. Moreover it is required that this matrix remain symmetric when premultiplied by C^{-1}, the diagonal matrix of the grounded capacitors. A sufficient condition for this to occur, is that the circuit consists of two-terminal resistors and capacitors, and the grounded capacitors be equal. The case of unequal grounded capacitors can also be included in this class, provided that a scaling of the rows of the matrix is performed.

Definition 7.4.4.

An integration algorithm is $\tilde{A}$-stable if, for each test problem with A symmetric, $\exists N > 0$ such that $\forall v_0 \in R^n, \exists \bar{k} > 0$

$$\| v_k \| < N \quad \forall k \geq \bar{k} \quad \forall h \in [0, \infty)$$

where v_k is the sequence generated by the algorithm.

Theorem 7.4.4.

The modified symmetric Gauss-Seidel time-advancement algorithms when applied in solving (7.4.9) with A being symmetric is $\tilde{A}$-stable. ■

Even if this result can be proven under restrictive assumptions, experimental results on a wide set of circuits [7.11] have shown that the modified symmetric Gauss-Seidel algorithm has better stability property than other time-advancement methods.

A major limiting factor in determining the step-size of the time-advancement methods is preserving the accuracy of the solution.

Definition 7.4.5.

Let $v(t_k)$ be the exact value of the solution to the test problem at time t_k. Let v_k be the computed solution at time t_k assuming $v_{k-1} = v(t_{k-1})$, i.e. no error has been made in computing the previous time-point value of v. Let $h = t_k - t_{k-1}$. The local truncation error is defined to be:

$$\varepsilon = \| v_k - v(t_k) \|$$

If $\varepsilon = 0(h^{r+1})$, r is said to be the order of the integration method.

Theorem 7.4.5.

The Gauss-Jacobi and Gauss-Seidel integration methods are first order integration algorithms. The modified symmetric Gauss-Seidel is a second order integration algorithm.■

In circuit analysis, another important criterion for evaluating the accuracy of an integration method, can be defined as *waveform accuracy* . In general, the computed approximation to the solution of a system of differential equations is the superposition of a principal solution and associated parasitic solutions. Parasitic solutions are generated by the numerical approximation of the integration methods. In particular, a P order integration method yields $P - 1$ parasitic solutions when applied to the test problem. For the algorithms under consideration, the displacement techniques introduce additional spurious components called *numerical solution components* .

If the original system to be analyzed does not contain an oscillatory component, the presence of such a component in the computed solution may be misleading in evaluating the performances of the system. It is then necessary to measure the waveform accuracy of the integration method. To this end, a subclass of the test problem is now introduced, characterized by the circuits whose state matrix A have no pair of complex eigenvalues, i.e. $\sigma(A) \in R$ These circuits do not show oscillatory components in the zero-input response.

It is clear that convergence of the time-advancement methods implies that oscillatory parasitic components can be bounded by limiting the step-size. By restricting the class of test problems to the subclass characterized by a symmetric A matrix, it is possible to prove a stronger result for the modified symmetric Gauss-Seidel integration method.

Theorem 7.4.6.

If A in (7.4.9) is a real symmetric matrix, the modified symmetric Gauss-Seidel method does not introduce parasitic oscillatory components for any value of the step-size. ■

The numerical properties of the three time-advancement integration algorithms applied to circuits without floating capacitors can be summarized as follows. The methods are stable and consistent and therefore they can provide reliable solutions, provided that the time-step is controlled to insure the required stability and accuracy. Waveform accuracy can be guaranteed by monitoring the step-size as well. The modified symmetric Gauss-Seidel integration algorithm shows better stability and accuracy properties than the two other methods. In particular the stepsize is not limited by stability and waveform accuracy consideration for a wide class of circuits. Moreover it is a second order integration method and therefore the step-size can assume larger values for a given bound on the local truncation error.

7.4.3 Circuits with floating capacitors

The use of time-advancement integration algorithms and timing simulation has been limited to circuits without floating capacitors. This implies that the gate-drain and gate-source feedthrough capacitances of MOS devices, often critical to circuit performance, cannot be included in simulation. Even if the physical values of these capacitances are small, their effects can be significantly magnified in situations involving gain like the well known Miller effect.

Early timing simulators avoided the problem of analyzing circuits with floating capacitors by not allowing the user to include them in the circuit description. In MOTIS, the effect of a floating capacitor is approximated by altering the values of the grounded capacitors at appropriate nodes in the circuit. In the MOTIS-C program, isolated floating capacitors are processed by maintaining the node coupling across the floating branch and solving the resulting 2 × 2 nodal circuit matrix at each time-point. This approach could be extended to deal with arbitrary connections of N floating capacitors, but this would require the solution of $N + 1$ coupled equations at each time-point and, hence, reduce the advantages of the node decoupling approach.

Here a general framework for dealing with floating capacitors is presented. Since the speed of the computation is of utmost importance, the time-advancement algorithms are generalized to circuit equation formulation that can be assembled directly from the input description.

When floating capacitors are present in the circuit to be analyzed, the nodal capacitance matrix $C(v)$ is not diagonal and it is not computationally efficient to invert C to obtain the nodal equations in state form. Therefore circuit equations are:

$$\begin{aligned} C(v)\dot{v} + f(v,t) &= 0 \\ v(0) &= v_0 \end{aligned} \tag{7.4.13}$$

By discretizing the derivative operator using the Backward Euler formula, the resulting set of nonlinear algebraic equations is:

$$C(v_{k+1})v_{k+1} - C(v_{k+1})v_k + hf(v_{k+1}, t_{k+1}) = 0 \tag{7.4.14}$$

This set of nonlinear algebraic equations must be solved at each time-point.

The approach used in the MOTIS program can be formalized in the frame of the time-advancement integration algorithms. The integration method is based on one-sweep Gauss-Jacobi relaxation.

$$c_{ii}(v^i_{k+1})v^i_{k+1} - c_{ii}(v^i_{k+1})v^i_k + hf^i(v^1_k, \dots, v^{i-1}_k, v^i_{k+1}, v^{i+1}_k, \dots, v^n_k, t_{k+1}) = 0; \quad i = 1,2, \dots, n \tag{7.4.15}$$

Note that using one Gauss-Jacobi sweep corresponds to simulating the effect of a floating capacitor by replacing it with a pair of grounded capacitors of the same value. In fact, the diagonal entries of the nodal capacitance matrix c_{ii} represent the sum of all capacitances connected to node i. (See Chapter 2).

The Gauss-Jacobi time-advancement algorithm is not consistent for circuits containing floating capacitors, so no matter how small an integration step-size is used, the computed waveform will approximate the exact solution of a circuit whose floating capacitors have been replaced by grounded capacitors, not the exact solution of the original circuit. Therefore, when the function of a floating capacitor is critical to the performance of the circuit (e.g. a bootstrapped inverter [7.24]) the numerical solution cannot provide accurate information. However, it is important to remark that this method has been widely used in the past. The method is useful for circuit simulation, as long as its limitations are spelled out clearly.

The Gauss-Seidel time-advancement method can be extended to circuits with floating capacitors:

$$\sum_{j=1}^{i} c_{ij}(v_{k+1}^{j}) v_{k+}^{j} - \sum_{j=1}^{i} c_{ij}(v_{k+1}^{j}) v_{k}^{j} + hf^{i}(v_{k+1}^{1}, \ldots, v_{k+1}^{i-1}, v_{k+1}^{i}, v_{k}^{i+1}, \ldots, v_{k}^{n}, t_{k+} = 0; \tag{7.4.16}$$

for $i = 1,2, \ldots, n$. This scheme takes into account only the feedforward effect of the floating capacitors, but it neglects the feedback effects. As a result, the method is not consistent.

The Gauss-Seidel time-advancement method has shown poor waveform accuracy for circuit containing floating capacitors [7.24b]. In particular, parasitic oscillations are present in the computed solution and these do not represent the physical dynamics of the circuit. These oscillations are related to considering only the feedforward effect of the floating capacitors and not the feedback effect, i.e. they are related only to the numerical manipulation of the equations.

As an attempt to consider both the feedback and the feedforward effect of floating capacitors, a family of symmetric Gauss-Seidel time-advancement methods has been studied [7.11]. As an example, the modified symmetric Gauss-Seidel time-advancement algorithm can be extended to circuits with floating capacitors as follows.

Forward step:

$$\sum_{j=1}^{i-1} c_{ij}(v_{k+1/2}^{j}) v_{k+1/2}^{j} + c_{ii}\left(\frac{v_{k+1/2}^{i} + v_{k}^{i}}{2}\right) v_{k+1/2}^{i} - \sum_{j=1}^{i-1} c_{ij}(v_{k+1/2}^{j}) v_{k}^{j} - c_{ii}\left(\frac{v_{k+1/2}^{i} + v_{k}^{i}}{2}\right) v_{k}^{i} +$$

(7.4. 7a)

$$+ hf^{i}\left(v_{k+1/2}^{1}, \ldots, v_{k+1/2}^{i-1}, \frac{v_{k+1/2}^{i} + v_{k+1}}{2}, v_{k}^{i+1}, \ldots, v_{k}^{n}, t_{k12}\right) = 0 \quad ; \; i = 1,2, \quad , n$$

Backward step:

$$\sum_{j=i+1}^{n} c_{ij}(v_{k+1}^{j}) v_{k+1}^{j} + c_{ii}\left(\frac{v_{k+1}^{i} + v_{k+1/2}^{i}}{2}\right) v_{k+1}^{i} - \sum_{j=i+1}^{n} c_{ij}(v_{k+1}^{j}) v_{k}^{j} - c_{ii}\left(\frac{v_{k+1}^{i} + v_{k+1/2}^{i}}{2}\right) v_{k+1/2}^{i} +$$

(7.4.17*b*)

$$+ hf^{i}\left(v_{k+1/2}^{1}, \ldots, v_{k+1/2}^{i-1}, \frac{v_{k+1/2}^{i} + v_{k+1}}{2}, v_{k+1}^{i+1}, \ldots, v_{k+1}^{n}, t_{k+1}\right) \quad ; \; i = n, n-1, \ldots, 1$$

where $\sum_{j=i}^{k} \equiv 0$ if $k < i$. The forward step takes into account the feedforward effect of the floating capacitors, while the backward step takes into account the feedback effect.

The modified symmetric Gauss-Seidel method, extended to circuits including floating capacitors, has been proven to be accurate, stable and even $\tilde{A}$-stable but only for particular classes of circuits. The interested reader is referred to [7.11] for the details. Experimental results have shown that the modified symmetric Gauss-Seidel algorithm can be used for the simulation of a large class of circuits and the step-size is not severely limited by accuracy and stability considerations.

Another family of time-advancement algorithms can be obtained by applying one relaxation sweep before replacing $\dot{v}$ by the Backward Differentiation Formulae. Consider the set of nonlinear differential equations at a time-point, for example at t_{k+1}:

$$C(v_{k+1})\dot{v}_{k+1} + f(v_{k+1}, t_{k+1}) = 0 \tag{7.4.18}$$

By using the Gauss-Jacobi relaxation scheme:

$$c_{ii}(v^i_{k+1})\dot{v}^i_{k+1} + \sum_{j=1, j\neq i}^{n} c_{ij}(v^j_k)\dot{v}^j_k + f^i(v^1_k, \dots, v^{i-1}_k, v^i_{k+1}, v^{i+1}_k, \dots, v^n_k, t_{k+1}) = 0; \tag{7.4.19}$$

for $i = 1,2, \dots, n$ where by the Backward Euler formula:

$$\dot{v}_{k+1} = \frac{v_{k+1} - v_k}{h} \quad \text{and} \quad \dot{v}_k = \frac{v_k - v_{k-1}}{h} \tag{7.4.20}$$

Similarly, by using the Gauss-Seidel relaxation scheme:

$$\sum_{j=1}^{i} c_{ij}(v^j_k)\dot{v}^j_{k+1} + \sum_{j=i+1}^{n} c_{ij}(v^j_k)\dot{v}^j_k + f^i(v^1_{k+1}, \dots, v^{i-1}_{k+1}, v^i_{k+1}, v^{i+}_k, \dots, v^n_k, t_{k+} = 0; \tag{7.4.21}$$

for $i = 1,2, \dots, n$. A symmetric Gauss-Seidel scheme can also be obtained by combining a forward and a backward Gauss-Seidel half-step. The Gauss-Seidel scheme was proposed by R.Newton and called *Implicit-Implicit-Explicit,* because the approximation $\dot{v}_{k+1} = \dot{v}_k$ and the use of the Backward-Euler formula (Implicit) is equivalent to using the Forward-Euler formula (Explicit). Hence the algorithm is halfway between implicit and explicit methods. This technique was used in some versions of program SPLICE1 [7.24] [7.25].

T. Huang proved recently the numerical properties of the Implicit-Implicit-Explicit method on the class of circuit having a grounded capacitance at each node and consisting of voltage-controlled current sources, voltage-controlled capacitors, voltage-controlled resistors and independent current sources [7.20]. For this class of circuits we can state:

Theorem 7.4.7

The integration methods (7.4.19) and (7.4.21) are consistent, stable and convergent. ■

For this reason the Implicit-Implicit-Explicit method can be used for reliable timing simulation of circuits with floating capacitors. However this method is not proven to be A-stable and waveform accuracy considerations may limit the step-size considerably.

Conclusions

In this chapter we have reviewed some decomposition techniques for the transient analysis of large-scale systems. The transient analysis leads to the solution of a set of ordinary nonlinear differential equations. Here we considered solution methods in which the simulation-time window is sliced into time-steps, and the computed solution is obtained by solving a set of algebraic nonlinear equations at each time-point. For this reason, these techniques are called incremental-time integration methods. The set of discretized nonlinear algebraic equations is solved at each time-point. If Newton's method is used, the nonlinear set of equation is solved by a sequence of sets of linear algebraic equations.

There is a hierarchy of decomposition techniques that can be applied to the solution of incremental-time integration methods. Decomposition techniques can be used to solve linear and nonlinear algebraic systems of equations. New integration methods are introduced by timing simulators, where single-step nonlinear relaxation modifies the structure and the properties of the integration formulae.

Eventually, it is conceivable to think of using decomposition techniques, such as relaxation, directly on the set of differential equations. In this case, the objects of the relaxation are in the solution space, i.e. are circuit waveforms. A solution to the set of nonlinear differential equations is obtained by improving an initial "guessed" solution, over an entire time-interval. This leads to another family of methods for the transient simulation, that have been recently discovered and applied: waveform relaxation methods. These techniques are dealt with in detail in the next chapter.

Acknowledgements

The authors would like to thank A. Ruehli, A. Sangiovanni Vincentelli and J. White for many helpful suggestions and clarifying discussions. They wish also to thank C. Otten for editing the manuscript.

References

[7.1] G. Arnout and H. DeMan, "The Use of Threshold functions and Boolean-controlled Network Elements for Macromodeling of LSI Circuits," IEEE J. Solid-State Circuits, vol. SC-13, pp. 326-332, June 1978.

C.Baker and C.Terman "Tools for verifying integrated circuit designs" Lambda, 4th quarter, 1980.

R.E. Bryant, "An Algorithm for MOS Logic Simulation, Lambda, 4th quarter, pp. 46-53, 1980.

[7.4] R.K. Brayton, G.D. Hachtel and F.G. Gustavson, "The Sparse Tableau Approach to Network Analysis and Design", IEEE Transactions on Circuit Theory., vol. CT-18, No. 1, pp. 101-113, January 1971.

[7.5] B.R. Chawla, H.K. Gummel and P. Kozak, "MOTIS-an MOS Timing Simulator," IEEE Trans. Circuits Syst., vol. CAS-22, pp. 901-909, December 1975.

[7.6] C.F.Chen, C. Lo, H.N.Nham and P. Subramaniam "The Second Generation MOTIS Mixed-mode Simulator," Proc. Des. Autom. Conf. pp. 10-17, Albuquerque, June 1984.

[7.7] G. Dahlquist and A. Bjorck, *NumericalMethods* , Englewood Cliffs, NJ: Prentice-Hall, 1974.

[7.8] H. DeMan et al., "DIANA: Mixed Mode Simulator with a Hardware Description Language for Hierarchical Design of VLSI," in IEEE ICC'80 Conf. Proc., Rye, NY, pp. 356-360, October 1980.

[7.9] G. DeMicheli, A. Sangiovanni-Vincentelli and A.R. Newton, "New Algorithms for Timing Analysis of Large Circuits," in Proc. 1980 Int. Symp. Circuits Syst. 1980.

[7.10] G. DeMicheli and A. Sangiovanni-Vincentelli, "Characterization of Integration Algorithms for the Timing Analysis of MOS VLSI Circuits," Int. J. Circuit Theory Appl., pp. 299-309, October 1982.

[7.11] G. DeMicheli, A.R. Newton and A. Sangiovanni-Vincentelli, "Symmetric Displacement Algorithms for the Timing Analysis of MOS VLSI Circuits.," IEEE Trans. Computer-Aided Design of CAS., pp. 167-179, July 1983.

[7.12] W.I. Engl, R. Laur and H. Dirks, "MEDUSA A Simulator for Modular Circuits," IEEE Trans. CAD Integ. Circ. Syst., vol. CAD-1, pp. 85-93, April 1982.

[7.13] S.P. Fan, M.Y. Hsueh, A.R. Newton and D.O. Pederson, "MOTIS-C A New Circuit Simulator for MOS LSI Circuits," in Proc. IEEE Int. Symp. Circuits Syst., April 1977.

[7.14] C.W. Gear, *Numerical Initial Value Problems for Ordinary Differential Equations*, Prentice-Hall, 1971.

[7.15] A. George, "On Block Elimination for Sparse Linear Systems", SIAM J. Numer. Anal. vol. 11, no. 3, pp. 585-603, June 1984.

[7.16] I.N. Hajj, "Sparsity Considerations in Network Solution by Tearing", IEEE Transaction on Circuits and Systems, vol. CAS-27, pp. 357-366, May 1980.

[7.17] G. Hachtel, R. Brayton and F. Gustavson, "The sparse tableau approach to network analysis and design", IEEE Trans. Circuit Theory, Vol. CT-18, pp. 101-113, Jan. 1971.

[7.18] G.D. Hachtel and A. Sangiovanni-Vincentelli, "A Survey of Third-Generation Simulation Techniques," Proc. IEEE, vol. 69, no. 10, October 1981.

[7.19] C-H. Ho, A.E. Ruehli and P.A. Brennan, "The Modified Nodal Approach to Network Analysis", IEEE Transactions on Circuits and Systems, vol. CAS-22, No.6, pp. 504-409, June 1975.

[7.20] T. Huang, "Generalization of the Implicit-Explicit Method for Floating Capacitors," Master Report, University of California, Berkeley, 1983.

[7.21] E. Isaacson and H.B. Keller, *Analysis of Numerical Methods*, New York: Wiley 1966.

[7.22] P.M. Lin, A.W. Nordsick, and H.Y. Hsieh "A Convergence-rate Investigation of a Multi-level Newton Algorithm for Large Circuit Analysis"
Proceeding of 26th Midwest Symposium on Circuits and Systems, August 1983.

[7.23] L.N. Nagel, "SPICE 2: A Computer Program to Simulate Semiconductor Circuits", ERL Memo No. ERL-M520, University of California, Berekely, CA, May 1975.

[7.24] A.R. Newton, "Techniques for the Simulation of Large-Scale Integrated Circuits," IEEE Trans. on Circ. and Sys., vol. CAS-26, pp. 741-749, September 1979.

[7.24b]A.R. Newton, "The Analysis of Floating Capacitors for Timing Simulation," in Proc. 13th Asilomar conf. Circuits Syst. Comput., November 1979.

[7.26] A. Newton and A. Sangiovanni-Vincentelli, "Relaxation-based Electrical Simulation," IEEE Trans. El. Dev., No. 9, pp. 1186-1207, September 1983.

[7.27] J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables* , Academic Press, 1970.

[7.28] G. Persky, D.N. Deutsch and D.G. Schweikert, "LTX-A System for the Directed Automation Design of LSI Circuits," in Proc. 13th Design Automation Conf., 1976.

[7.29] N. Rabbat, A.E. Ruehli, G.W. Mahoney and J.J. Coleman, "A Survey of Macromodeling," IEEE Int. Symp. Circuits Systems, pp. 139-143, April 1985.

[7.30] N. Rabbat and H.Y. Hsieh, "A Latent Macromodular Approach to Large-Scale Sparse Networks," IEEE Trans. Cricuit Systems, vol. CAS-22, pp. 745-752, December 1976.

[7.31] N.B.G. Rabbat, A. Sangiovanni-Vincentelli and H.Y. Hsieh, "A Multilevel Newton Algorithm with Macromodeling and Latency for the Analysis of Large-Scale Nonlinear Circuits in the Time Domain", IEEE Transactions on Circuits and Systems, vol. CAS-26, no. 9, pp. 733-740, September 1979.

[7.32] A.E. Ruehli G. Rabbat and H.Hsieh, "Macromodeling - an approach for analyzing large scale circuits" CAD, Vol. 10, No. 2, pp.121-130, March 1978.

[7.33] A. Ruehli, A. Sangiovanni-Vincentelli and G. Rabbat, "Time Analysis of Large-Scale Circuits Containing One-Way Macromodels, IEEE Transaction of Circuits and Systems, vol. CAS-29, pp. 185-191, March 1982.

[7.34] A.E. Ruehli and G. Ditlow, "Circuit Analysis, Logic Simulation and Design Verification for VLSI," Proc. IEEE, vol. 71, pp. 36-68, January 1983.

[7.35] R. Saleh, "Iterated Timing Analysis and SPLICE1," M.S. Thesis, Univ. of California, Berkeley, 1983.

[7.36] A. Sangiovanni-Vincentelli, L.K. Chen and L.O. Chua, "Node-Tearing, Nodal Analysis", Electronics Research Lab., Univ. of California, Berkeley, Memo No ERL-M582, September 1976.

[7.37] A.Sangiovanni Vincentelli, Li-Kuan Chen and L.O.Chua, "An Efficient Cluster Algorithm for Tearing Large-Scale Networks", IEEE Trans. on Circ. and Syst., vol CAS-24, no. 12, pp. 709-717, dec. 1977.

[7.38] S.A. Szygenda and E.W. Thompson, "Digital logic Simulation in a Time-Based, Table-Driven Environment. Part 1. Design Verification," Computer, March 1975, pp. 24-36.

[7.39] E.G. Ulrich, "Time Sequenced Logical Simulation Based on Circuit Delay and Selective Tracing of Active Network Path," in Proc. ACM 20th Nat. Conf., pp. 437-448, 1965.

[7.40] R.S. Verga, *Matrix Iterative Analysis* , Prentice-Hall, Inc., New Jersey, 1962.

[7.41] W.T. Weeks, A.J. Jiminez, G.W. Mahoney, D. Mehta, H. Qassemzadeh and T.R. Scott, "Algorithms for ASTAP-A Network Analysis Program," IEEE Trans. Circuit Theory, vol. CT-20, pp. 628-634, November 1973

[7.42] J. White, F. Odeh, A. Sangiovanni and A. Ruehli "Waveform Relaxation: Theory and Practice," Transaction of the Society for Computer Simulation, Vol. 2, No.1 pp. 95-133, 1985.

[7.43] F.F. Wu, "Solution of Large-Scale Networks by Tearing", IEEE Transactions on Circuits and Systems, vol. CAS-23, pp.706-713, December 1976.

[7.44] P. Yang, I.N. Hajj, and T.N. Trick, "SLATE: A Circuit Simulation Program with Latency Exploitation and Node Tearing:, Proceeding of IEEE Int. Conf. Circuits and Computers, Port Chester, NY, October 1980, pp. 353-355.