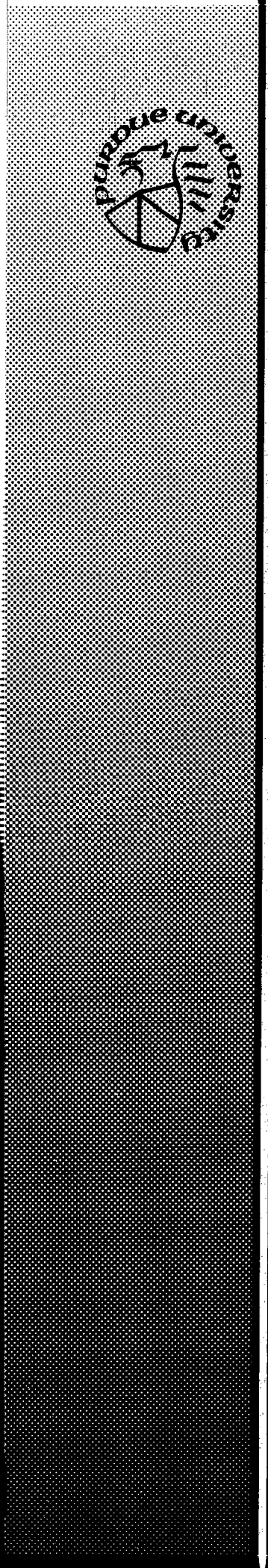
2-1-1985

Vectorized Circuit

Yi-Xiang Wang
Purdue University

Kai Hwang
Purdue University



Vectorized Circuit Analysis Using a Modified Newton Algorithm

Yi-Xiang Wang
Kai Hwang

TR-EE 85-03
February 1985

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

This research was supported by an IBM research grant to Purdue University

VECTORIZED CIRCUIT ANALYSIS USING A MODIFIED NEWTON ALGORITHM ON THE CYBER-250 SUPERCOMPUTER*

Yi-Xiang Wang
Kai Hwang

TR-EE 85-03

February 1985

*This research was supported fully by an IBM research grant to Purdue University from Nov. 1982 to Nov. 1985.

TABLE OF CONTENTS

Page

ABSTRACT

In this report, a newly modified Newton algorithm (MNA) and a data structure for sparse matrix manipulation are presented for analyzing large-scale electronic circuits on the Cyber-205 supercomputer. The MNA is improved from the Multilevel Newton Algorithm (MLNA) developed by Rabbat, Sanjiovanni-Vincentelli, and Hsieh (1979). The time complexity and convergence rate of MNA are analyzed. The computation steps are shown in detail by some example circuits. Scalar and vectorized simulation programs have been tested run on a VAX 11/780 scalar machine and on the Cyber 205 vector processor at Purdue University. From the results obtained, we observe that the MNA results a speedup of about 100 on the Cyber-205 as compared with using a scalar computer to analyze an electronic circuit containing 500 identical subcircuits.

CHAPTER 1
INTRODUCTION

This introductory chapter describes the problem environment and outlines the paper organization and research contributions. Related previous works are briefly reviewed.

1.1 Circuit Analysis Methodologies

Digital computers have been used widely in large-scale circuit analysis. This report presents a *Modified Newton Algorithm* (MNA) for circuit analysis. The supercomputer Cyber-205 is used for analyzing large-scale electronic circuits with this new algorithm. In the time-domain, a nonlinear lumped circuit system is characterized by a set of differential equations. [1] [5]

$$\mathbf{f}(\mathbf{u}(t), \dot{\mathbf{u}}(t), t) = \mathbf{0} \quad T \geq t \geq 0 \quad , \tag{1.1}$$

where $\mathbf{u}(t) \in \mathbf{R}^p$ is a vector of node voltages, or branch currents, or capacitor charges, or inductor fluxes, and $\mathbf{0}$ is the origin in $\mathbf{R}^p$, The mapping, $\mathbf{f} : \mathbf{R}^p \times \mathbf{R}^p \times \mathbf{R}^1 \rightarrow \mathbf{R}^p$, is a differential function with respect to $\mathbf{u}(t)$ and $\dot{\mathbf{u}}(t)$, On a digital computer, the *Backward Differential Formula* (BDF) [16] can be used to discretize the operator $\frac{d}{dt}$. The BDF of order k is defined by:

$$-h\, \dot{\mathbf{u}}_{n+1} = \sum_{i=0}^{k} \alpha_i\, \mathbf{u}_{n+1-i} \quad , \tag{1.2}$$

where $\dot{\mathbf{u}}_{n+1}$ is the computed value of $\dot{\mathbf{u}}(t_{n+1})$, and $\mathbf{u}_{n+1-i}$ is the computed value of $\mathbf{u}(t_{n+1-i})$, for $i = 0, 1, \cdots k$. The time increment $h = t_{n+1} - t_n$, and the α_i's are selected such that Eq 1.2 is exact for polynomials of the degree $\leq k$. Substituting Eq 1.2 at $t = t_{n+1}$ into Eq 1.1, we obtain

$$\mathbf{f}(\mathbf{u}_{n+1}, \mathbf{u}_n, \cdots \mathbf{u}_{n+1-k}, t_{n+1}) = 0 \quad 0 \leq t_{n+1} \leq T \tag{1.3}$$

Since the k past values $\mathbf{u}_n, \ldots \mathbf{u}_{n+1-k}$ are known at time t_{n+1}, Eq.1.3 becomes a function of $\mathbf{u}_{n+1}$. Then Eq 1.3 can be written as

$$\mathbf{F}_{n+1}(\mathbf{u}_{n+1}) = 0 \tag{1.4}$$

Where $\mathbf{F}_{n+1} : \mathbf{R}^p \rightarrow \mathbf{R}^p$ is continuously differentiable, and the index $n+1$

indicates different time instants. Then a digital computer can be used to solve a nonlinear circuit by Eq.1.4 . For example, a linear capacitor is characterized by:

$$C \cdot \frac{dV}{dt} = I \tag{1.5}$$

or

$$\frac{dV}{dt} = \frac{I}{C} \tag{1.6}$$

Where V is the voltage across the capacitor, I is the current through it, and C is the capacitance. Using the BDF to discretize Eq.1.6, we obtain

$$V_{n+1} = V_n + h \cdot \frac{I_{n+1}}{C} \tag{1.7}$$

or

$$I_{n+1} = \frac{C}{h} V_{n+1} - \frac{C}{h} V_n \tag{1.8}$$

Using Eq.1.8, at time t_{n+1}, the capacitor is equivalent to a resistor and a current source, called an associated discrete circuit [5]. Figure 1.1 shows this equivalence for a linear capacitor at time t_{n+1}. After such an equivalence, there will be no time dependent elements in the circuit. Only linear resistors, nonlinear resistors, independent and controlled sources appear. Then the circuit can be solved by the *Newton Raphson Algorithm* at time t_{n+1} for $0 \leq t_{n+1} \leq T$.

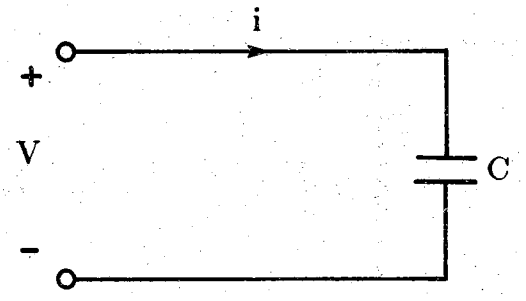
There are several algorithms for solving the nonlinear equation defined in Eq.1.4, such as *Single Level Newton Algorithm* (SLNA) and *Multilevel Newton Algorithm* (MLNA). This report presents a newly modified Newton algorithm to analyze large-scale circuits and studies the speedup from code vectorization.

1.2 Circuit Analysis On A Supercomputer

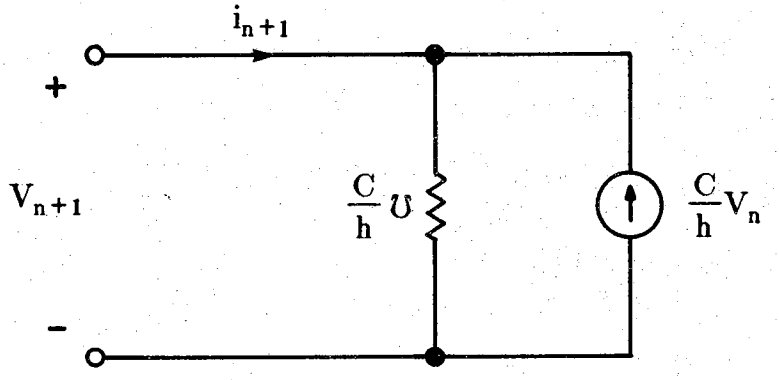
Circuit analysis requires to solve the linear system of equations:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{1.9}$$

When the size of a circuit is large, the matrix **A** becomes very large and very sparse. Since the additions and multiplications with the zero operands are redundant, avoiding them may gain speedup and save memory space. There exist several techniques for sparse matrix manipulation, such as using a row-column pointer structure and bit matrix mask structure as described in [2]. For the row-column pointer method, the nonzero elements of **A** are stored



(a)



(b)

Figure 1.1 A linear capacitor and equivalent circuit

rowwise in increasing order. We denote this vector as NZ . This vector has length m, where

$$m = p \cdot n^2 \tag{1.10}$$

The parameter n indicates the dimension of the matrix **A**, and p is the percentage of nonzero elements in **A**. This method needs to use two extra integer arrays to locate the nonzero elements. One array, called the row identifier array IUR, has length n and contains the location of the first nonzero element of each rows of **A**. Another array, called the column identifier array IUL, has length m and contains the corresponding column numbers. Adding or multiplying the nonzero elements need to access NZ, IUR and IUL for locating the nonzero elements in **A**. These operations need extra CPU time beyond the regular addition or multiplication times. This method is used only when **A** is small.

The second method uses a bit mask matrix **B** to replace the vectors IUR and IUL to locate the nonzero elements. The matrix **B** has the same dimension as **A**. Each entry in **B** has only one bit, with a value 1 for a nonzero element in the corresponding position of **A**, and a value 0 for a zero element. For a computer which has the capability of bit processing like Cyber-205, memory space can be saved when this technique is used. However, to locate a nonzero element requires to count the number of 1's in **B** from the beginning. When the dimension of **B** is very large, the counting may become very time consuming.

The third method uses the same bit mask matrix **B** as in the second method. An integer array R is used to indicate the first nonzero element in each row of **A**. For example, R(i)=j means that NZ(j) is the first nonzero element in the i-th row of **A**. Although this method needs a little extra memory to store the vector R, only one row of 1's in **B** needs to be counted at one time. So it can reduce the addressing time from $O(n^2)$ to O(n).

The supercomputer Cyber-205 at Purdue University has two vector arithmetic pipelines and a bit masking pipeline. It has a complete set of instructions for bit processing. Therefore the Cyber-205 is very suitable to implement the modified-bit-matrix method for manipulating very large and sparse matrices.

1.3 Organization And Contributions

A new algorithm, MNA, is proposed in this paper which is developed from the SLNA and MLNA. The SLNA and MLNA have some problems when used

for solving a very large scale circuit. The MNA is developed to overcome these problems. A given circuit is partitioned into a main circuit and many subcircuits in our approach. These subcircuits are treated as a vector and are solved by a pipeline supercompute efficiently. The nonlinear equations are not used here. And we do not have to solve the Jacobian matrices as in MLNA.

Solving a large-scale circuit partitioned into l-levels, demands l-levels of Newton loop in MLNA (one main loop and l-1 inner loop). So the number of iterations in MLNA increase as an exponential function of the number of levels. There is no inner loop in MNA. The number of iterations in MNA is a constant. The MNA has quadratic convergence in most cases, which is faster than the "Pairwise quadratic convergence" in the MLNA. Using the MNA to perform circuit simulation experiments on Cyber-205, significant CPU time can be saved.

Chapter 2 explains the MNA and compares it with the SLNA and MLNA in their relative merits. Mathematical proofs of MNA are given there. In Chapter 3, two examples are used to illustrate the computational steps in the MNA. The complexity and convergence of MNA are then analyed. Chapter 4 shows sparse matrix techniques for solving large matrices on supercomputer, and illustrates how to vectorize the MNA . The scalar version of programs are also explained. In Chapter 5, the Cyber-205 is used to solve large-scale circuit examples by various program versions. The scalar computer VAX 11/780 is used as a reference machine to solve the same problems. The results are presented based on simulation experiments. The speedup of each computation step is shown by some curves. Conclusions and suggestions are given in Chapter 6. The input data sets, numerical results from the simulations and three versions of circuit simulation programs are attached in the Appendices.

CHAPTER 2
THE MODIFIED NEWTON ALGORITHM

This chapter reviews the Single-Level and Multilevel Newton Algorithms and presents the new algorithm, MNA, for large-scale circuit simulation on a vector processing supercomputer

2.1 Single Level And Multilevel Newton Algorithms

The *Single Level Newton Algorithm* (SLNA) has been used to analyse the circuits widely. Since the complexity of solving a circuit characterized by an n×n matrix is O(n^3). It will require many hours to solve a large system with more than several thousand unknowns. Moreover, the main memory in most of today's computer does not have enough space to hold the entire data base. The *tearing techniques* are used to overcome these problems [17]. In the SLNA, a set of nonlinear equations $\mathbf{F(X)} = 0$ is used to characterize a nonlinear circuit. In the j-th Newton iteration, we have

$$\mathbf{x}^{j+1} = \mathbf{x}^{j} - \left[\frac{d\mathbf{F}(\mathbf{x})}{d\mathbf{x}} \right]^{-1}_{x=x^j} \mathbf{F}(\mathbf{x}^{j})$$

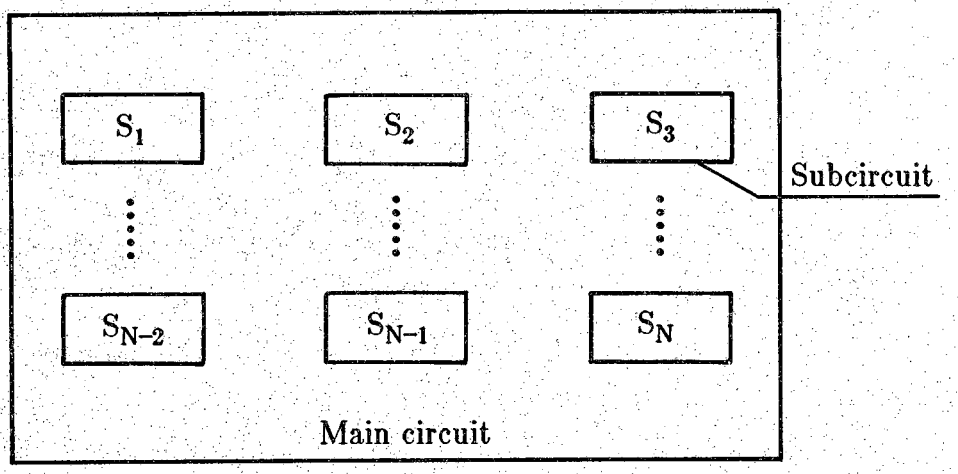
or

$$\left[\frac{d\mathbf{F}(\mathbf{x})}{d\mathbf{x}} \right]_{x=x^j} \Delta\mathbf{x}^{j} = -\mathbf{F}(\mathbf{x}^{j}) \qquad (2.1)$$

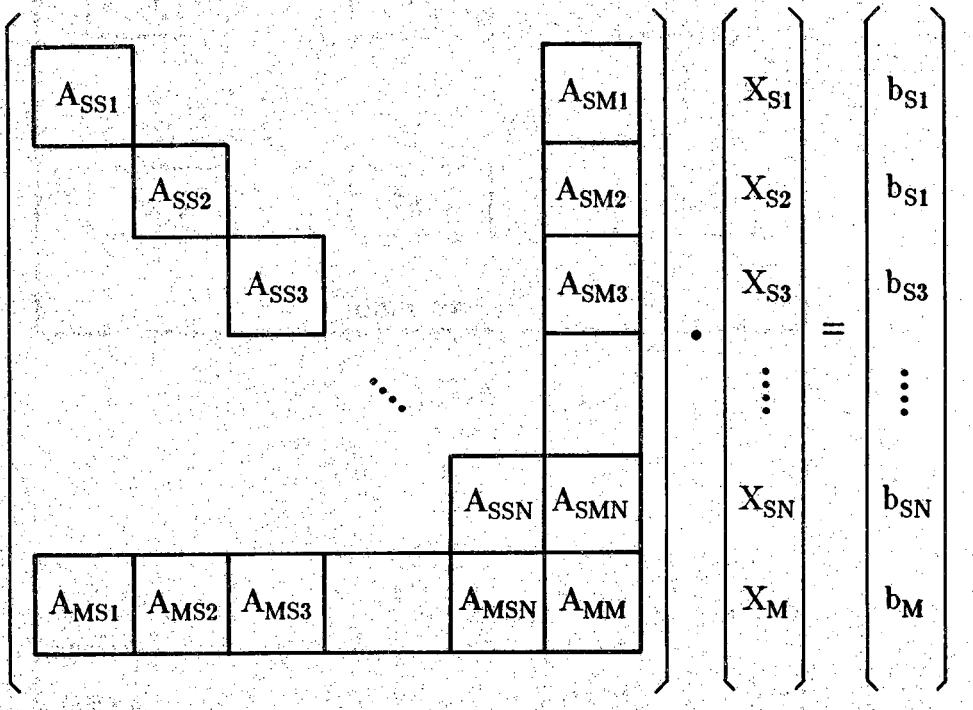
Where $\mathbf{x}^j$ is an unknown vector in the j-th iteration, the $\left[\frac{d\mathbf{F}(\mathbf{x})}{d\mathbf{x}} \right]$ is the Jacobian matrix of $\mathbf{F}(\mathbf{x})$, and

$$\Delta\mathbf{x}^{j} = \mathbf{x}^{j+1} - \mathbf{x}^{j} \qquad (2.2)$$

In general, a large-scale digital circuit or memory array may have many repeated subcircuits with the same structure as shown in Fig.2.1a. Assume all subcircuits are only connected to the main circuit, and no connections each other, the matrix in Eq.2.1 will have a block-diagonal form as in Fig.2.1b. In the tearing technique, this matrix is partitioned into one main matrix and some submatrices that can be solved separately. There are five steps in each Newton iteration:



(a)



(b)

Figure 2.1 A main circuit with many identical subcircuits

1) **LU** decomposition of submatrices

$$\mathbf{A}_{ssi} = \mathbf{L}_{ssi}\mathbf{U}_{ssi}$$

2) Solve for

$$\mathbf{R}_i \text{ from } \quad \mathbf{L}_{ssi}\mathbf{R}_i = \mathbf{A}_{smi}$$

$$\mathbf{T}_i \text{ from } \quad \mathbf{T}_i\mathbf{U}_{ssi} = \mathbf{A}_{msi}$$

$$\mathbf{c}_i \text{ from } \quad \mathbf{L}_{ssi}\mathbf{c}_i = \mathbf{b}_{si}$$

3) Matrix multiplications

$$\mathbf{A}^*_{mi} = \mathbf{R}_i\mathbf{T}_i$$

$$\mathbf{b}^*_{mi} = \mathbf{T}_i\mathbf{c}_i$$

4) Solving the main matrix

$$\left[\mathbf{A}_{mm} - \sum_{i=1}^{N}\mathbf{A}^*_{mi}\right]\mathbf{x}_m = \left[\mathbf{b}_m - \sum_{i=1}^{N}\mathbf{b}^*_{mi}\right]$$

5) Back-substitutions using the submatrices

$$\mathbf{L}_{ssi}\mathbf{y}_{si} = [\,\mathbf{b}_{si} - \mathbf{A}_{smi}\mathbf{x}_m\,]$$

$$\mathbf{U}_{ssi}\mathbf{x}_{si} = \mathbf{y}_{si}$$

Where from step 1 to 3 are used for solving the i-th submatrix $\mathbf{A}_{ssi}$. Assume the dimension of the submatrices is ns, the time complexities in the first three steps are about $\frac{1}{3}ns^3$, ns^3 and ns^3. The calculating in step 2 and 3 may take a long time when the submatrices are large. If there are N submatrices, they can be solved in parallel, and the total time complexity is estimated to be:

$$N(\frac{1}{3}ns^3 + ns^3 + ns^3) = \frac{7}{3}N{\cdot}ns^3$$

Step 4 is for solving the main matrix. Let n is the dimension of the main matrix, then the time complexity in step 4 is about $\frac{1}{3}n^3$, and the time complexity in step 5 is about $N{\cdot}ns^2$. Assume the number of the Newton iterations for this approach is p, the total time complexity for tearing technique SLNA will be about

$$p(\frac{7}{3}N{\cdot}ns^3 + \frac{1}{3}n^3) = \frac{p}{3}(7N{\cdot}ns^3 + n^3) \qquad (2.3)$$

Another approach is called *Multilevel Newton Algorithm* (MNLA) which use a inner Newton loop to solve the subcircuits instead of the step 1 to 3 mentioned above. We can use

$$\mathbf{F}(\mathbf{U}, \mathbf{Y}, \omega) = 0 \tag{2.4}$$

to formulate a main circuit, and use

$$\mathbf{H}_i(\mathbf{U}_i, \mathbf{Y}_i, \mathbf{X}_i) = 0 \tag{2.5}$$

to formulate the i-th subcircuit $\mathbf{S}_i$ in the circuit. The $\mathbf{F}$ and $\mathbf{H}_i$ are sets of nonlinear equations, the $\mathbf{U}$, $\mathbf{Y}$, ω, $\mathbf{U}_i$, $\mathbf{Y}_i$, and $\mathbf{X}_i$ are vectors, $\mathbf{U}$ is the outputs of the main circuit, $\mathbf{Y}$ is the inputs of the main circuit, ω is the inner variables of main circuit, $\mathbf{U}_i$ and $\mathbf{Y}_i$ are inputs and outputs of the i-th subcircuit, and $\mathbf{U}_i \in \mathbf{U}$, $\mathbf{Y}_i \in \mathbf{Y}$, $\mathbf{X}_i$ are the inner variables of the i-th subcircuit.

At the j-th iteration of the main loop in MLNA, one has to solve the following equation:

$$\left[\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}} + \frac{\partial \mathbf{F}}{\partial \mathbf{Y}} \frac{\partial \mathbf{Y}}{\partial \mathbf{U}} \right), \frac{\partial \mathbf{F}}{\partial \omega} \right] \begin{bmatrix} \Delta \mathbf{U}^j \\ \Delta \omega^j \end{bmatrix} = -\mathbf{F}(\mathbf{U}^j, \mathbf{Y}^j, \omega^j) \tag{2.6}$$

$$\Delta \mathbf{U}^j = \mathbf{U}^{j+1} - \mathbf{U}^j$$

$$\Delta \omega^j = \omega^{j+1} - \omega^j$$

Differentiating [5] [14], we have

$$\frac{\partial \mathbf{F}}{\partial \mathbf{U}} = \begin{bmatrix} \frac{\partial F_1}{\partial U_1} & \frac{\partial F_1}{\partial U_2} & \cdots & \frac{\partial F_1}{\partial U_k} \\ \frac{\partial F_2}{\partial U_1} & \frac{\partial F_2}{\partial U_2} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial F_n}{\partial U_1} & \frac{\partial F_n}{\partial U_2} & \cdots & \frac{\partial F_n}{\partial U_k} \end{bmatrix} \tag{2.7}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{Y}} = \begin{bmatrix} \frac{\partial F_1}{\partial Y_1} & \frac{\partial F_1}{\partial Y_2} & \cdots & \frac{\partial F_1}{\partial Y_k} \\ \frac{\partial F_2}{\partial Y_1} & \frac{\partial F_2}{\partial Y_2} & \cdots & \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial F_n}{\partial Y_1} & \frac{\partial F_n}{\partial Y_2} & \cdots & \frac{\partial F_n}{\partial Y_k} \end{bmatrix} \tag{2.8}$$

$$\frac{\partial \mathbf{F}}{\partial \boldsymbol{\omega}} = \begin{bmatrix} \frac{\partial F_1}{\partial \omega_1} & \frac{\partial F_1}{\partial \omega_2} & \cdots & \frac{\partial F_1}{\partial \omega_m} \\ \frac{\partial F_2}{\partial \omega_1} & \frac{\partial F_2}{\partial \omega_2} & \cdots & \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial F_n}{\partial \omega_1} & \frac{\partial F_n}{\partial \omega_2} & \cdots & \frac{\partial F_n}{\partial \omega_m} \end{bmatrix} \tag{2.9}$$

$$\frac{\partial \mathbf{Y}}{\partial \mathbf{U}} = \begin{bmatrix} \frac{\partial Y_1}{\partial U_1} & \frac{\partial Y_1}{\partial U_2} & \cdots & \frac{\partial Y_1}{\partial U_k} \\ \frac{\partial Y_2}{\partial U_1} & \frac{\partial Y_2}{\partial U_2} & \cdots & \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial Y_l}{\partial U_1} & \frac{\partial Y_l}{\partial U_2} & \cdots & \frac{\partial Y_l}{\partial U_k} \end{bmatrix} \tag{2.10}$$

Since the nonlinear equations $\mathbf{F(X)}$ is known, the Jacobian matrix $[\frac{\partial \mathbf{F}}{\partial \mathbf{U}}]$, $[\frac{\partial \mathbf{F}}{\partial \mathbf{Y}}]$ and $[\frac{\partial \mathbf{F}}{\partial \omega}]$ in Eq.2.6 can be obtained directly. But the Jacobian matrix $[\frac{\partial \mathbf{Y}}{\partial \mathbf{U}}]$ can be obtained only after all of the subcircuits are solved. Fortunately the subcircuits are only connected to the main circuit and no connections are between each other. The internal variables in different subcircuits are not related. Therefore the entries in Eq.2.10 satisfy the following property:

$$\frac{\partial Y_p}{\partial U_q} = 0$$

When Y_p and U_q are not in the same subcircuit, $\frac{\partial Y_p}{\partial U_q}$ becomes zero. So we can separately calculate these nonzero entries in $[\ \frac{\partial \mathbf{Y}}{\partial \mathbf{U}}\ ]$ as follows:

Assume the i-th subcircuit is characterized by Eq.2.5. Since the elements of $\mathbf{U}_i$ are calculated from the main circuit in last iteration, the Eq.2.5 can be written as

$$\mathbf{H}_i(\ \mathbf{X}_i\ ,\ \mathbf{Y}_i\ ) = 0 \tag{2.11}$$

Then another Newton algorithm loop (inner loop) is needed to solve the $\mathbf{X}_i$ and $\mathbf{Y}_i$. In this inner loop we have

$$\left[\ \frac{\partial \mathbf{H}}{\partial \mathbf{X}}\ ,\ \frac{\partial \mathbf{H}}{\partial \mathbf{Y}}\ \right]\cdot\begin{bmatrix}\Delta \mathbf{X}^j\\ \Delta \mathbf{Y}^j\end{bmatrix} = -\mathbf{H}(\ \mathbf{X}^j\ ,\ \mathbf{Y}^j\ ) \tag{2.12}$$

After this loop we obtain the unknowns in the subcircuit $\mathbf{X}_i$, $\mathbf{Y}_i$, Moreover, from the Eq.2.5, we have

$$\left[\ \frac{\partial \mathbf{H}}{\partial \mathbf{U}} + \frac{\partial \mathbf{H}}{\partial \mathbf{X}}\ \frac{\partial \mathbf{X}}{\partial \mathbf{U}} + \frac{\partial \mathbf{H}}{\partial \mathbf{Y}}\ \frac{\partial \mathbf{Y}}{\partial \mathbf{U}}\ \right]_{X=X_i\ Y=Y_i} = 0$$

or

$$\left[\ \frac{\partial \mathbf{H}}{\partial \mathbf{X}}\ ,\ \frac{\partial \mathbf{H}}{\partial \mathbf{Y}}\ \right]_{X=X_i\ Y=Y_i}\cdot\begin{bmatrix}\frac{\partial \mathbf{X}}{\partial \mathbf{U}}\\ \frac{\partial \mathbf{Y}}{\partial \mathbf{U}}\end{bmatrix} = -\frac{\partial \mathbf{H}}{\partial \mathbf{U}} \tag{2.13}$$

Where

$$\frac{\partial \mathbf{H}}{\partial \mathbf{U}} = \begin{bmatrix}\frac{\partial H_1}{\partial U_1} & \frac{\partial H_1}{\partial U_2} & \cdots & \frac{\partial H_1}{\partial U_c}\\ \frac{\partial H_2}{\partial U_1} & \frac{\partial H_2}{\partial U_2} & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ \cdot & \cdot & \cdots & \cdot\\ \frac{\partial H_{ns}}{\partial U_1} & \frac{\partial H_{ns}}{\partial U_2} & \cdots & \frac{\partial H_{ns}}{\partial U_c}\end{bmatrix}$$

$$\frac{\partial \mathbf{H}}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial H_1}{\partial X_1} & \frac{\partial H_1}{\partial X_2} & \cdots & \frac{\partial H_1}{\partial X_b} \\ \frac{\partial H_2}{\partial X_1} & \frac{\partial H_2}{\partial X_2} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial H_{ns}}{\partial X_1} & \frac{\partial H_{ns}}{\partial X_2} & \cdots & \frac{\partial H_{ns}}{\partial X_b} \end{bmatrix}$$

$$\frac{\partial \mathbf{H}}{\partial \mathbf{Y}} = \begin{bmatrix} \frac{\partial H_1}{\partial Y_1} & \frac{\partial H_1}{\partial Y_2} & \cdots & \frac{\partial H_1}{\partial Y_c} \\ \frac{\partial H_2}{\partial Y_1} & \frac{\partial H_2}{\partial Y_2} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial H_{ns}}{\partial Y_1} & \frac{\partial H_{ns}}{\partial Y_2} & \cdots & \frac{\partial H_{ns}}{\partial Y_c} \end{bmatrix}$$

Where ns is the dimension of the subsystem, b is the number of inner variables in the subsystem, c is the number of inputs and outputs in the subsystem. Substituting the results $\mathbf{X}_i$, $\mathbf{Y}_i$ from the inner loop into Eq.2.14, we obtain the values of $\frac{\partial \mathbf{Y}}{\partial \mathbf{U}}$, which is just needed by Eq.2.9 in main loop [5]. Therefore for each iteration of main loop, it needs to do a whole inner loop to solve the $\mathbf{X}_i$ and $\mathbf{Y}_i$.

Let the N, n, and ns are all as defined in SLNA, the time complexity for solving the main circuit and subcircuits are about $\frac{1}{3}n^3$ and $\frac{1}{3}ns^3$. Suppose a two-level circuit demands p iterations in the main loop, and q iterations in the inner loop. The total time complexity in MLNA is about

$$p\left(\frac{1}{3}qN{\cdot}ns^3 + \frac{1}{3}n^3\right) = \frac{p}{3}\left(qN{\cdot}ns^3 + n^3\right) \qquad (2.14)$$

In this approach, the inner loop is used to solve the subcircuits instead of solving some extra matrices and matrix multiplications, if q equals to 7, Eq.2.14 would be the same as Eq.2.3. Lin has proved that, if we do not apply the latency technique, the amount of computation in MLNA should be close to that in SLNA.

2.2 The Modified Newton Algorithm

The new algorithm, *Modified Newton Algorithm* (MNA) is proposed to improve the efficiencies of the SLNA and MLNA. This new algorithm reduces the calculation steps and number of iterations, and preserves all the advantages of the SLNA and MLNA. Sometimes, it even improve the convergency of the MLNA. Moreover, we do not need to use all the nonlinear equations $\mathbf{F}$ and $\mathbf{H}_i$, and can avoid solving any Jacobian matrices like those in Eq.2.6 and Eq.2.13.

In the SLNA , we can apply the Newton algorithm at the element level. That is to establish a associated discreted equivalent circuit to simulate the nonlinear circuit in the j-th iteration, and use the linear nodal equation to solve it [6] [7]. In the MNA, we use a set of independent sources and controlled sources to simulate the subcircuits at each iteration. Then we include these sources in the main circuit and use linear equations to solve them. Figure 2.2 shows the the flowchart of the MNA.

For example, consider N nonlinear subcircuits each with $c+1$ ports which are connected to a main circuit. The structure of the equivalent current sources for the subcircuit is as shown in Fig.2.3. There are c voltage controlled current sources $G_{si}^{j}(\mathbf{v}_s)$ which are functions of the port voltages $\mathbf{v}_s$, and c iterative independent current sources J_{si}^{j} in each equivalent subcircuit. The superscript j means that they have the different values at different iterations. The main circuit and these equivalent subcircuits can be solved by the *Associated Discrete Equivalent Circuit* (ADEC) method.

Any current at any port must be a function of the voltages across all ports of a subcircuit as shown in Fig.2.4a. Where $\mathbf{v}_s$ is the vector of voltages across all ports of this subcircuit, I_k is the current through the k-th port of the subcircuit. When this port is considered as one branch of the main circuit as shown in Fig.2.4b, we can use the nodal equation to solve the main circuit. The J_k and E_k characterize the independent sources in the main circuit, $\hat{v}_k$is the branch voltages in the main circuit, $\hat{I}$ is the branch current in the main circuit. Since the currents through all other devices in the main circuit are also the functions of the branch voltages. So they can be characterized by the same function $G_k(\mathbf{v}_s)$ as the subcircuits. For an example, if we replace a linear resistor in the main circuit by the active devices in Fig.2.4b, the J_k, E_k will become zero, and the G_k becomes only a constant.

According to the *Nodal Equations Method* (NEM) [6], the main circuit with the branches in Fig.2.4b, is described by:

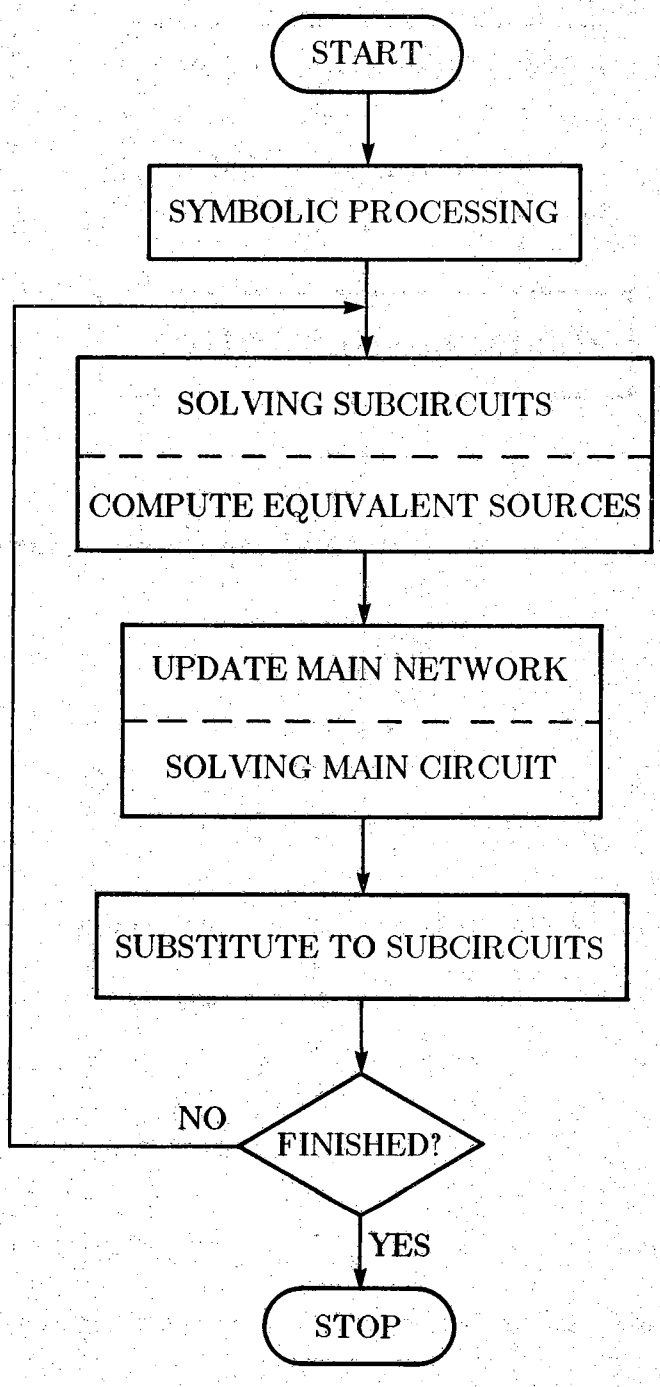


Figure 2.2 Flowchart of the MNA

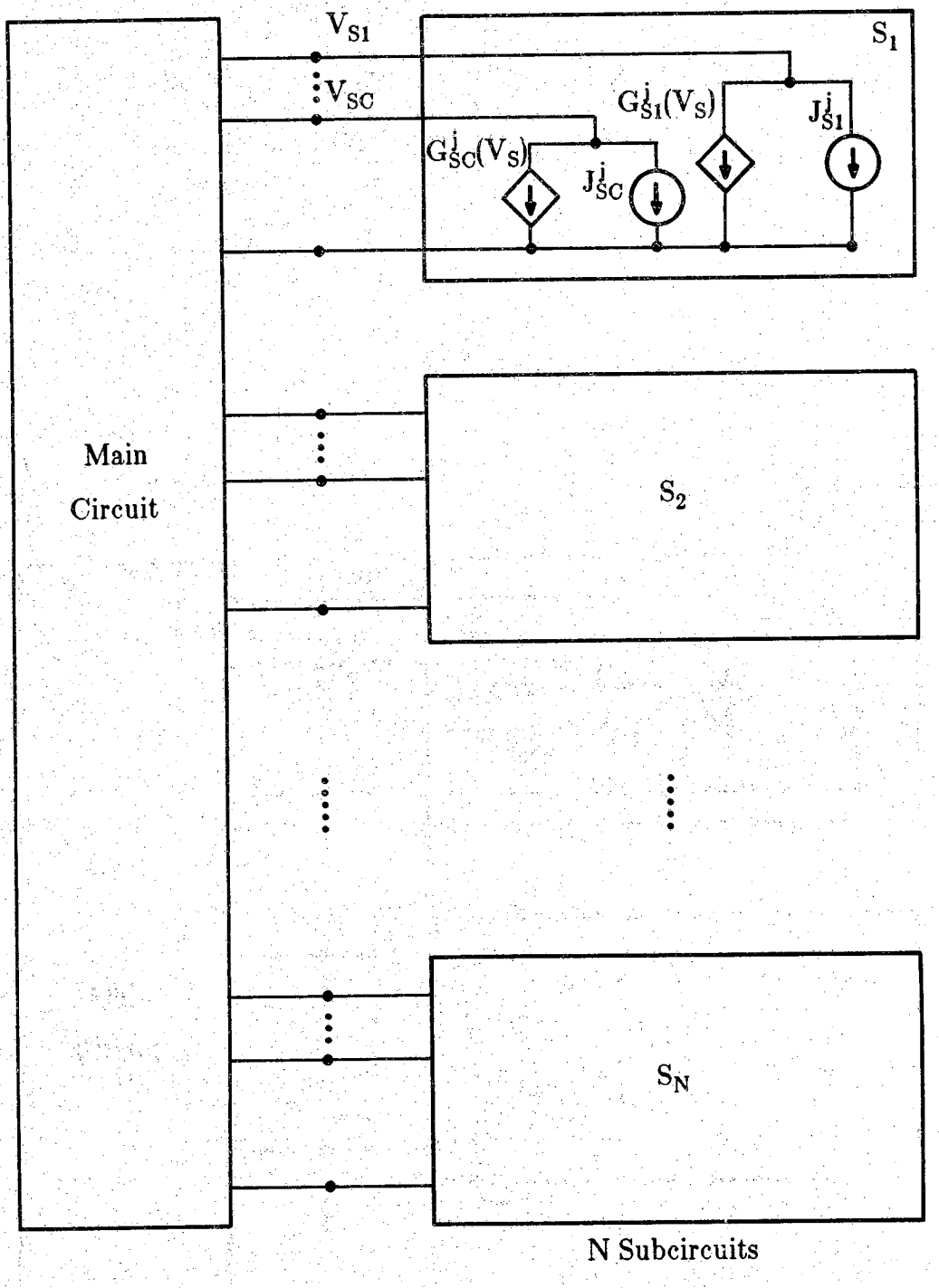
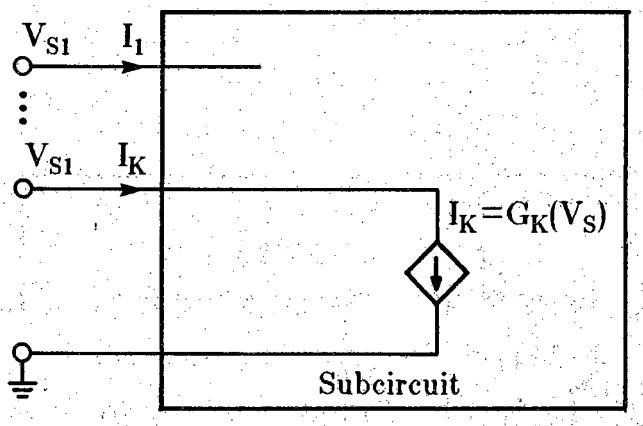
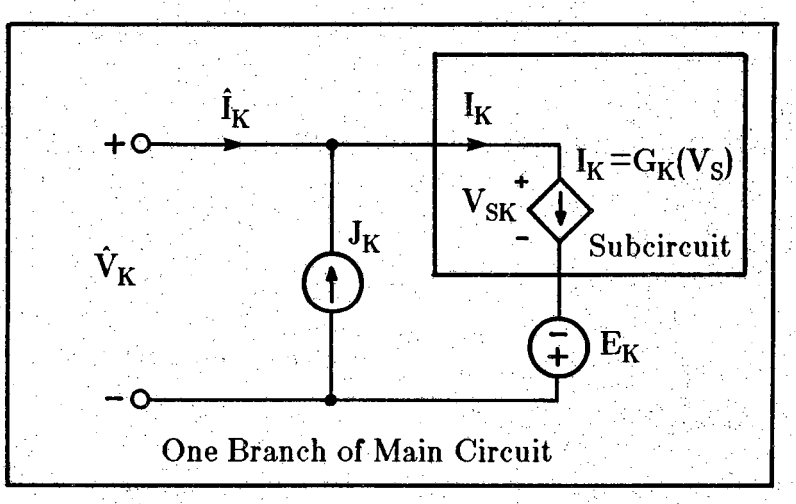


Figure 2.3 Equivalent current sources in subcircuits



(a)



(b)

Figure 2.4 The equivalent circuit of subcircuit ports

$$\mathbf{A}\,\mathbf{J} = \mathbf{A}\,\mathbf{I} = \mathbf{A}\,\mathbf{g}(\mathbf{v}_s) \tag{2.15}$$

Since

$$\mathbf{A}^t\mathbf{v}_n = \hat{\mathbf{v}} = \mathbf{v}_s - \mathbf{E} \tag{2.16}$$

$$\mathbf{v}_s = \mathbf{A}^t\mathbf{v}_n + \mathbf{E}$$

Equation 2.15 can be written as

$$\mathbf{A}\,\mathbf{g}(\mathbf{A}^t\mathbf{v}_n + \mathbf{E}) - \mathbf{A}\,\mathbf{J} = 0 \tag{2.17}$$

$\mathbf{A}$ is the reduced incidence matrix of the main circuit, $\mathbf{g}$ is a function of $(\mathbf{A}_t\mathbf{v}_n + \mathbf{E})$, $\mathbf{I}$ is the branch current vector, $\mathbf{J}$ is the independent current sources vector, and $\mathbf{E}$ is the independent voltage sources vector.

$$\mathbf{I} = \left[\mathrm{I}_1, \mathrm{I}_2, \cdots \mathrm{I}_n\right]^t$$

$$\mathbf{J} = \left[\mathrm{J}_1, \mathrm{J}_2, \cdots \mathrm{J}_n\right]^t$$

$$\mathbf{E} = \left[\mathrm{E}_1, \mathrm{E}_2, \cdots \mathrm{E}_n\right]^t$$

Where the n is the number of the nodes in the main circuit, $\mathbf{v}_n$ is the vector that indicates the voltages form all nodes in main circuit to the datum, $\mathbf{v}_s$ is the voltage vector of the subcircuit ports and the main circuit devices. Using the Newton Raphson algorithm to solve Eq.2.17, we obtain the equations at the j-th iteration:

$$\mathbf{v}_n^{j+1} = \mathbf{v}_n^j - \left[\mathbf{A}\,\frac{\partial \mathbf{g}(\mathbf{A}^t\mathbf{v}_n + \mathbf{E})}{\partial \hat{\mathbf{v}}}\,\frac{\partial \hat{\mathbf{v}}}{\partial \mathbf{v}_n}\right]^{-1}_{\hat{\mathbf{v}} = \hat{\mathbf{v}}^j}\left[\mathbf{A}\,\mathbf{g}(\mathbf{A}^t\mathbf{v}_n^j + \mathbf{E}) - \mathbf{A}\,\mathbf{J}\right]$$

$$= \mathbf{v}_n^j - \left[\mathbf{A}\,\frac{\partial \mathbf{g}(\mathbf{A}^t\mathbf{v}_n + \mathbf{E})}{\partial \hat{\mathbf{v}}}\,\mathbf{A}^t\right]^{-1}_{\hat{\mathbf{v}} = \hat{\mathbf{v}}^j}\left[\mathbf{A}\,\mathbf{g}(\mathbf{A}^t\mathbf{v}_n^j + \mathbf{E}) - \mathbf{A}\,\mathbf{J}\right] \tag{2.18}$$

and

$$\mathbf{A}^t\,\mathbf{v}_n^j + \mathbf{E} = \mathbf{v}_s^j$$

$$\mathbf{g}(\mathbf{A}^t\mathbf{v}_n^j + \mathbf{E}) = \mathbf{g}(\mathbf{v}_s^j) = \mathbf{I}^j \tag{2.19}$$

Where $\mathbf{v}_n^j$ is the voltage vector $\mathbf{v}_n$ in the j-th iteration, and $\mathbf{v}_s^j$ is the voltages across the subcircuits in j-th iteration, and $\mathbf{I}^j$ is the vector of currents through the subcircuits' ports in the j-th iteration. Let us define here:

$$\mathbf{Y}_s^j = \left[\frac{\partial g(\mathbf{A}^t \mathbf{v}_n + \mathbf{E})}{\partial \hat{\mathbf{v}}} \right]_{\hat{\mathbf{v}} = \hat{\mathbf{v}}^j} \tag{2.20}$$

The Jacobian matrix $\mathbf{Y}_s^j$ is the incremental conductance matrix in the j-th iteration. That is

$$\mathbf{Y}_s^j = \left[\begin{array}{cccc} \frac{\partial G_1}{\partial \hat{v}_1} & \frac{\partial G_1}{\partial \hat{v}_2} & \cdots & \frac{\partial G_1}{\partial \hat{v}_n} \\ \frac{\partial G_2}{\partial \hat{v}_1} & \frac{\partial G_2}{\partial \hat{v}_2} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial G_n}{\partial \hat{v}_1} & \frac{\partial G_n}{\partial \hat{v}_2} & \cdots & \frac{\partial G_n}{\partial \hat{v}_n} \end{array} \right]_{\hat{\mathbf{v}} = \hat{\mathbf{v}}^j} \tag{2.21}$$

G_k here are functions of $\mathbf{v}_s$. Then the Eq 2.18 can be written as

$$\mathbf{v}_n^{j+1} = \mathbf{v}_n^j - [\, \mathbf{A}\mathbf{Y}_s^j \mathbf{A}^t \,]^{-1} [\, \mathbf{A}\mathbf{I}^j - \mathbf{A}\mathbf{J} \,] \tag{2.22}$$

or

$$[\, \mathbf{A}\mathbf{Y}_s^j \mathbf{A}_t \,]\, \mathbf{v}_n^{j+1} = \mathbf{A} [\, \mathbf{J} - \mathbf{I}_j + \mathbf{Y}_s^j \mathbf{A}^t \mathbf{v}_n^j \,]$$

$$= \mathbf{A} [\, \mathbf{J} - \mathbf{I}^j + \mathbf{Y}_s^j (\mathbf{v}_s^j - \mathbf{E})] \tag{2.23}$$

Define:

$$\mathbf{J}_s^j = \mathbf{I}^j - \mathbf{Y}_s^j \mathbf{v}_s^j \tag{2.24}$$

Then the Eq 2.24 becomes

$$[\, \mathbf{A}\mathbf{Y}_s^j \mathbf{A}^t \,] \mathbf{v}_n^{j+1} = \mathbf{A} [\mathbf{J} - (\, \mathbf{I}^j - \mathbf{Y}_s^j \mathbf{v}_s^j \,) - \mathbf{Y}_s^j \mathbf{E} \,]$$

$$= \mathbf{A} [(\, \mathbf{J} - \mathbf{J}_s^j \,) - \mathbf{Y}_s^j \mathbf{E} \,] \tag{2.25}$$

Now, let us look back to the linear circuit. If we use the standard linear branch as shown in Fig.2.5 to replace the nonlinear branch in the same circuit as mentioned before. Then using the NEM to solve this linear circuit we have:

$$[\, \mathbf{A}\mathbf{Y}_b \mathbf{A}^t \,] \mathbf{v}_n = \mathbf{A} [\, \mathbf{J} - \mathbf{Y}_b \mathbf{E} \,] \tag{2.26}$$

Where the $\mathbf{A}$ is the reduced incidence matrix, $\mathbf{Y}_b$ is the conductance matrix, $\mathbf{v}_n$ is the nodal voltage vector, $\mathbf{J}$ is the current sources vector, $\mathbf{E}$ is the voltage sources vector. Here Eq.2.25 and Eq.2.26 have the similar structures. The only differences are that the conductance matrix $\mathbf{Y}_b$ in Eq.3.41 is being replaced by the incremental conductance matrix $\mathbf{Y}_s^j$, and the vector $\mathbf{J}$ is being replaced by

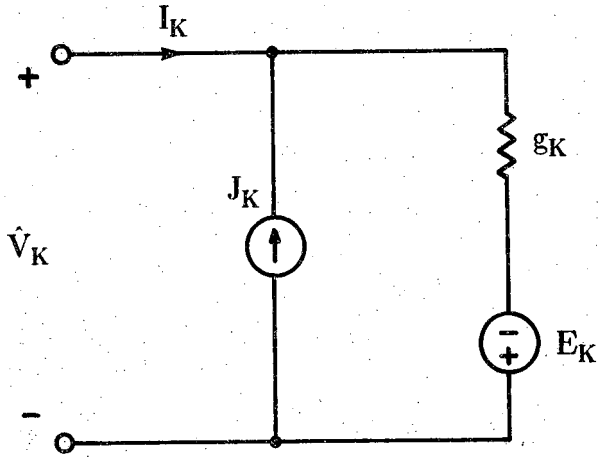


Figure 2.5 A branch in a linear circuit

$\mathbf{J} - \mathbf{J}_s^j$. These mean we can use some equivalent conversions for solving nonlinear circuit per iteration. After the conversion, all subcircuits and nonlinearity in the main circuit will be replaced by the linearized discrete equivalent circuit, Then the standard NEM can be used to solve the linearized circuit.

The discrete equivalent circuit for each branch is shown in Fig.2.6a , where J_{sk}^j is the entry of $\mathbf{J}_s^j$, G_{sk}^j is obtained from some entries in matrix $\mathbf{Y}_s^j$. Because the G_{sk}^j is the function of vector $\mathbf{v}_s$, it can not be characterized as a simple conductor but a current source controlled by the vector $\mathbf{v}_s$ as in Fig.2.6a. The circuit in Fig.2.6a can be reconstructed as in Fig.2.6b. By comparing the Fig.2.6b with the Fig.2.4b, we can known why we use two current sources to replace each port of the subcircuits in MNA.

From the Eq.2.21 and Eq.2.24 we have

$$\mathbf{J}_s^j = \mathbf{I}^j - \mathbf{Y}_s^j \mathbf{v}_s^j = \begin{bmatrix} I_1^j \\ I_2^j \\ \cdot \\ \cdot \\ \cdot \\ I_n^j \end{bmatrix} - \begin{bmatrix} \frac{\partial G_1}{\partial \hat{v}_1} & \frac{\partial G_1}{\partial \hat{v}_2} & \cdots & \frac{\partial G_1}{\partial \hat{v}_n} \\ \frac{\partial G_2}{\partial \hat{v}_1} & \frac{\partial G_2}{\partial \hat{v}_2} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial G_n}{\partial \hat{v}_1} & \frac{\partial G_n}{\partial \hat{v}_2} & \cdots & \frac{\partial G_n}{\partial \hat{v}_n} \end{bmatrix} \begin{bmatrix} v_{s1} \\ v_{s2} \\ \cdot \\ \cdot \\ \cdot \\ v_{sn} \end{bmatrix}$$

Since

$$\hat{\mathbf{v}} = \mathbf{v}_{sk} - \mathbf{E}_k$$

$$\mathbf{v}_{sk} = \hat{\mathbf{v}}_k + \mathbf{E}_k$$

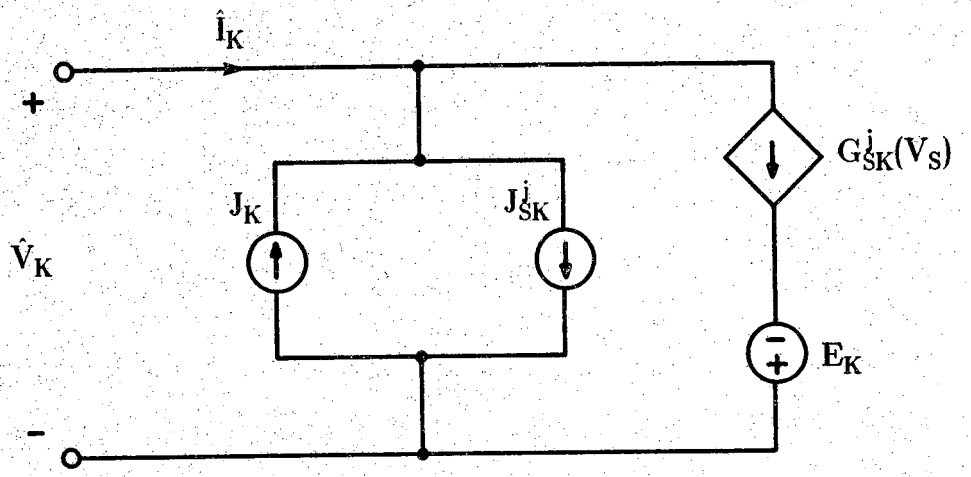
So the entries of $\mathbf{Y}_s^j$ becomes

$$\frac{\partial G_i}{\partial \hat{v}_k} = \frac{\partial G_i}{\partial v_{sk}} \frac{\partial v_{sk}}{\partial \hat{v}_k} = \frac{\partial G_i}{\partial v_{sk}}$$

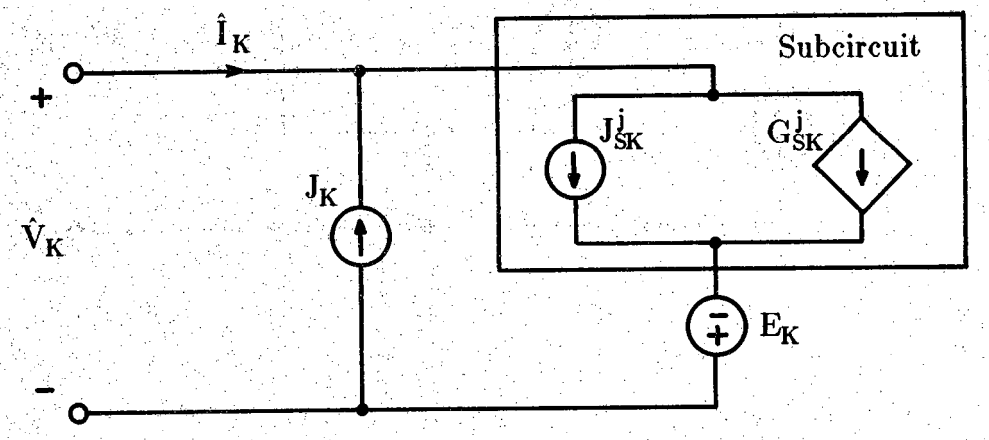
Then we have

$$G_{sk}^j = \left[\frac{\partial G_k}{v_{s1}}, \frac{\partial G_k}{\partial v_{s2}}, \cdots \frac{\partial G_k}{v_{sn}} \right] \tag{2.27}$$

and the J_{sk}^j can be calculated by



(a)



(b)

Figure 2.6 The equivalent circuit of a subcircuit

$$J_{sk}^{j} = I_{k}^{j} - \left[\frac{\partial G_k}{\partial v_{s1}}, \frac{\partial G_k}{\partial v_{s2}}, \cdots \frac{\partial G_k}{\partial v_{sn}} \right] \begin{bmatrix} v_{s1} \\ v_{s2} \\ . \\ . \\ v_{sn} \end{bmatrix} \tag{2.28}$$

Since the different subcircuits have no connections each other, the $\frac{\partial G_p}{\partial v_{sq}} = 0$ when G_p and v_{sq} are in different subcircuits. Therefore the parameters in different subcircuits can be calculated simultaneously. When there are many identical subcircuits in the main circuit, we can treat them as a vector and use a pipelined supercomputer to process them efficiently. The major difference between the MNA and the MLNA lies in the method of solving the subcircuits. When the values of G_s^j and J_s^j are calculated in the MNA, it does not use the differential equations $\mathbf{H}_i$. The equivalent circuit is used to simulating the subcircuits. Instead of a inner loop in MLNA, the simple NEM is used directly for obtaining the G_s^j and J_s^j.

For example, consider c+1 ports subcircuit shown in Fig.2.7a. Using the ADEC method, the equivalent circuit is obtained in Fig.2.7b . Using the MNA method, we obtain the port currents in the j-th iteration $\mathbf{I}^j$. The incremental conductance $G_{sk}^j(\mathbf{v}_s)$ is calculated in two steps.

First we set all the independent sources in Fig.2.7b to be zero, The resulting circuit is shown in Fig.2.7c. The input voltage v_1^j is applied to the subcircuit, and the other ports of the subcircuit are connected to datum as shown in Fig.2.8a. Solving it we can obtain the subcircuit incremental current $\mathbf{I}_1^j$.

$$\mathbf{I}_1^j = \left[I_{11}^j, I_{12}^j, \cdots I_{1c}^j \right]^t$$

Then applying the input voltage v_2^j to the subcircuit as shown in Fig.2.8b, we obtain another incremental current $\mathbf{I}_2^j$.

$$\mathbf{I}_2^j = \left[I_{21}^j, I_{22}^j, \cdots I_{2c}^j \right]^t$$

After we apply the voltages from v_{s1} to v_{sc} to the subcircuit, the current vectors from $\mathbf{I}_1^j$ to $\mathbf{I}_c^j$ can be obtained accordingly.

Secondly we use these incremental currents to calculate the corresponding incremental conductance in the j-th iteration:

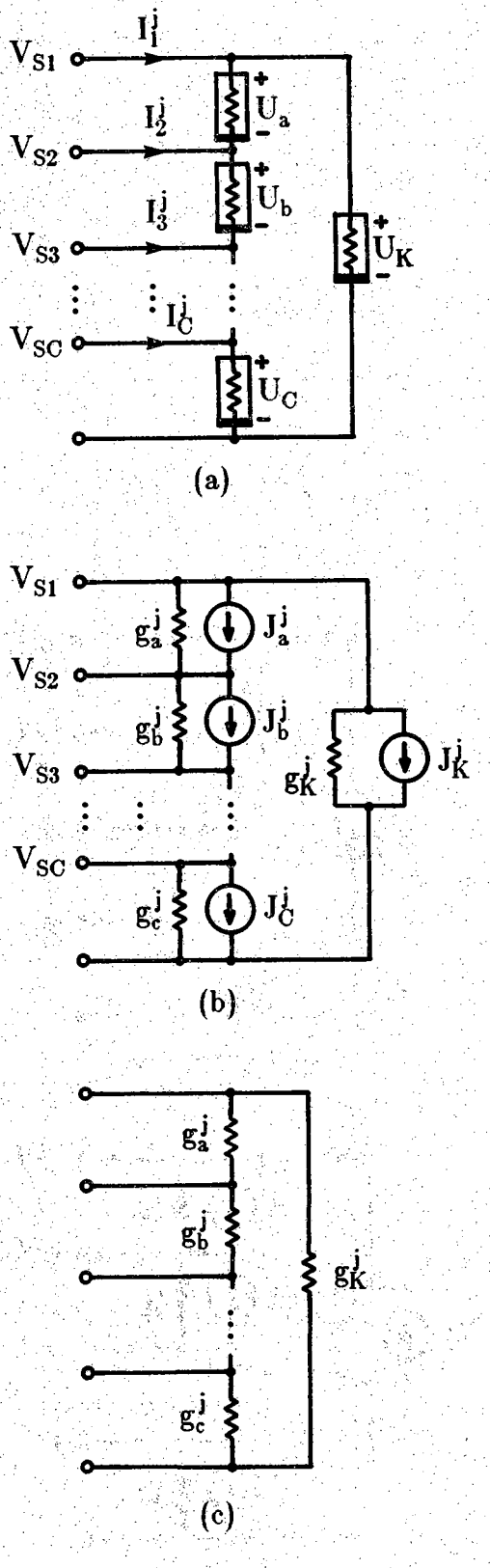
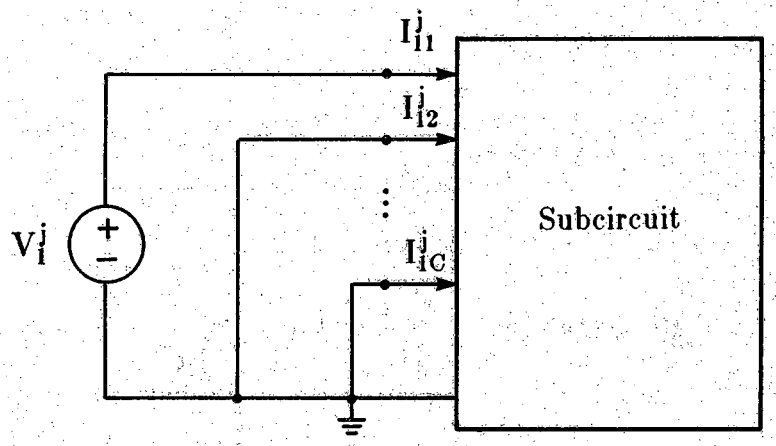
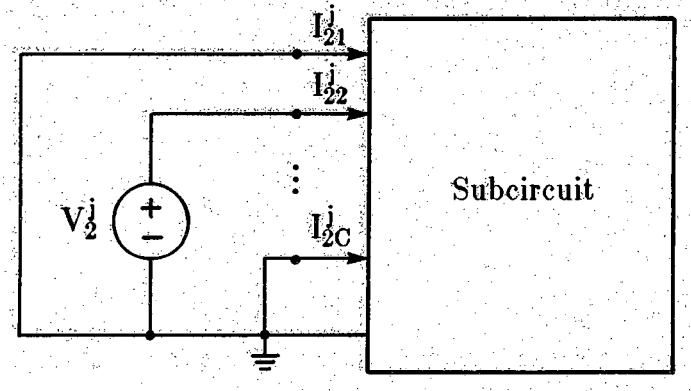


Figure 2.7 An example subcircuit



(a)



(b)

Figure 2.8 Circuits used for solving incremental currents

$$\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s1}} = \frac{I_{11}^j}{v_1}$$

$$\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s2}} = \frac{I_{21}^j}{v_2}$$

$$\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{sc}} = \frac{I_{c1}^j}{v_c}$$

$$\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s1}} = \frac{I_{12}^j}{v_1}$$

$$\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s2}} = \frac{I_{22}^j}{v_2}$$

$$\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{sc}} = \frac{I_{c2}^j}{v_c}$$

With the results of $\mathbf{I}^j$ and $\mathbf{G}_s^j$, the iterative current sources of equivalent circuit $\mathbf{J}_s^j$ cab be calculated by Eq.2.28. Applying the value of $\mathbf{G}_s^j$ and $\mathbf{J}_s^j$ in the main circuit and using the ADEC method, we then obtain the complete solution of the system in the j-th iteration.

2.3 Comparisons Of Three Newton Algorithms

In the MNA, the given circuit is partitioned in to one main circuit with many subcircuits, and the equivalent current sources are calculated in each iteration. The dimension of the matrices used will be reduced after this partition, and these subcircuits can be treated as a vector, and be solved in parallel. So the pipelined supercomputer can be used to process these vectorized equations efficiently. The more subcircuits in the system, the higher speedup can be achieved in MNA.

Using a tearing technique for the SLNA, some extra calculations are needed to solve the subcircuits. If do not consider the sparse technique, the time complexity of each iteration in SLNA is much higher than that in MNA. In MLNA, for each iteration of main loop, the entire inner loop operations must be repeated to solve the subcircuits. So a large number of iterations will be demanded. When we solve a l-level system, the number of iterations may increase as an exponential function of l. This may destroy the advantages of vector processing.

MNA has only one main loop. The number of iterations in the main loop is a constant which does not increase with the number of levels in the system. In each iteration of the MNA, only one **LU** decomposition and back-substitution are needed for solving the equivalent sources for each subcircuit. This leads to potential speedup advantage over a vector processor.

CHAPTER 3
COMPUTATIONAL REQUIREMENTS

Two examples are used to illustrate computational steps in the MNA. We start with a multiple-port subcircuit which shows how to compute the equivalent sources of the subcircuits. Another example is used to show all calculation steps in the MNA. Finally, we discuss the complexity and convergence issues of the proposed MNA for computer aided circuit analysis.

3.1 Circuit Formulation Using The MNA

We have to find the equivalent circuits for all the subcircuits in each iteration of the main loop. An example is used below to formulate the equivalent circuit. Consider a four ports subcircuit in Fig.3.1 which is characterized by the following equations:

$$i_1 = 0.5U_1^2 + U_1$$

$$i_2 = U_2^4 + U_2$$

$$i_3 = U_3^3$$

$$i_4 = 0.5U_4^2$$

$$i_5 = U_5^2 - U_5$$

In the j-th iteration, we assume initial values are: $v_1^j = 1$, $v_2^j = 3$, and $v_3^j = 2$. The associated discrete equivalent circuit of subcircuit in the j-th iteration is as shown in Fig.3.2 [7]. Using the ADEC method, we obtain the following formulation:

$$g_1^j = \frac{di_1}{dU_1} = U_1 + 1 = 2 \quad , \qquad J_1^j = i_1 - g_1^j U_1 = -0.5$$

$$g_2^j = \frac{di_2}{dU_2} = 4U_2 + 1 = 5 \quad , \qquad J_2^j = i_2 - g_2^j U_2 = -3$$

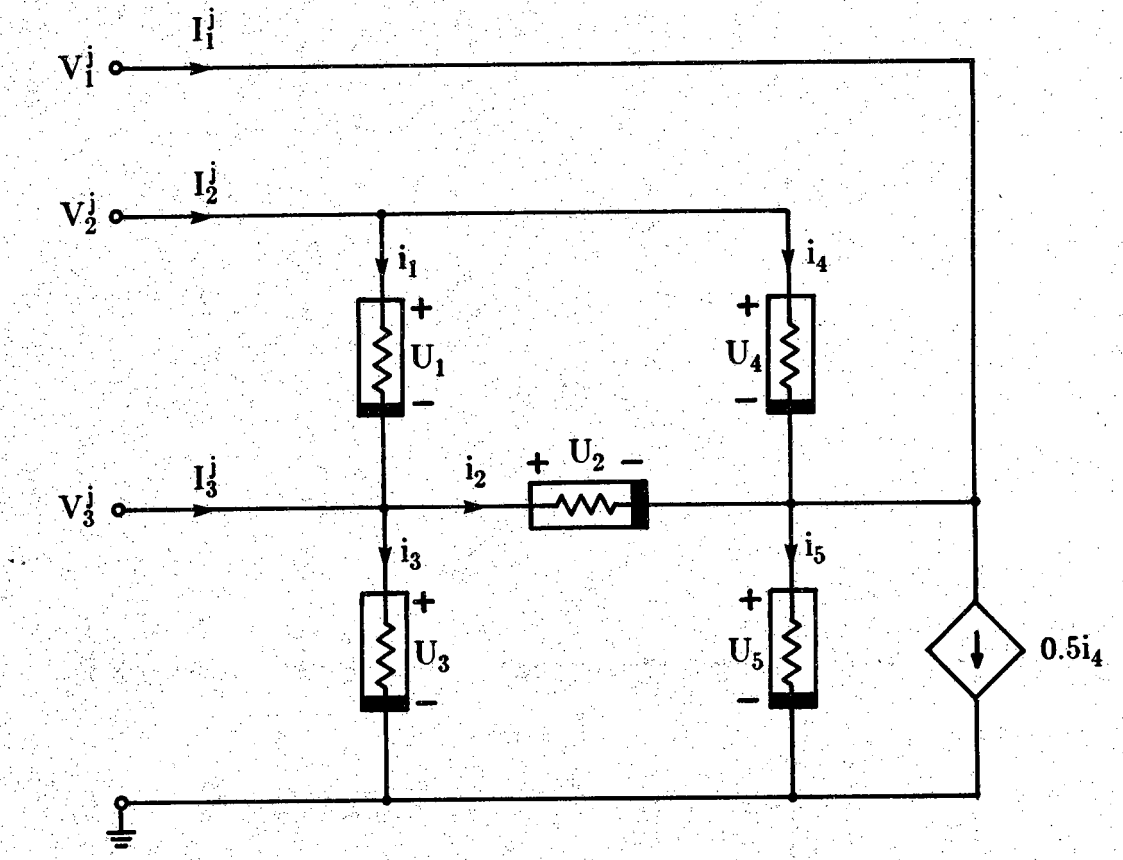


Figure 3.1 A four-port subcircuit

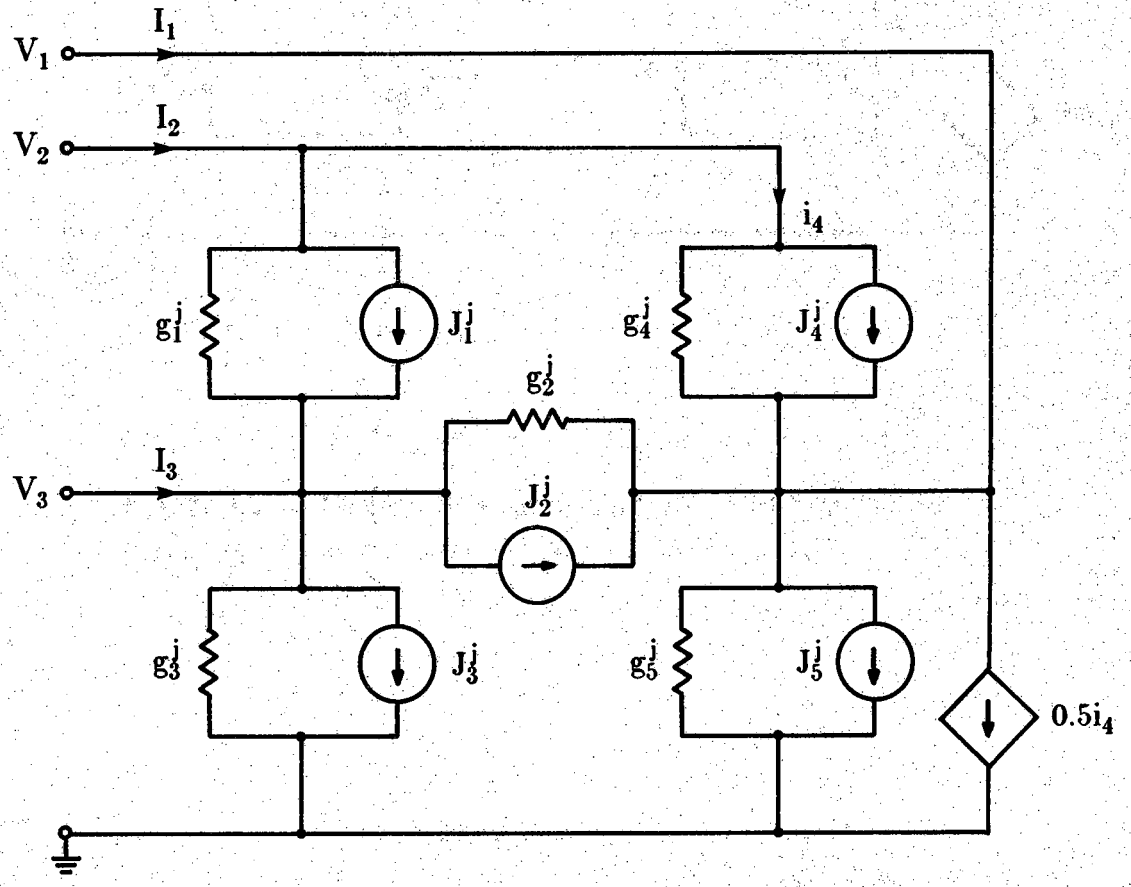


Figure 3.2 The discrete equivalent circuit to Figure 3.1

$$g_3^j = \frac{di_3}{dU_3} = 3U_3^2 = 12 \quad , \quad J_3^j = i_3 - g_3^j U_3 = -16$$

$$g_4 = U_4 = 2 \quad , \quad J_4^j = i_4 - g_4^j U_4 = -2$$

$$g_5^j = 2U_5 - 1 = 1 \quad , \quad J_5^j = i_5 - g_5^j U_5 = -1$$

Using the MNA method to solve this linearized circuit, we have the following system of equations:

$$\begin{bmatrix} 0 & 2 & -2 & 0 & -1 & 0 & 1 \\ -5 & -2 & 19 & 0 & 0 & -1 & 0 \\ 6 & 0 & -5 & -1 & 0 & 0 & -0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ I_1 \\ I_2 \\ I_3 \\ i_4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 18.5 \\ -2 \\ 1 \\ 3 \\ 2 \\ 2 \end{bmatrix} \tag{3.1}$$

The results are expressed as a column vector:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ I_1 \\ I_2 \\ I_3 \\ i_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ -3 \\ 3.5 \\ 8.5 \\ 2 \end{bmatrix}$$

The current vector in j-th iteration is thus obtained as:

$$\mathbf{I}^j = \begin{bmatrix} I_1^j \\ I_2^j \\ I_3^j \end{bmatrix} = \begin{bmatrix} -3 \\ 3.5 \\ 8.5 \end{bmatrix}$$

We set all independent sources in the equivalent circuit to be zero, the circuit in Fig.3.2 becomes that in Fig.3.3. Then we calculate the incremental conductance $\frac{\partial G_i(\mathbf{v}_s)}{\partial v_{sk}}$ in the following steps:

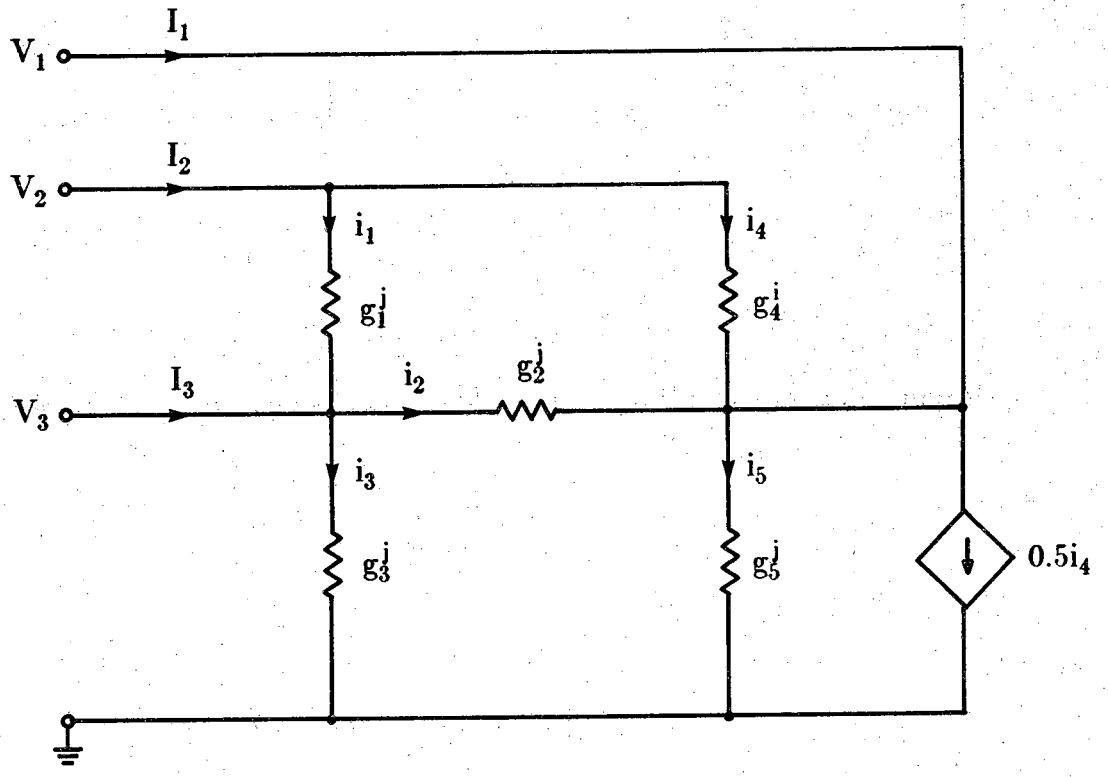


Figure 3.3 An equivalent circuit for solving the incremental currents

Step 1: Let the $v_1 = 1$, $v_2 = 0$, $v_3 = 0$, then using the MNA method to solve the circuit in Fig.3.3, we have to solve the following system:

$$\begin{bmatrix} 0 & 2 & -2 & 0 & -1 & 0 & 1 \\ -5 & -2 & 19 & 0 & 0 & -1 & 0 \\ 6 & 0 & -5 & -1 & 0 & 0 & -0.5 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -2 & 2 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ I_1 \\ I_2 \\ I_3 \\ i_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.2}$$

with the results

$$\mathbf{I}_1 = \begin{bmatrix} 7 \\ -2 \\ -5 \end{bmatrix}$$

and

$$\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s1}} = \frac{7}{1} = 7$$

$$\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s1}} = \frac{-2}{1} = -2$$

$$\frac{\partial G_3(\mathbf{v}_s)}{\partial v_{s1}} = \frac{-5}{1} = -5$$

Step 2: Let $v_1 = 0$, $v_2 = 3$, and $v_3 = 0$. Solving the circuit again, we obtain:

$$\mathbf{I}_2 = \begin{bmatrix} -3 \\ 12 \\ -6 \end{bmatrix}$$

and

$$\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s2}} = \frac{-3}{3} = -1$$

$$\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s2}} = \frac{12}{3} = 4$$

$$\frac{\partial G_3(\mathbf{v}_s)}{\partial v_{s2}} = \frac{-6}{3} = -2$$

Step 3: Let $v_1 = 0$, $v_2 = 0$, and $v_3 = 2$. we obtain:

$$\mathbf{I}_3 = \begin{bmatrix} -10 \\ -4 \\ 38 \end{bmatrix}$$

and

$$\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s3}} = \frac{-10}{2} = -5$$

$$\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s3}} = \frac{-4}{2} = -2$$

$$\frac{\partial G_3(\mathbf{v}_s)}{\partial v_{s3}} = \frac{38}{2} = 19$$

Then the $\mathbf{G}_s^j$ and $\mathbf{J}_s^j$ can be obtained as following:

$$G_{s1}^j = \left[\frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s1}} , \frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s2}} , \frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s3}} \right]$$

$$= \left[\ 7 \ , \ -1 \ , \ -5 \ \right]$$

$$G_{s2}^j = \left[\frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s1}} , \frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s2}} , \frac{\partial G_2(\mathbf{v}_s)}{\partial v_{s3}} \right]$$

$$= \left[\ -2 \ , \ 4 \ , \ -2 \ \right]$$

$$G_{s3}^j = \left[\frac{\partial G_3(\mathbf{v}_s)}{\partial v_{s1}} , \frac{\partial G_3(\mathbf{v}_s)}{\partial v_{s2}} , \frac{\partial G_3(\mathbf{v}_s)}{\partial v_{s3}} \right]$$

$$= \left[\ -5 \ , \ -2 \ , \ 19 \ \right]$$

and

$$J_{s1}^j = I_1^j - G_{s1}^j v_s^j$$
$$= -3 - \left[7\times1 + (-1)\times3 + (-5)\times2 \right] = 3$$

$$J_{s2}^j = I_2^j - G_{s2}^j v_s^j$$
$$= 3.5 - \left[(-2)\times1 + 4\times3 + (-2)\times2 \right] = -2.5$$

$$J_{s3}^j = I_3^j - G_{s3}^j v_s^j$$
$$= 8.5 - \left[(-5)\times1 + (-2)\times3 + 19\times2 \right] = -18.5$$

The final equivalent circuit in the j-th iteration is shown in Fig.3.4. Then we use the ADEC method to solve the main circuit.

3.2 Computation Steps In The MNA

Another example circuit (Fig.3.5) is used to illustrate the computations involved in the MNA. The associated discrete equivalent circuit for the subcircuit is shown in Fig.3.6a. The equivalent circuit for solving the main circuit in the j-th iteration is shown in Fig.3.6b.

Assume the initial guess is $v_1^0 = 7$, $v_2^0 = 1$, and $v_3^0 = 0$. The following steps are needed in each iteration of the main loop:

Step 1: Calculate the G_s^0 and J_s^0.

Using the ADEC method, we have

$$g_1^0 = \left[\frac{di_1}{dU_1} \right]_{U_1 = v_2^0} = 2$$

$$g_2^0 = \left[\frac{di_2}{dU_2} \right]_{U_2 = v_3^0} = 0$$

$$J_1^0 = i_1^0 - g_1^0 v_2^0 = -1$$

$$J_2^0 = i_2^0 - g_2^0 v_3^0 = 0$$

By the MNA method, we obtain the equation

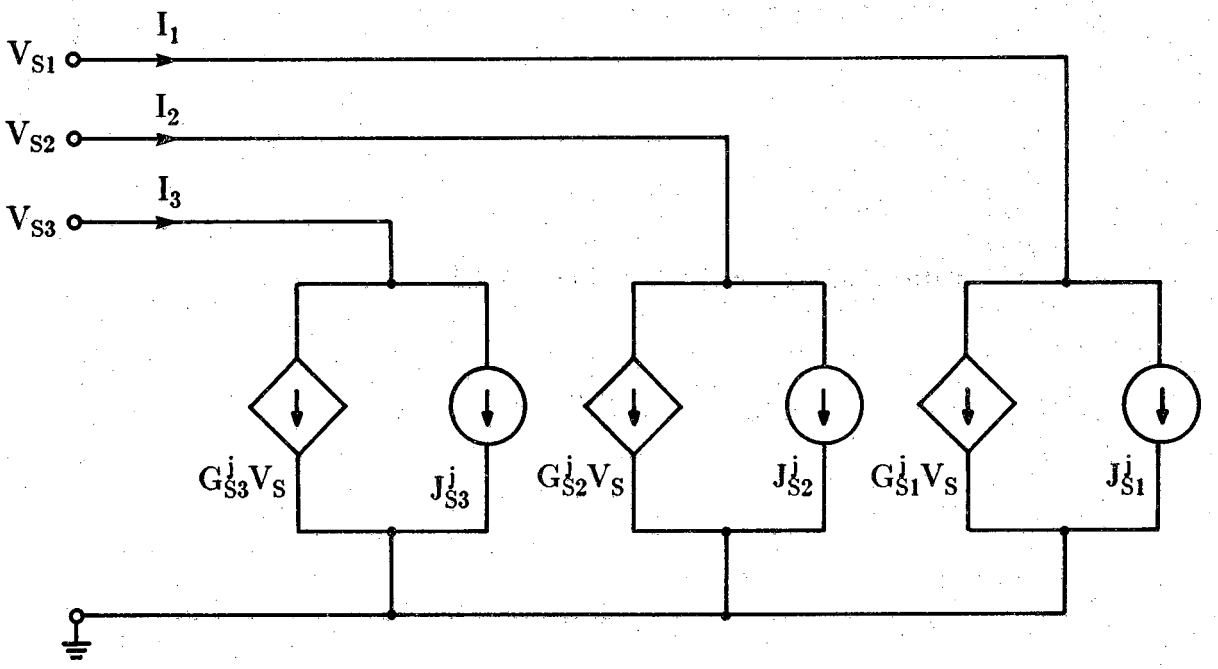


Figure 3.4 Equivalent current sources in a subcircuit

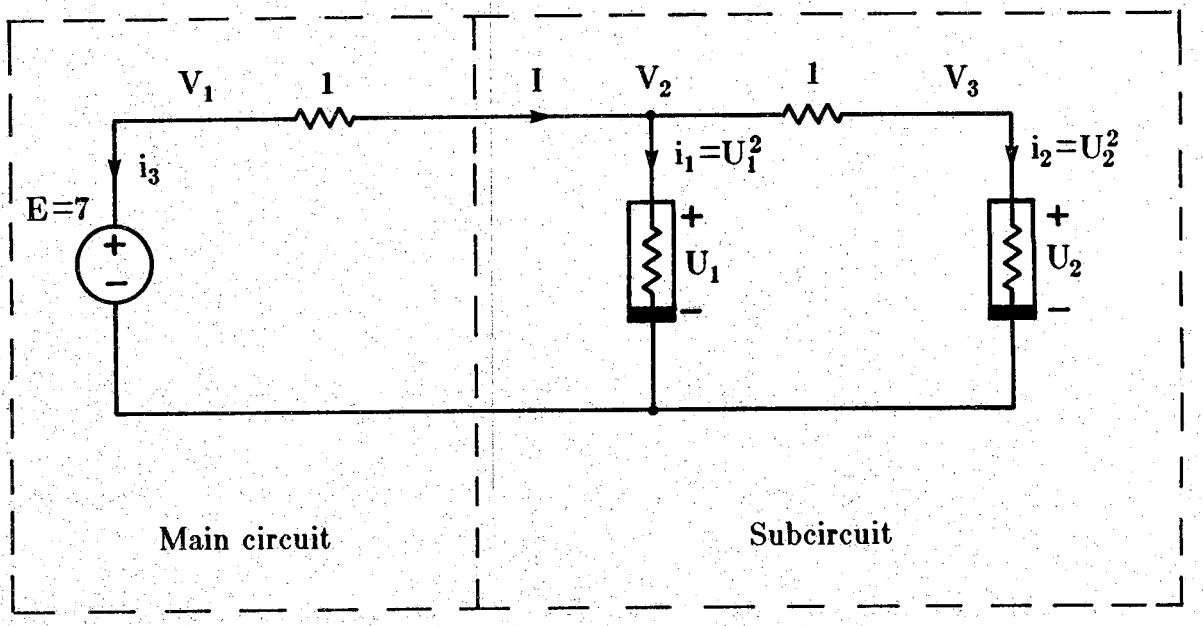


Figure 3.5 An example circuit

$$\begin{bmatrix} 1+g_1^0 & -1 & -1 \\ -1 & 1+g_2^0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} -J_1^0 \\ -J_2^0 \\ v_2^0 \end{bmatrix}$$

or

$$\begin{bmatrix} 3 & -1 & -1 \\ -1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \tag{3.3}$$

The solution vector is obtained:

$$\begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

where I is the current at port I^0. Then we set all independent sources in Fig.3.6a to be zero and keep the input voltage v_2^0, we have

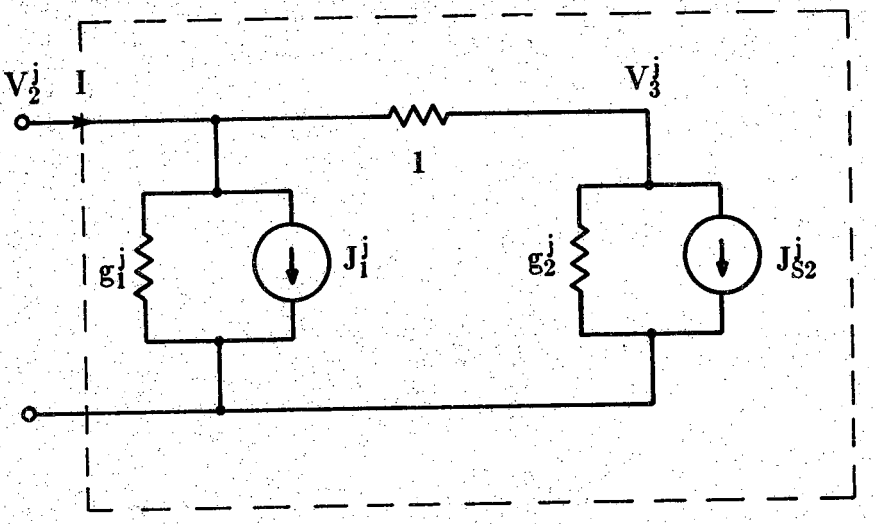
$$\begin{bmatrix} 1+g_1^0 & -1 & -1 \\ -1 & 1+g_2^0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ v_2^0 \end{bmatrix} \tag{3.4}$$

The solutions are:

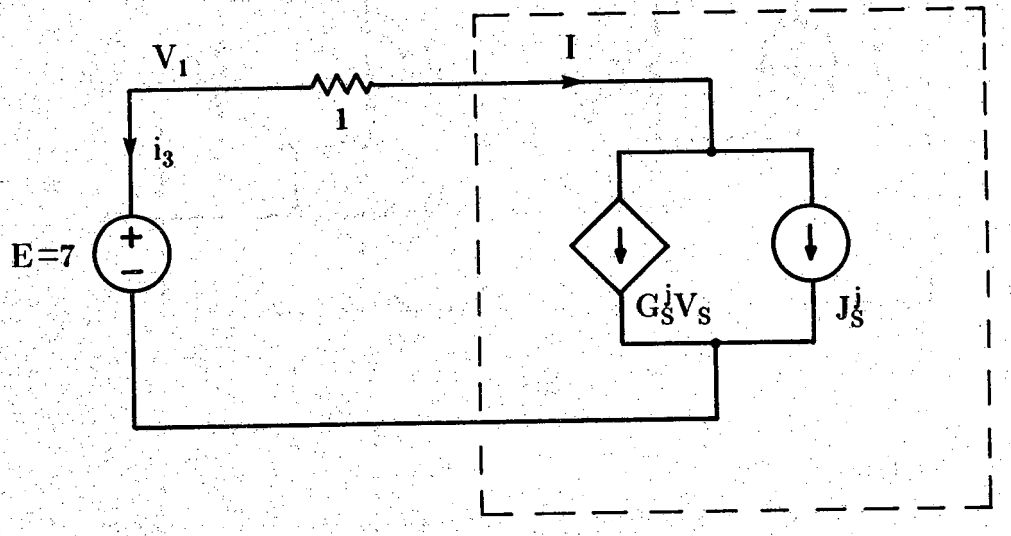
$$\begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

Here I is the incremental current. Since the subcircuit has only one input voltage v_2, the equivalent circuit is formulated as follows:

$$G_s^0 = \frac{\partial G_1(\mathbf{v}_s)}{\partial v_{s1}} = \frac{2}{1} = 2$$



(a)



(b)

Figure 3.6 The equivalent circuit to Figure 3.5

$$G_s^0 \cdot \mathbf{v}_s = 2\,v_s$$

$$J_s^0 = I^0 - G_s^0 \cdot \mathbf{v}_s = 1 - 2\times 1 = -1 \tag{3.5}$$

Step 2: Substituting G_s^0 and J_s^0 into the main circuit as shown in Fig.3.6b, and using the MNA method , we obtain:

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1+G_s^0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ -J_s^0 \\ 7 \end{bmatrix}$$

or

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 3 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ i_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 7 \end{bmatrix} \tag{3.6}$$

This step yields the solutions:

$$v_1^1 = 7.0$$

$$v_2^1 = 2.666667$$

$$i_3^1 = -4.333333$$

Step 3: Substituting the results from the main circuit into the subcircuit, we obtain:

$$\begin{bmatrix} 1+g_1^0 & -1 & -1 \\ -1 & 1+g_2^0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} -J_1^0 \\ -J_2^0 \\ v_2^1 \end{bmatrix} \tag{3.7}$$

The corresponding results are:

$$v_2^1 = 2.666667$$

$$v_3^1 = 2.666667$$

$$I = 4.333333$$

Now, the first iteration of entire circuit is completed. The results v_1^1 , v_2^1, and v_3^1 will be used as the initial values to start the second iteration. The steps in second iteration will be similar to those in the first one. Detailed steps are skipped. Only the results after the second iteration are shown below:

Repeat step 1: Calculate the G_s^1 and J_s^1. Using the ADEC method, we have:

$$g_1^1 = \left[\frac{di_1}{dU_1} \right]_{U_1 = v_1^1} = 5.333333$$

$$g_2^1 = \left[\frac{di_2}{dU_2} \right]_{U_2 = v_2^1} = 5.333333$$

$$J_1^1 = i_1^1 - g_1^1 v_1^1 = -7.111111$$

$$J_2^1 = i_2^1 - g_2^1 v_2^1 = -7.111111$$

Using the MNA method to solve the circuit in Fig3.6a , we have:

$$\begin{bmatrix} 1+g_1^1 & -1 & -1 \\ -1 & 1+g_2^1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} -J_1^1 \\ -J_2^1 \\ v_2^1 \end{bmatrix}$$

and

$$v_2 = 2.666667$$

$$v_3 = 1.543860$$

$$I = 8.233918$$

where I is the port current I^1 of the subcircuit. Then we set the independent sources in Fig.3.6a to be zero. solving the circuit we have:

$$\begin{bmatrix} 1+g_1^1 & -1 & -1 \\ -1 & 1+g_2^1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2 \\ v_3 \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ v_2^1 \end{bmatrix}$$

and

$$v_2 = 2.666667$$

$$v_3 = 0.421053$$

$$I = 16.467840$$

Thus

$$G_{s1}^1 = \frac{\partial G_1(v_s)}{\partial v_{s1}} = \frac{16.467840}{2.666667} = 6.175439$$

$$G_{s1}^1 \cdot v_s = 6.175439\, v_2$$

$$J_s^1 = I^1 - G_{s1}^1 v_s^1 = -8.233918$$

Repeat step 2: Substituting G_s^1 and J_s^1 into the main circuit and using the MNA method, we obtain:

$$v_1^2 = 7.0$$

$$v_2^2 = 2.1230645$$

$$i_3^2 = -4.876936$$

Repeat step 3: Substituting the results from the main circuit to the subcircuit, we obtain:

$$v_3^2 = 1.458028$$

The second iteration of MNA is then completed. The values v_1^2, v_2^2, and v_3^2 will be used to start the third iteration similarly. The results of the main circuit and the subcircuit in successive iterations are listed in TableA.1 of Appendix A. In this example, it takes five iterations to obtain the exact solutions:

$$v_1 = 7.0$$

$$v_2 = 2.0$$

$$v_3 = 1.0$$

$$i_3 = -5.0$$

The procedures described above correspond to one main loop in MNA. Instead multiple loops are required in the MNLA. This is the important difference between the two algorithms [2].

3.3 Complexity And Convergence Issues

From the above two examples for each c-port subcircuit, the computations involved requires to solve c linear systems of equations characterized by $\mathbf{A}\,\mathbf{x}_1 = \mathbf{b}_1$, $\mathbf{A}\,\mathbf{x}_2 = \mathbf{b}_2$, $\cdots$ $\mathbf{A}\,\mathbf{x}_c = \mathbf{b}_c$. Since they are described by the same coefficient matrix **A**, the **L U** decomposition method is used:

$$\mathbf{A} = \mathbf{L}\,\mathbf{U}$$

$$\mathbf{L}\,\mathbf{y}_i = \mathbf{b}_i$$

$$\mathbf{U}\,\mathbf{x}_i = \mathbf{y}_i \qquad (3.8)$$

The matrix **A** is decomposed into a lower-triangular matrix **L** and a upper-triangular matrix **U**. Then the back-substitution is used to obtain the vector $\mathbf{y}_i$ and the solution vector $\mathbf{x}_i$. In each iteration of the MNA, we need to perform one **L U** decomposition and c back-substitutions for a c-port subcircuit. we know that the time complexity for the submatrices **L U** decomposition is $O(ns^3)$, but for back-substitution it is only $O(ns^2)$. When the dimension of the submatrices **A** is large, the time complexity for solving c equations with the same coefficient matrix **A** is $O(ns^3) + cO(ns^2) = O(ns^3)$ that is the same time complexity for solving one equation.

Table 3.1 gives the time complexities of these algorithms. As a reference, the time complexity of the *Semi-Direct Method* (SDM) is listed here [19]. The time complexity of each iteration in this method is about the same as in MNA, but this method has the linear convergency rate. Assume the circuit is partitioned to two levels. T_1, T_2, T_3 and T_4 are time needed for solving the subcircuit in four algorithms, T is the time needed for solving the main circuit, and S is the speedup of vector processing over scalar processing.

In the MNLA the entire inner loop is required in each iteration of the main loop, and one **L U** decomposition is needed for each inner loop iteration. The comparison of computation steps in the MLNA and MNA are shown in Fig. 3.7. Figure 3.7a shows a main loop in the MNA and there are three major steps in each iteration. Figure 3.7b shows a two-level structure of the MLNA. In solving an l-level system, the program should have l levels of looping. If there are p iterations in each loop, then the MNLA would need p^l iterations for solving a subcircuit in the l-th level. But only p iterations are needed in the MNA. This is a significant improvement, when the system becomes large.

In general the SLNA has a quadratic convergence rate. Let $\mathbf{F}(\mathbf{U}) = 0$ be the set of equations in the SLNA. Then the increment $\| \Delta U \|$ in each iteration would be approximately the square of the increment in the last

Table 3.1. Time Complexity.

ALGORITHM / COMPUTER TYPE	SLNA*	MLNA	MNA	SDM**
SCALAR	$P(NT_1+T)$	$P(qNT_2+T)$	$P(NT_3+T)$	$P(NT_3+T)$
VECTOR	$P(NT_1+T)/s$	$P(qNT_2+T)/s$	$P(NT_3+T)/s$	$P(NT_4+T)/s$

* Assume the tearing technique is used

** This method has the different convergency rate with the other algorithms.

N: The number of subcircuits

n: The number of unknowns in the main circuit

ns: The number of unkowns in one subcircuit

s: Speedup of vector processor over scalar processor

$$T = \frac{1}{3}n^3 \qquad T_1 = \frac{1}{3}ns^3 + 2ns^3 = \frac{7}{3}ns^3$$

$$T_2 = \frac{1}{3}ns^3 \qquad T_3 = \frac{1}{3}ns^3 \quad T_4 = \frac{1}{3}ns^3$$

$$T_1 > T_2 = T_3 = T_4$$

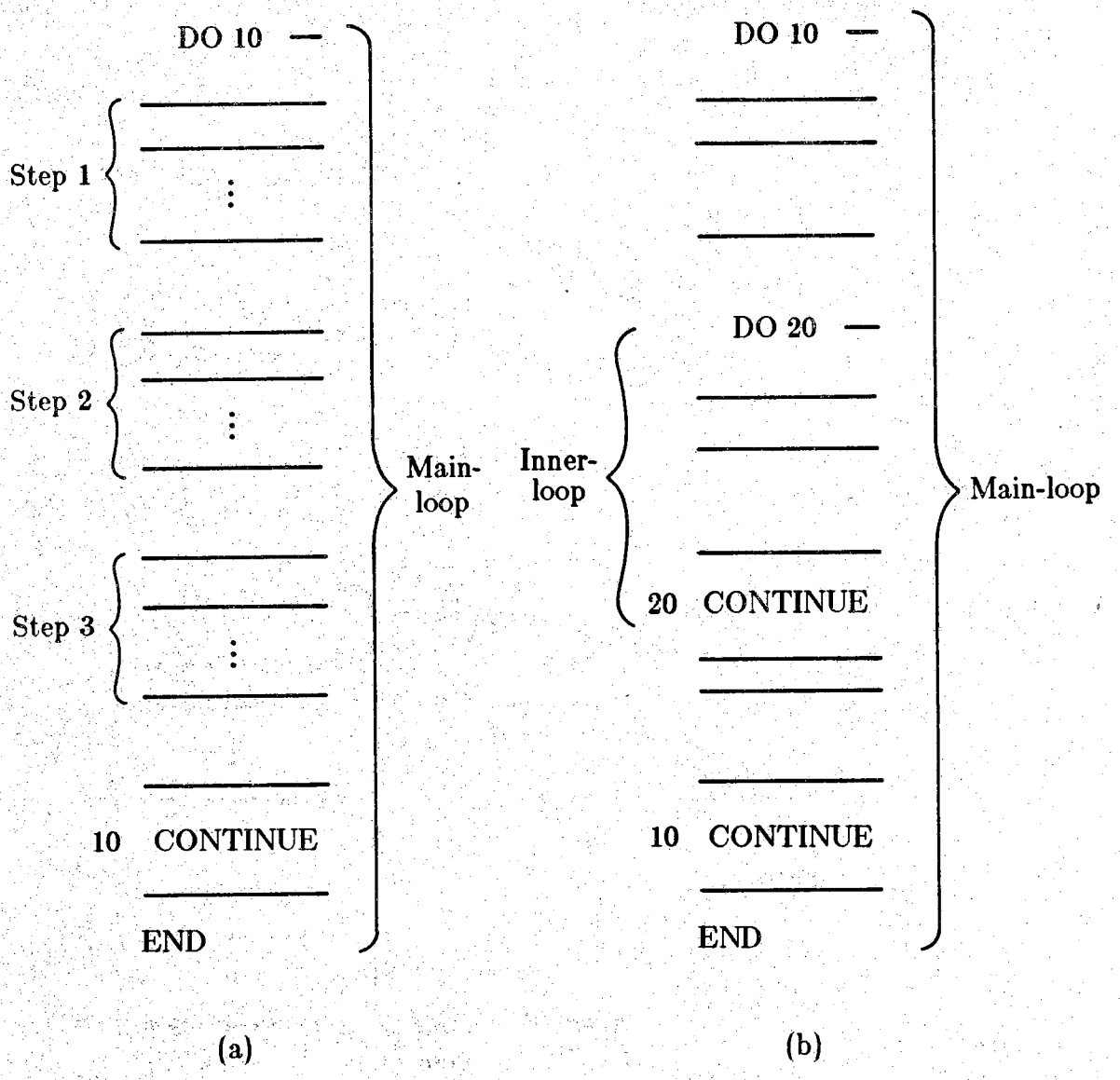


Figure 3.7 Program structures in the MNA and MLNA

iteration. Let us demonstrate this by the same circuit shown in Fig.3.5. By the SLNA the discrete equivalent circuit is shown in Fig.3.8. Assuming the initial guess is the same as before, $v_1^0 = 7$, $v_2^0 = 1$, $v_3^0 = 0$, we have:

$$g_1^0 = \left[\frac{di_1}{d_1}\right]_{U_1 = v_1^0} = 2$$

$$g_2^0 = \left[\frac{di_2}{dU_2}\right]_{U_2 = v_2^0} = 0$$

$$J_1^0 = i_1^0 - g_1^0 v_1^0 = -1$$

$$J_2^0 = i_2^0 - g_2^0 v_2^0 = 0$$

Using the MNA method we have the following set of equations:

$$\begin{bmatrix} -1 & -1 & 0 & 1 \\ -1 & 2+g_1^0 & -1 & 0 \\ 0 & -1 & 2+g_2^0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ i \end{bmatrix} = \begin{bmatrix} 0 \\ -J_1^0 \\ -J_2^0 \\ v_1^0 \end{bmatrix} \qquad (3.9)$$

Solving it, we obtain the results after the first iteration of the SLNA.

$$v_1^1 = 7.0$$

$$v_2^1 = 2.666667$$

$$v_3^1 = 2.666667$$

$$i^1 = -4.333333$$

Table A.2 in Appendix A gives the results after each iteration in the SLNA. The increments decrease after each iteration at a quadratic convergence rate. Equations 2.24 and 2.25 show that the MNA has the same convergency rate as in the SLNA. This also can be found by comparing the TableA.1 and TableA.2, They exactly have the same values corresponding to each iteration.

In a strict sense, the convergence of the MNA depends on the structure of the circuit. Suppose the circuit is characterized by a set of differential equations, $\mathbf{F(U)} = 0$. Let U^* is the solution vector, and the $\mathbf{J(U)}$ is the

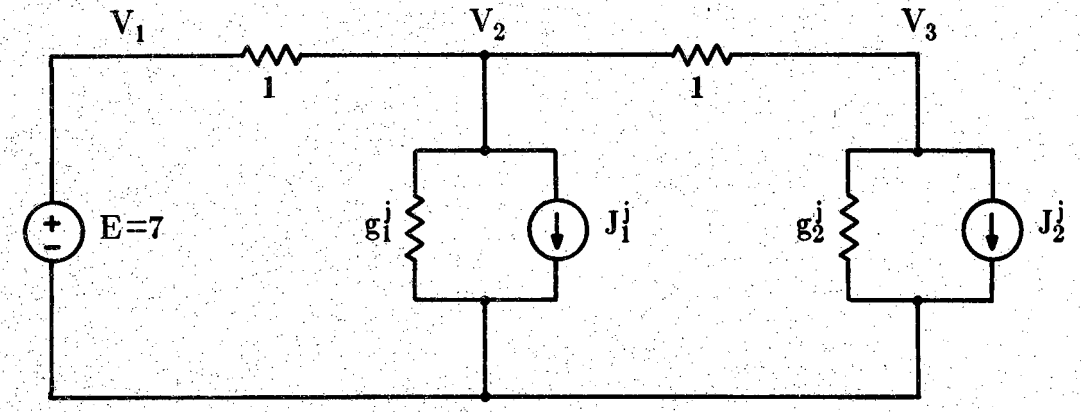


Figure 3.8 The discrete equivalent circuit for SLNA

Jacobian matrix of $\mathbf{F(U)}$. When

$$\det \mathbf{J}(\mathbf{U}^*) = 0 \tag{3.10}$$

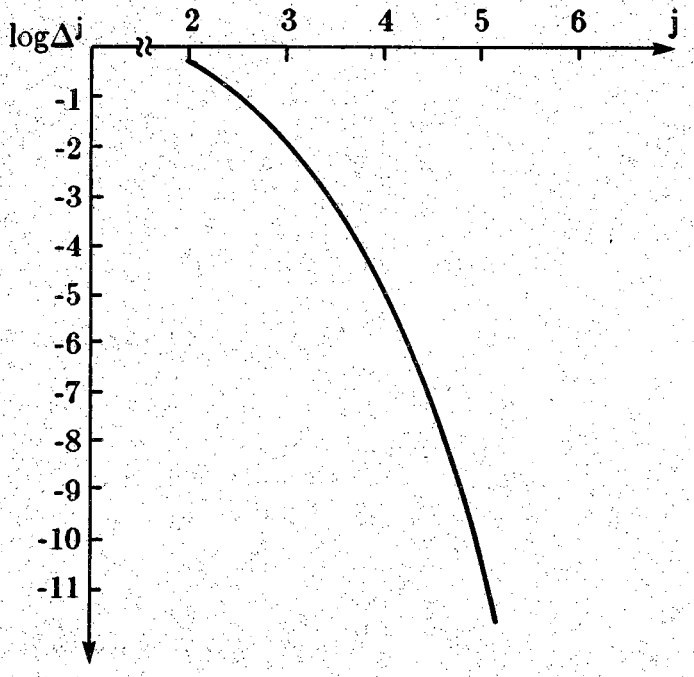
the MNA and SLNA both have linear convergence. However, in general, when Eq.3.10 is not true, the MNA and SLNA would have a quadratic convergence rate or faster [5] [6]. So we claim that the MNA has a quadratic convergence rate.

In the MLNA, the convergence of the main loop depends on the precision of the results in the inner loop. The higher is the precision in the inner loop, the higher will be the convergency in the main loop [1]. In general the inner loop termination criterion is chosen as

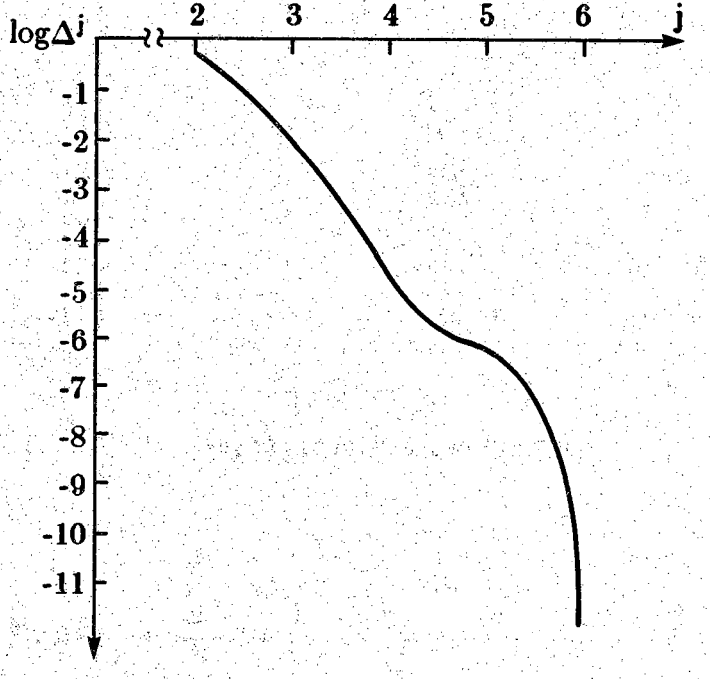
$$\left|\left| \Delta X , \Delta Y \right|\right| \leq \min \{ r^0 , \left|\left| \Delta U , \Delta\omega \right|\right|^{\alpha} \} \tag{3.11}$$

Where ΔX and ΔY are the increments of the inner loops, ΔU and $\Delta\omega$ are the increments of the main loop, $\mathbf{r}^0$ is the initial termination criterion of the inner loop. It has been proved that for $\alpha = 2$, the MLNA has the quadratic convergence or faster [1] [5]. However this means the precision of the inner loop should be very high. As an example, with an increment of the main loop in the j-th iteration $\left|\left|\Delta U , \Delta\omega\right|\right| = 0.0001$, the precision at the inner loop should be equal to or less than 0.00000001, which demands many inner iterations to satisfy this criterion. If one wants to improve the convergence of the main loop in the MLNA, the number of iterations in the inner loop would be increased, and the total time complexity of both main and inner loops may not be reduced.

If $\alpha \leq 1$, inner loop can then use the same termination criterion as used in the main loop. The convergence rate of MLNA is neither quadratic nor linear, but called "Pairwise quadratic convergence" as proved in [5]. This convergence is shown in Fig.3.9a. It displays some "kinks" which mean that the curve alternates between slow-decreasing and fast-decreasing intervals, even when the number of iterations j become very large. Figure 3.9b shows the curve of quadratic convergency of MNA. It converges faster than the MLNA when $\alpha \leq 1$. The comparison of the convergence rates of four algorithms is shown in Table 3.2.



(a) Quadratic convergence



(b) Pairwise quadratic convergence

Figure 3.9 Convergence rates

Table 3.2. Convergence Rates

SLNA	MLNA	MNA	SDM
QUADRATIC det $J(U^*) \neq 0$	QUADRATIC $\alpha = 2$ (Rabbat, et al.) PAIRWISE QUADRATIC $\alpha \leq 1$ (Lin, et al.)	QUADRATIC det $J(U^*) \neq 0$	LINEAR (Lin, et al.)

CHAPTER 4
VECTORIZED SIMULATION PROGRAMS

This chapter presents the major vectorized programs used in circuit simulation. Section 4.1 describes the subnetwork update program. The program for **LU** decomposition to solve the subcircuits is explained in section 4.2. Section 4.3 illustrates the program for the main network update. The programs for solving the main circuit are described in section 4.4.

4.1 Subnetwork Update Programs

As mentioned before, in MNA, the circuit is partitioned to a main circuit and some subcircuits. Hence, the dimensions of the matrices in the equation can be reduced. Moreover, if there are a lot of identical subcircuits in the circuit, they can be treated as the elements of a vector. For example, assume there are N subcircuits which have the same structure, and there are two parameters p_i and q_i in i-th subcircuit. Since all the subcircuits have the same structure, the vector $\mathbf{P} = [p_1 , p_2 , \cdots p_N]$ and $\mathbf{Q} = [q_1 , q_2 , \cdots q_N]$ can be used to represent the parameters in all the subcircuits. If we need to add p_i and q_i up , we can use the vector pipeline of the supercomputer Cyber-205 to do the vector addition $\mathbf{P}+\mathbf{Q}$ for all the subcircuits. It can obtain a higher speedup than using the scalar processor to add them individually.

From the example in the last chapter, we know that the associated discrete equivalent circuits of the nonlinear resistors as shown in Fig.3.2 are needed. And these values are put into the equations before solving the equivalent current sources of subcircuits. We call these procedure subcircuit update.

Assume the current of a nonlinear resistor is the function of the voltage across it,

$$i = f(v) \tag{4.1}$$

Then the incremental conductance is calculated by

$$g^j = \left[\frac{\partial f(v)}{\partial v} \right]_{v=v^j} \tag{4.2}$$

The iterative current is calculated by

$$J^j = i^j - g^j v^j \tag{4.3}$$

The f(v) can be any kind of function of the voltage v's, which may assume a very complicated form. So, for calculating easily on the computer, we use the Taylor expansions of functions f(v) and $\frac{\partial f(v)}{\partial v}$. The more items of Taylor expansion are taken, the more precision will be obtained. We use the first 10 items of the expansion in our simulation program. The coefficients of the items of the functions will be stored in a two-dimensional array called PO. Each column of PO corresponds to one type of nonlinear resistor. The coefficients of $\frac{\partial f(v)}{\partial v}$ are stored in the upper half of the columns in PO, the coefficients for f(v) are stored in the lower half positions. As an example, there is a PO whose structure is as follows:

$$PO = \begin{bmatrix} a_1 & k_1 \\ a_2 & k_2 \\ a_3 & k_3 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ a_{10} & k_{10} \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ a_{20} & k_{20} \end{bmatrix}$$

It means that the the first 10 items of the Taylor expansion of $\frac{\partial f(v)}{\partial v}$ for the first type of nonlinear resistors are

$$\frac{\partial f(v)}{\partial v} = a_1 v^9 + a_2 v^8 + , \cdots a_{10}$$

and the Taylor expansion of f(v) for the first type of nonlinear resistors is

$$f(v) = a_{11} v^9 + a_{12} v^8 + , \cdots a_{20}$$

Generally, there are more than one types of nonlinear resistors in the circuit, and the k-th column of PO corresponds to the k-th type of nonlinear resistor. A two-dimensional array called DS is used to store the pointer for each nonlinear resisitor in the subcircuits. Each row of DS corresponds to one

nonlinear resistor. The first and second entries in a row are the numbers of nodes to which the corresponding nonlinear resistor is connected. The voltage across the resistor can be obtained from these two nodes. The third entry of the row indicates which column of PO corresponds to this nonlinear resistor. If a number i is in the third position of one row of DS, it means the corresponding coefficients of the nonlinear resistor is stored in i-th column of PO. The last entry of DS is either 1 or 0. 0 means that the resistor is connected to the datum of the subcircuit, and 1 means that the resistor is not connected to the datum.

Let us look at an example shown in Fig.4.1a. The first row of DS corresponds to the nonlinear resistor shown in Fig.4.1b, which is connected from node 3 to node 5, and not connected to the datum. The corresponding coefficients of its function are stored in the second column of PO. The second row of DS means that the nonlinear resistor in the subcircuit shown in Fig.4.1c is connected from node 4 to node 7 that is the datum of the subcircuit. The corresponding coefficients of its function are stored in the third column of PO.

All the nonzero elements of the matrix **A** for solving the subcircuit are stored in array NZS, and the elements of the right vector b are stored in array brs. The values of g^j and J^j need to be inserted into the arrays NZS and brS. Therefore, two bit mask arrays are used to locate g^j and J^jin the NZS and brS.

In the simulating program, we assume all the subcircuits have the same structure, and each subcircuit has only two ports. Furthermore, we assume the symbolic processing and row exchanges, column exchanges for all matrices have been done before the simulation. Two integer arrays NZPSc and brSc are used to indicate the order exchanges of g^j and J^j in NZS and brS. Another integer array CIS indicates the column exchanges of matrix **A** for subcircuits. The program for calculating the g^j and J^j in subcircuits is as follows:

1) Obtain the voltages accross the nonlinear resistor

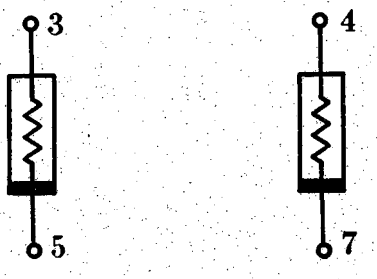
```
P=0
DO 10 i=1, nls
IF (DS(i,4).EQ.0) THEN
V(1;y)=XS(1,DS(i,1);y)
ELSE
V(1;y)=XS(1,DS(i,1);y)-XS(1,DS(i,2);y)
ENDIF
```

2) Calculate the polynomial

```
g(1;y)=0.0
```

$$DS = \begin{bmatrix} 3 & 5 & 2 & 1 \\ 4 & 7 & 3 & 0 \\ . & . & . & . \\ . & . & . & . \end{bmatrix}$$

(a) Array DS



(b) Resistor used

Figure 4.1 The array DS and resistors used

```
      DO 20 j=1, 10
      g(1;y)=g(1;y)*V(1;y)+PO(j,DS(i,3))
20    CONTINUE
      I(1;y)=0.0
      DO 30 j=11,20
      I(1;y)=I(1;y)*V(1;y)+PO(j,DS(i,3))
30     CONTINUE
```

3) Obtain the J and G

```
      J(1,i;y)=g(1;y)*V(1;y)-I(1;y)
      p=p+1
      G(1,p;y)=g(1;y)
      IF (DS(i,4).EQ. 0) GOTO 10
      p=p+1
      G(1,p;y)=-g(1;y)
10    CONTINUE
```

Where nls is the number of nonlinear resistors in each subcircuit, the y in program is the number of identical subcircuits in the circuit. All the subcircuits will be treated as a vector and be solved by the vector pipelines of Cyber-205.

The vector XS is used for storing all variables of the subcircuits, G is a two-dimensional array for storing all g in NZS. Changing order of g and J and inserting them into NZS and brs respectively can be achieved by the following operations:

```
      p=0
      DO 40 i=1, ms
      IF (BTOL(NZPS(i))) THEN
      p=p+1
      NZS(1,i;y)=G(1,NZPSc(p);y)
      ENDIF
40    CONTINUE

      p=0
      DO 50 i=1, ns
      IF (BTOL(bps(i))) THEN
      p=p+1
      brs(1,i;y)=J(1,bpSc(p);y)
      ENDIF
```

```
50    CONTINUE
```

Where ms is the number of nonzero elements in NZS, ns is the dimension of the matrix **A** for the subcircuits. If only the scalar processor of Cyber-205 is used, the corresponding program to do the same operations as the above one will be as follows:

```
      DO 10 z=1, y
      p=0
      DO 20 i=1, nls
      IF (DS(i,4).EQ.0) THEN
      V=XS(z,DS(i,1))
      ELSE
      V=XS(z,DS(i,1))-XS(z,DS(i,2))
      ENDIF

      g=0
      DO 30 j=1, 10
      g=g*V+PO(j,DS(i,3))
30    CONTINUE

      I=0
      DO 40 j=11, 20
      I=I*V+PO(j,DS(i,3))
40    CONTINUE

      J(i)=g*V-I
      p=p+1
      G(p)=g
      IF (DS(i,4).EQ.0) GOTO 20
      p=p+1
      G(p)=-g
20    CONTINUE

      p=0
      DO 50 i=1, ms
      IF (BTOL(NZPS(i))) THEN
      p=p+1
      NZS(z,i)=G(p)
      ENDIF
```

```
50    CONTINUE

      p=0
      DO 60 i=1, ns
      IF (BTOL(bps(i))) THEN
      p=p+1
      brs(z,i)=J(p)
      ENDIF
60    CONTINUE
10    CONTINUE
```

From the program above we can see, the program of scalar version needs an additional loop to replace a set of vector instructions in the vector version. So it is not efficient.

Finally, to update the subcircuits, the port voltages of the subcircuits which come from the main circuit, should be inserted into array brS. Because each subcircuit has only two ports in our simulation, an integer vector bpV is used to locate these port voltages in the equation of main network, and a variable bpp is used to indicate their positions in the equations of each subnetwork.

4.2 Subcircuit Decomposition Programs

From the example in chapter three, we can see, that the equivalent independent current sources and controlled current sources of the subcircuits should be calculated after updating the subnetworks. Then these values are put into the equations of main circuit in order to solve it in each iteration. In MNA, it needs to do only one **LU** decomposition for each subcircuit instead of a loop where one **LU** decomposition is done for each iteration in MLNA algorithm for each main iteration. Since it may need to do more than one time of substitutions for one **LU** decomposition, the operations to do these two things are written as a separate subroutine.

LU decomposition method is used to solve the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$. The elements in **L** and **U** are determined by the following formulas:

$$l_{i,j} = \begin{cases} 0 & \text{if } i<j \\ a_{i,j} - \sum_{p=1}^{j-1} l_{i,p}\, u_{p,j} & \text{if } i \geq j \end{cases} \qquad (4.4)$$

$$u_{i,j} = \begin{cases} 0 & \text{if } i>j \\ 1 & \text{if } i=j \\ \frac{1}{l_{i,i}}\left(a_{i,j} - \sum_{p=1}^{i-1} l_{i,p}\, u_{p,j}\right) & \text{if } i<j \end{cases} \qquad (4.5)$$

The program to calculate the k-th column of **L** is as follows:

1) Initialize

```
      DO 10 i=k, ns
      IF (BTOL(BS(i,k))) THEN
      sum(1;y)=0.0
      ql=RS(i)-1
```

2) Calculate the sum of the $l_{ip} \cdot u_{pj}$

```
      DO 20 p=1, k-1
      IF (BTOL(BS(i,p))) THEN
      ql=ql+1
      IF (BTOL(BS(p,k))) THEN
      qu=RS(p)+Q8SCNT(BS(p,1;k))-1

      sum(1;y)=NZS(1,ql;y)*NZS(1,qu;y)+sum(1;y)
      ENDIF
      ENDIF
   20 CONTINUE
```

3) Obtain the k-th column of L

```
      ql=ql+1
      NZS(1,ql;y)=NZS(1,ql;y)-sum(1;y)
      ENDIF
   10 CONTINUE
```

Where BS is a two-dimensional bit mask array for the subcircuits, RS is an integer vector to locate the first nonzoero element of each row of **A** for the subcircuits.

Since the first column of does not need any computation which can be obtained directly from the matrix **A**, we can calculate the columns of **L** only from 2 to ns. Similarly, the last row of **U** does not need calculating either. The program for calculating the k-th row of **U** is as follows:

1) Initialize

```
IF (k.NE.ns) THEN
DO 30 j=k+1, ns
IF (BTOL(BS(k,j))) THEN
sum(1;y)=0.0
ql=RS(k)-1
qq=ql
```

2) Calculate the sum of $l_{ip} \cdot u_{pj}$

```
DO 40 p=1, k-1
IF (BTOL(BS(k,p))) THEN
ql=ql+1
IF (BTOL(BS(p,j))) THEN
qu=RS(p)+Q8SCNT(BS(p,1;j)))-1
sum(1;y)=NZS(1,ql;y)*NZS(1,qu;y)+sum(1;y)
ENDIF
ENDIF
40  CONTINUE
```

3) Obtain the k-th row of **U**

```
qq=Q8SCNT(BS(k,1;y))+qq
ql=ql+1
NZS(1,qq;y)=(NZS(1,qq;y)-sum(1;y))/NZS(1,ql;y)
ENDIF
30  CONTINUE
```

The Q8SCNT is one of the intrinsic functions of the CYBER 200 FORTRAN which is available on Cyber-205. Appendix B will give the illustration in detail of these intrinsic functions used in our simulation. Since the corresponding program of scalar version for simulation is too long, we are not going to present it here but put it in Appendix D.

When solving the linear equations by **LU** decomposition, we need to solve the eqution as in Eq.3.8. The elements of vector **x** and **y** can be obtained by the following formulas:

$$\mathbf{y}_k = \frac{1}{l_{kk}} (b_k - \sum_{p=1}^{k-1} l_{k,p} \mathbf{y}_p) \qquad (4.6)$$

$$\mathbf{x}_k = \mathbf{y}_k - \sum_{p=k+1}^{n} u_{k,p}\,\mathbf{x}_p \qquad (4.7)$$

The program for solving k-th element of **y** is as follows:

1) Calculate the sum of $l_{k,p} \cdot y_p$

```
      sum(1;y)=0.0
      dd=RS(K)-1
      DO 10 j=1, k-1
      IF (BTOL(BS(k,j))) THEN
      dd=dd+1
      sum(1;y)=NZS(1,dd;y)*YS(1,j;y)+sum(1;y)
      ENDIF
10    CONTINUE
```

Where YS in the program is the temporary vector **y**.

2) Obtain the k-th element of **y**

```
      dd=dd+1
      YS(1,k;y)=(brS(1,k;y)-sum(1;y))/NZS(1,dd;y)
```

The program for solving k-th element of **x** of subcircuits is as follows:

1) Calculate the sum of $u_{k,p} \cdot x_p$

```
      l=ns-k
      qq=RS(l)+Q8SCNT(RS(l,1;j))-1
      sum(1;y)=0.0
      DO 20 p=l+1, ns
      IF (BTOL(BS(l,p))) THEN
      qq=qq+1
      sum(1;y)=NZS(1,qq;y)*XS(1,p;y)+sum(1;y)
      ENDIF
20    CONTINUE
```

2) Obtain the k-th element of **x**

```
      XS(1,l;y)=YS(1,l;y)-sum(1;y)
```

From the XS we can obtain the values of the currents of the subcircuits. Then the equivalent current sources are calculated by Eq.2.27 , Eq.2.28 and Eq.2.29. This part of program is simple and we put it in Appendix C. The results of equivalent sources for subcircuits are stored in array EG and EJ.

4.3 Main Network Update Programs

After the values of the equivalent sources are found, there are two operations should be done. First the discrete equivalent sources for nonlinear elements in main circuit should be calculated. Second the equivalent sources of the subcircuits should be inserted into the main circuit, and then the main circuit is solved with the NEM method. We call these operations the main network update, which is similar to subnetwork update except that there is only one main circuit in main network update while there are y subcircuits in subnetwork update. It is difficult to optimize the vectorized program especially when the main circuit is small.

In simulation program the nonzero elements of **A** for main circuit are stored in the array NZ, the vector b on the right side of the equation $\mathbf{Ax} = \mathbf{b}$ is stored in the array br. The integer array D has the same structure as DS in subnetwork update. g^j and J^j contain the values of discrete equivalent sources in the main circuit, the integer arrays NZPc and bpc are used to indicate the position exchanges of them in NZ and br. The bit arrays NZP and brp are used to locate the positions of g^j and J^j. The program for calculating g^j and J^j is as follows:

1) Find the voltages accross the nonlinear elements

```
p=0
DO 10 i=1, nl
IF (D(i,4).EQ.0)) THEN
v=X(D(i,1))
ELSE
v=X(D(i,1))-X(D(i,2))
ENDIF
```

2) Calculate the polynomial of nonlinear elements

```
g=Q8VPOLY(v,PO(1,D(i,3);10);g)
I=Q8VPOLY(v,PO(11,D(i,3);10);I)
```

3) Obtain **J** and **G**

```
J(i)=g*v-I
p=p+1
G(p)=g
IF (D(i,4).EQ.0) GOTO 10
p=p+1
G(p)=-g
```

```
10    CONTINUE
```

Where nl is the number of nonlinear elements in main circuit. Q8VPOLY is one of the intrinsic functions of CYBER 200 FORTRAN for calculating the polynomial. Since the length of the polynomial is short in the simulation, the speedup for this function is poor. To optimize the vectorized program, we add two temporary arrays NZo and bro in the program. The program for inserting g^j and J^j into NZ and br is as follows:

1) Update the NZ

```
NZo(1;p)=Q8SCATR(G(1;p),NZPc(1;p);NZo(1;p))
G(1;p)=NZo(1;p)
NZ(1;m)=NZ(1;m)+Q8VXPND(G(1;p),NZP(!;m);NZo(1;m))
```

2) Update the br

```
bro(1;nl)=Q8VSCATR(J81;nl),bpc(1;nl);bro(1;nl))
J(1;nl)=bro(1;nl)
br(1;n)=br(1;n)+Q8VXPND(J(1;nl),bp(1;n);bro(1;n))
```

Where n is the dimension of **A** for the main circuit. Q8VSCATR and Q8VXPND are intrinsic functions which are illustrated in Appendix B.

The bit arrays NZQ and brq are used to locate the values of the equivalent sources of the subcircuits in NZ and br. The integer arrays NZQc and bqc are used to indicate the exchanges of these values in NZ and br. The bit array E is used to show whether the subcircuit is connected to datum of the main circuit. A temporary array EO is used to optimize the vectorization. Inserting of the equivalent sources can be done by the following operations:

1) Calculate the EO

```
p=0
DO 10 i=1, y
p=p+1
EO(p)=EG(i)
IF (.NOT.(BTOL(E(i)))) GOTO 10
p=p+1
EO(p)=-EG(i)
10    CONTINUE
```

2) Update the NZ

```
NZO(1;p)=Q8VSCATR(EO(1;p),NZQc(1;p);NZO(1;p))
EO(1;p)=NZO(1;p)
```

```
NZ(1;m)=NZ(1;m)+Q8VXPND(EO(1;p),NZQ(1;m);NZO(1;m))
```

3) Update the br

```
bro(1;y)=Q8VSCATR(EJ(1;y),bqc(1;y);br0(1;y))
EJ(1;y)=bro(1;y)
br(1;n)=br(1;n)+Q8VXPND(EJ(1;y),bq(1;n);bro(1;n))
```

Where m is the number of nonzero elements in NZ, p is a counter here.

4.4 Programs For Solving The Main Circuit

Since only one equation $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ is required for solving main circuit, it is difficult to vectorize the program. The temporary arrays sum, tu, tl, and NZT are used here to optimize the program. By Eq.4.6, $\mathbf{y}_k$ can be calculated after the k-th column of **L** is obtained. So we compute the **L**, **U**, and **Y** by one DO-loop for saving the CPU time. This part of program is as follows:

1) Generate the vector tu

```
      tu(1;k-1)=0.0
      DO 10 i=1, k-1
      IF (BTOL(B(i,k))) THEN
      qu=R(i)+Q8SCNT(B(i,1;k))-1
      tu(i)=NZ(qu)
      ENDIF
10    CONTINUE
```

2) Calculate the sum of $l_{i,p} \cdot u_{p,j}$ for **L**

```
      DO 20 i=k, n
      IF (BTOL(B(i,k))) THEN
      ql=R(i)

      tl(1;k-1)=Q8VXPND(NZ(ql;k-1),B(i,1;k-1);tl(1;k-1))
      sum(1)=Q8SDOT(tl(1;k-1),tu(1;k-1))
```

3) Obtain the **L**

```
      ql=Q8SCNT(B(i,1;k-1))+ql
      NZ(ql)=NZ(ql)-sum(1)
      ENDIF
20    CONTINUE
```

4) Calculate the sum of $l_{k,p} \cdot y_p$ for **y**

```
      ql=R(k)
```

```
tl(1;;k-1)=Q8VXPND(NZ(ql,k-1),B(k,1;k-1);tl(1;k-1))
sum(1)=Q8SDOT(tl(1;k-1),Y(1;k-1))
```

5) Obtain the **y**

```
dd=Q8SCNT(B(k,1;k-1))+ql
Y(k)=(br(k)-sum(1))/NZ(dd)
```

6) Calculate the sum $l_{i,p} \cdot u_{p,j}$ for **U**

```
IF (k.NE.n) THEN
ql=R(k)-1
dd=n-k
sum(1;dd)=0.0
DO 30 j=1, k-1
IF (BTOL(B(k,j))) THEN
ql=ql+1
qu=R(i)+Q8SCNT(B(j,1;k))

tu(1;dd)=Q8VXPND(NZ(qu;dd).B(j,k+1;dd);tu(1;dd))
WHERE (B(k,k+1;dd)) sum(1;dd)=sum(1;dd)+tu(1;dd)*NZ(ql)
ENDIF
30  CONTINUE
```

7) Obtain the **U**

```
nqu=Q8SCNT(B(k,k+1 dd))
qu=ql+1

NZT(1;nqu)=Q8VCMPRS(sum(1;dd),B(k,k+1;dd);NZT(1;nqu))
NZ(qu+1;nqu)=(NZ(qu+1;nqu)-NZT(1;nqu))/NZ(qu)
ENDIF
```

The program for solving the **x** is as following:

```
l=n-k
qq=R(l)+Q8SCNT(B(l,1;l)
tu(1;k)=Q8VXPND(NZ(qq;k),B(l,l+1;k);tu(1;k))
sum(1)=Q8SDOT(tu(1;k),X(l+1;k))
X(l)=Y(l)-sum(1)
```

The key parts of the simulation programs have been described in this chapter. Two detailed versions of the programs are given in Appendices C and D: one written in vector code and the other scalar code.

CHAPTER 5
CYBER-205 SIMULATION RESULTS

We use large-scale circuit example to obtain the simulation results on the Cyber-205. Various programs for different purposes are used in the simulation experiments. Section **5.1** presents the example circuit and initial conditions. Section **5.2** presents the results for different program versions, and analyzes the speedup performance of the MNA. The implications and further improvements are then elaborated.

5.1 The Sample Circuit Being Simulated

An example circuit is used for the circuit analysis simulation. The main circuit with the equivalent current sources of the subcircuits is shown in Fig.5.1. N is the number of the subcircuits which are connected to the main circuit. In our simulation program, N can vary from 1 to 1000. All the subcircuits have the same structure shown in Fig.5.2, and they are connected together in parallel, which means that they have the same parameters. This is for simplifying the input and output procedure such that we only need to input and output the variables in one of the subcircuits. Assume the initial guess for nodal voltages in main circuit is

$$v_1^0 = 15.0 \qquad v_2^0 = 14.3$$

$$v_3^0 = 6.0 \qquad v_4^0 = 5.0$$

$$v_5^0 = 0.0$$

The initial guess for nodal voltages in subcircuits is

$$v_1^0 = 15.0 \qquad v_2^0 = 5.0$$

$$v_3^0 = 7.0 \qquad v_4^0 = 4.0$$

5.2 Numerical Results And Speedup Analysis

All input data and circuit simulation results are given in Appendix A. The Table A3 lists the final results of main circuit with different number of the

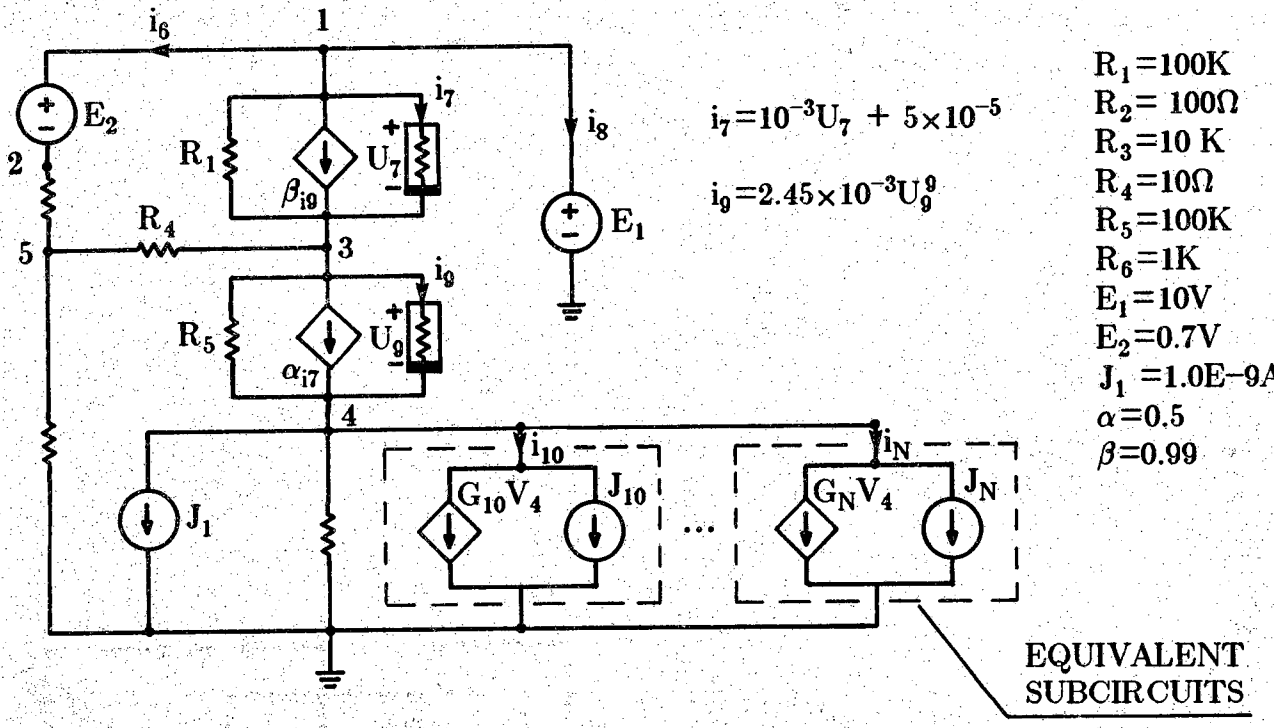
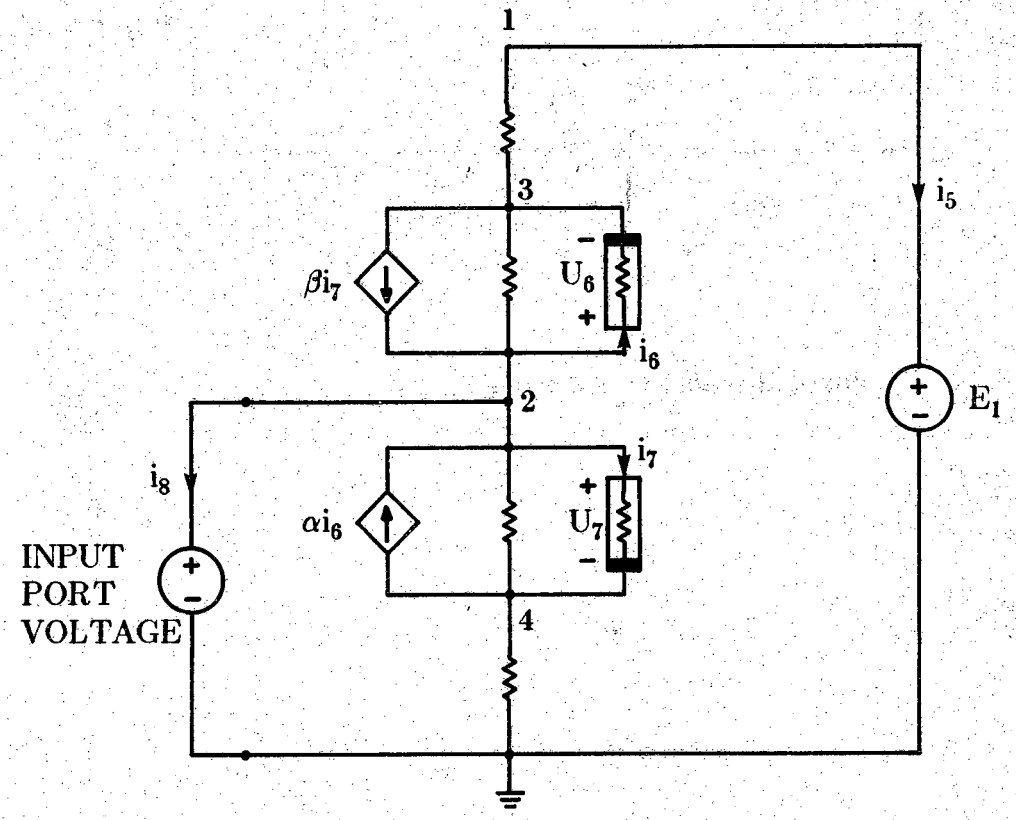


Figure 5.1 The main circuit



$i_6 = 2\times10^{-4}U_6 - 5\times10^{-6}$

$i_7 = 2.45\times10^{-3}U_7^9$

$R_1 = 50\Omega$
$R_2 = 100K$
$R_3 = 100K$
$R_4 = 500\Omega$
$E_1 = 15V$
$\alpha = 0.5$
$\beta = 0.99$

Figure 5.2 One subcircuit

subcircuits. Since the variables in each subcircuits are the same, the results for only one of the subcircuits are listed here. All subcircuits are connected in parallel. Table A4 lists the variables in one of the subcircuits with different number of subcircuits in main circuit.

In Table A5, the average CPU time in each calculation step is listed, where the time unit used is second. The input time and the output time are not included in the table. Since the CPU time for each calculation step includes some operating system overhead time, doing the same operation may take different time. So we use the average values in the tables. There are several program versions in our simulation experiments. Table A5.a lists the CPU time where the vector pipelines of Cyber-205 are used. Table A5.b lists the CPU time where only the scalar processor of Cyber-205 is used. The number of subcircuits N used in vector version simulations takes value 1, 10, 50, 100, 200, 500, and 1000. The same values for N are used in the scalar version except 1000 because the CPU time of the Cyber-205 is expensive. Table A5.a and Table A5.b show that the performance of updating and solving the subcircuits in the vector version is even worse than that in the scalar version when the circuit contains only one subcircuit. This is because the number of the subcircuits is too small to utilize the pipeline. Table A5.c lists the CPU time for using the scalar processor of Cyber-205 and without any bit processing instructions used. As a reference, Table A5.d lists the CPU time for using the VAX 11/780 machine. Due to limitation by the memory space to user, the number of subcircuits in Table A5.c and Table A5.d is only up to 200. The average speedup of the vectorized program versus the program of the scalar version is shown by some curves. The curves of speedup for each calculation step are shown in from Fig.5.3 to Fig.5.7 . The speedup increases rapidly when N increases. Figure 5.3 shows that the speedup of the subcircuit update is about 100 when N equals to 500. Figure 5.5 shows that the speedup for solving the subcircuits is about 10. Since there is only one main network, it is difficult to optimize the vectorized program. Figure 5.4 shows the speedup of main network update is about 10 when N is 500.

The dimension of the matrix for solving the main circuit is much larger than that for the subcircuits, so solving main circuit takes most of the total CPU time. To optimize this part of program is very important even though it may be difficult. We add some temporary vectors to improve the efficiency of the program, which will require some extra memory space, but it can save the CPU time. Figure 5.6 shows the speedup for solving main circuit is more than 100. This is significant. Figure 5.7 shows the speedup for back-substitution is about 20. Figure 5.9 shows the overall speedup is about 100. From the curves

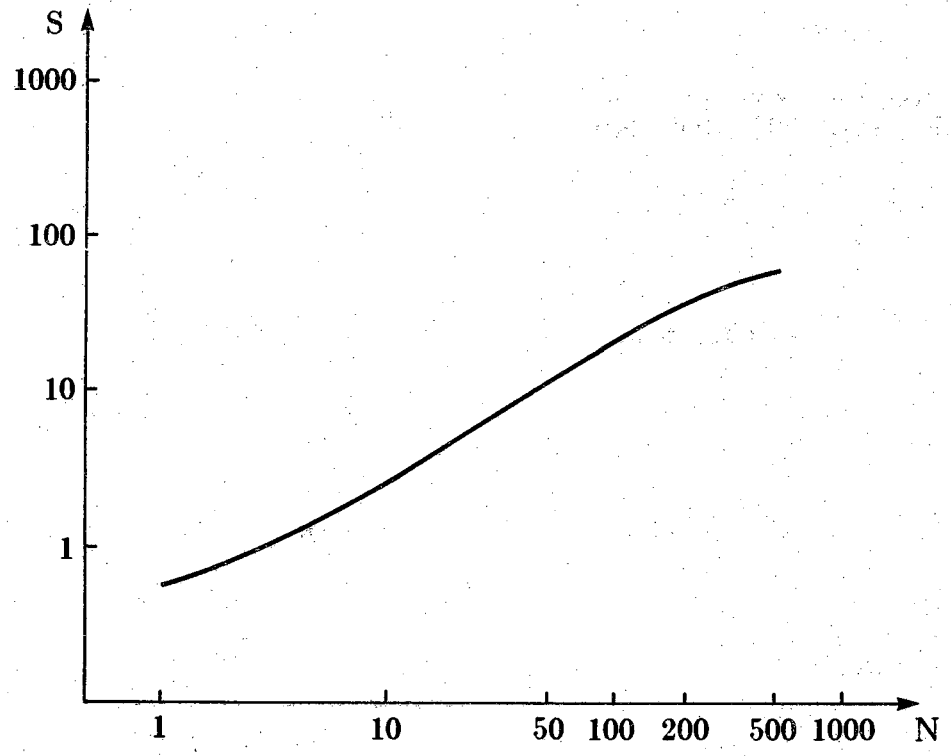


Figure 5.3 Speedup in subnetworks updates

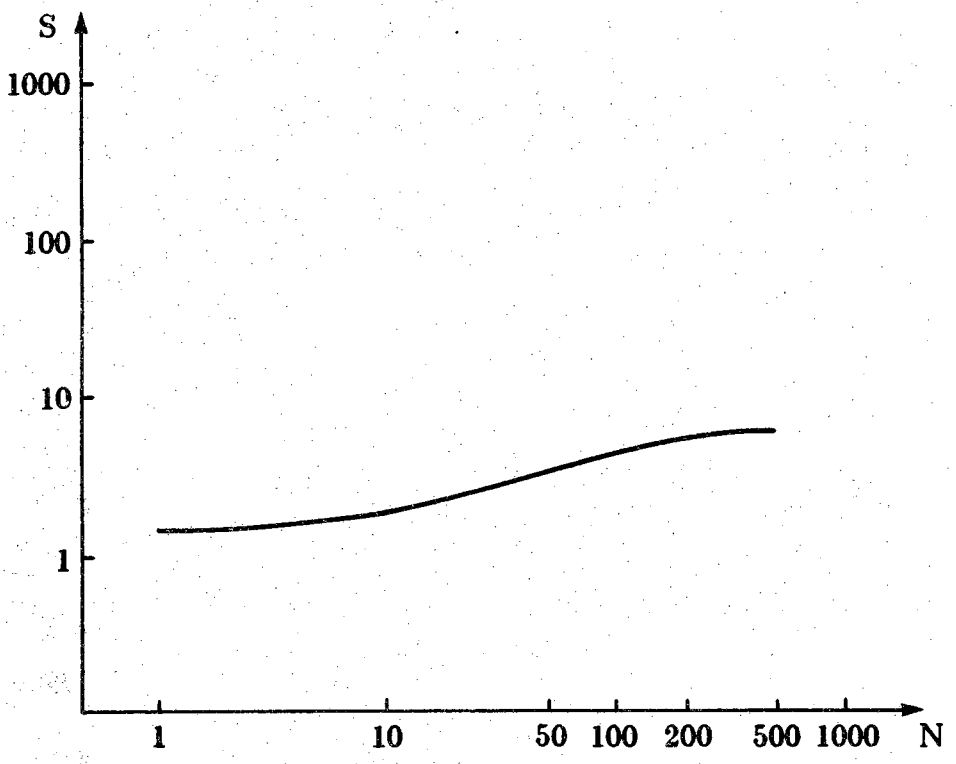


Figure 5.4 Speedup in main networks updates

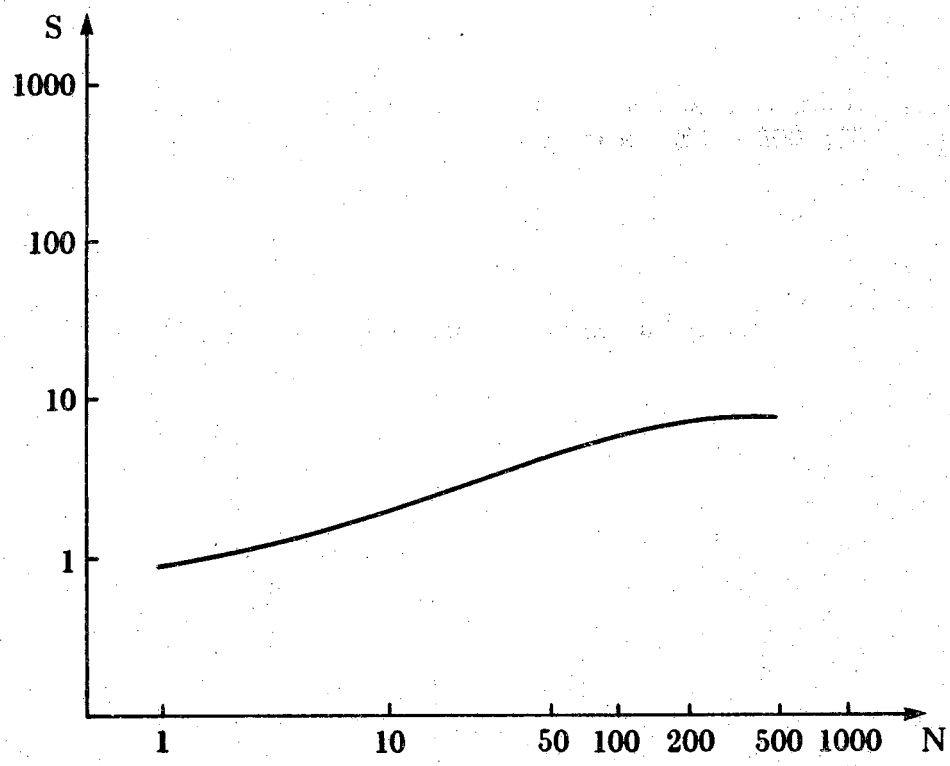


Figure 5.5 Speedup in solving subcircuits

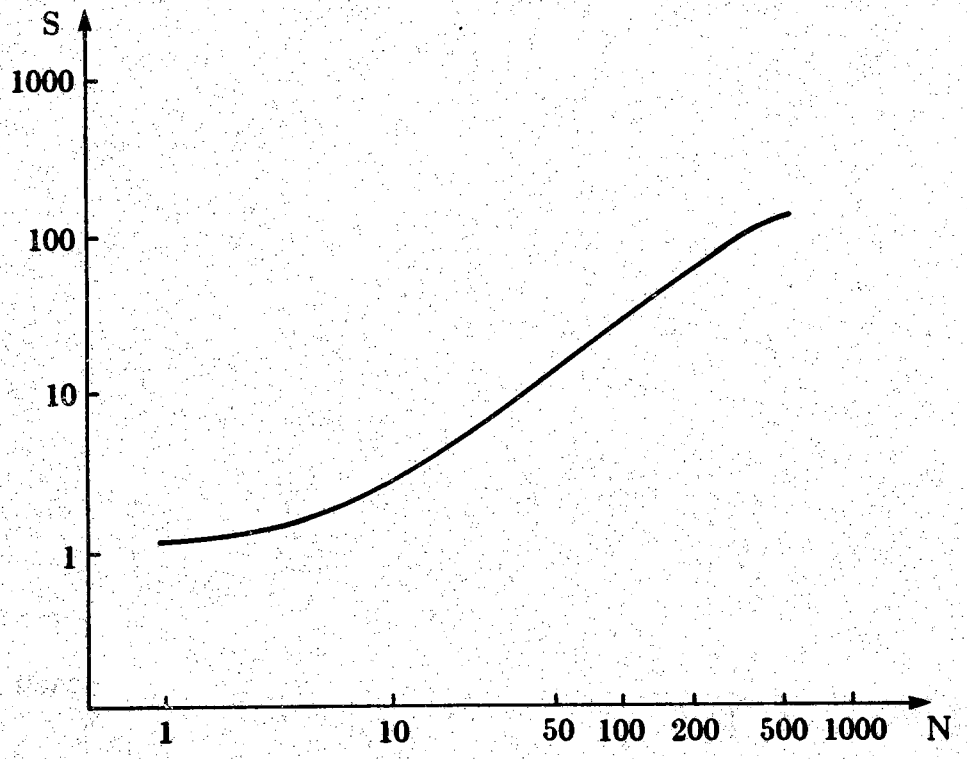


Figure 5.6 Speedup in solving the main circuit

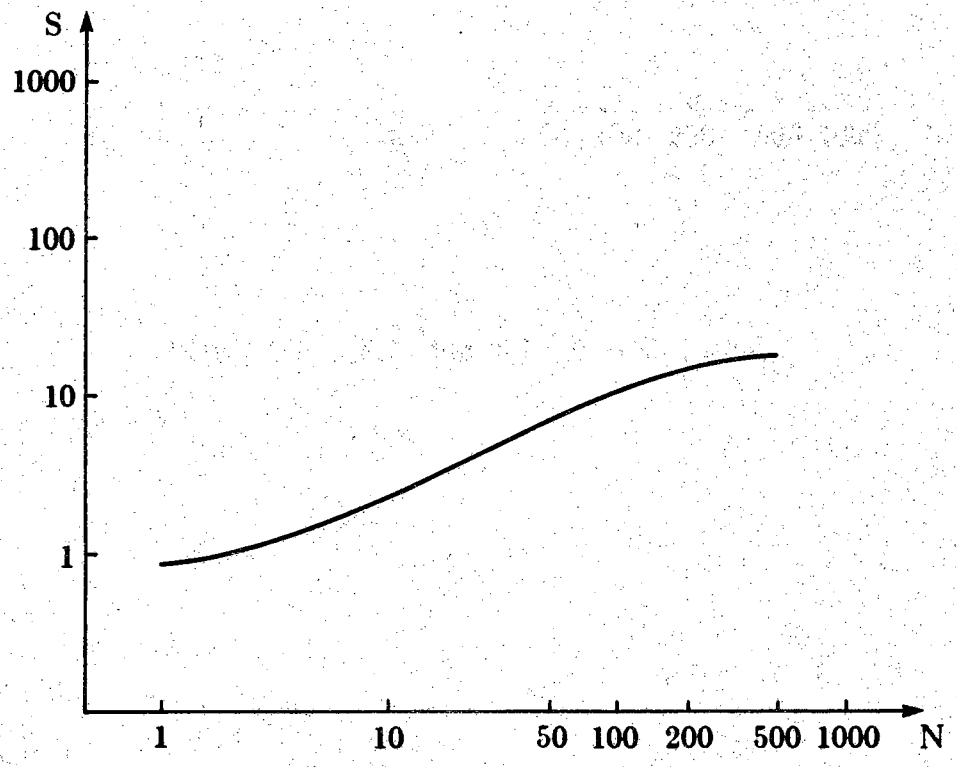


Figure 5.7 Speedup in back-substitution

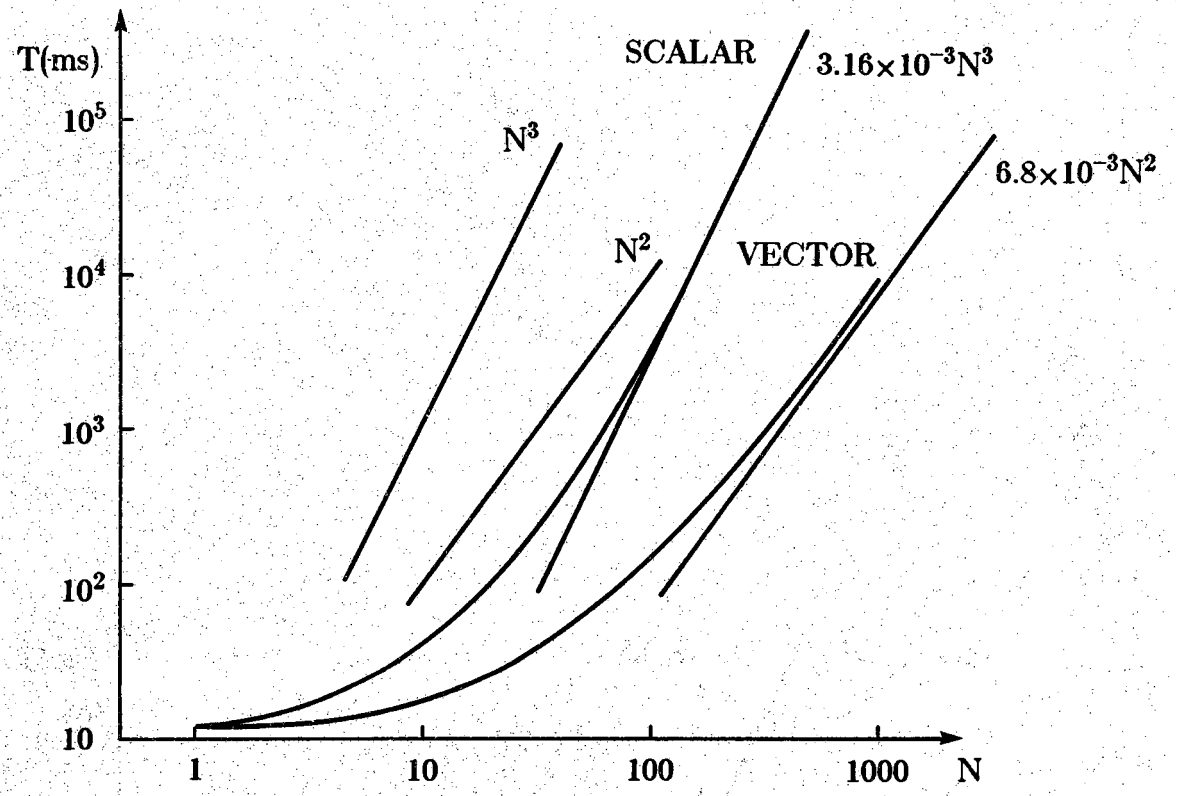


Figure 5.8 CPU times for scalar and vector programs

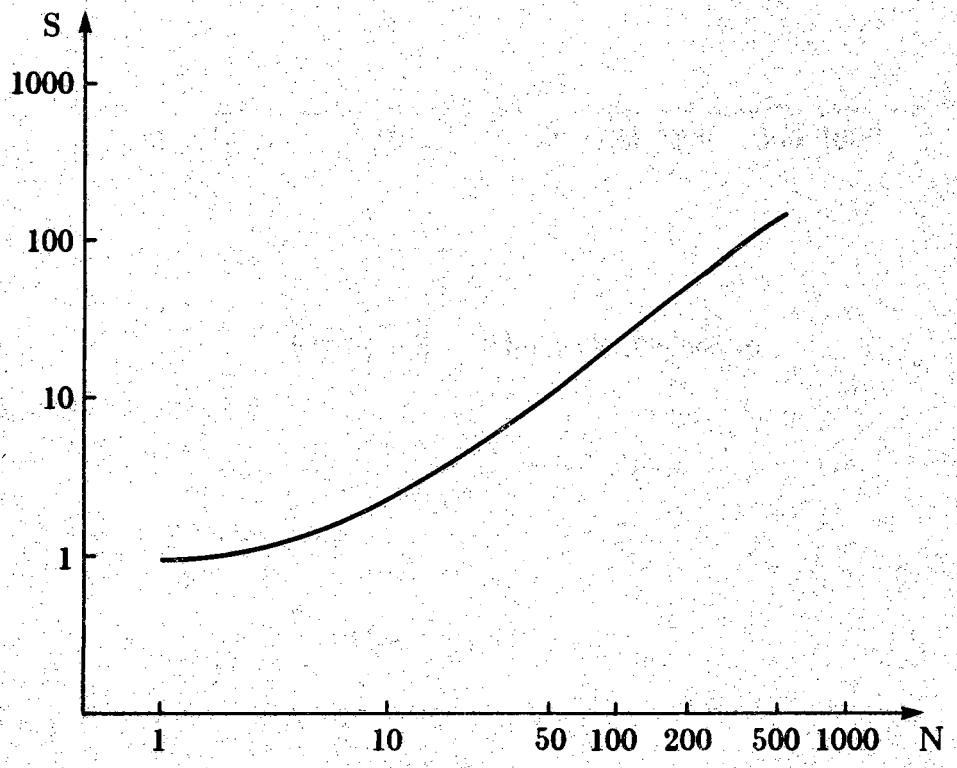


Figure 5.9 The overall speedup

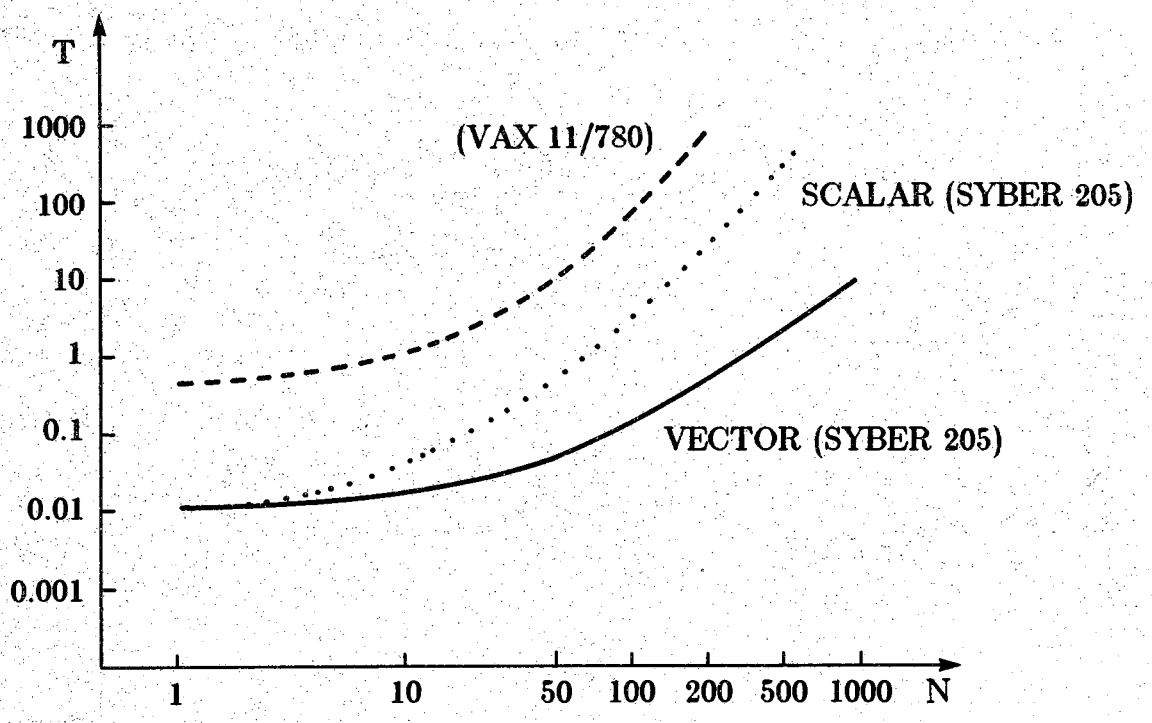


Figure 5.10 The CPU times needed in various simulations

we can see, the speedup increases rapidly at the beginning, and slows down when N becomes very large. This means it may tend to a constant. From Figure 5.8 we can see, when the subcircuit number N is very large the CPU time for vector version program is approximately $6.8\times10^{-3}N^2$, and for scale version is about $3.16\times10^{-3}N^3$. Figure 5.10 gives the curves of the CPU time for different versions of the program. As a reference, the CPU time for VAX 11/780 is shown on the top of the figure.

5.3 Implications And Further Improvements

From the results of simulation we can see, when the vector pipelines of Cyber-205 is used to solve the large-scale circuit by MNA, a high speedup can be achieved. Since the setup time for the pipelines in Cyber-205 is long, when the number of subcircuits is smaller, the speedup is poor. Figure 5.9 shows that the increase of the speedup will slow down when N becomes very large, which means the speedup might tend toward a constant with a very large system. Because the CPU time of Cyber-205 is expensive, N takes limited values up to only 500 in the curves, and the corresponding overall speedup is more than 100. If more subcircuits were in the main circuit, the speedup might be higher than this. The vectorization in each calculation step is different, and the speedups for these steps are different too. In Fig.5.8 the total CPU time includes the time for calculation, data access and system overhead, such as page fault handling etc., the input-output time is excluded here. For the example circuits, the total time complexity for vector version is $O(N^2)$, and for scalar version it is $O(N^3)$.

A considerable amount of time has been used for page fault handling in our vectorized simulation. Table A6 shows that the execution time for scalar code is 340 seconds when N is 500, and the time for page fault handling is 12 seconds. For vector code, the time is reduced to 4 seconds, but the time for page fault handling does not change. This means that reducing page faults becomes very important in the vectorized MNA.

CHAPTER 6
CONCLUSIONS AND SUGGESTIONS

Our research findings are summarized below. Several suggestions are made for those who wish to conduct further studies on vectorized circuit analysis using supercomputers.

6.1 Concluding Remarks

A new Newton algorithm has been proposed to perfrom circuit analysis on vector computers. The subcircuits are treated as the elements of a vector and are processed by a vector pipeline. Higher efficiency can be achieved over existing algorithms. We revealed the speedup advantages of the new algorithm The symbolic processing and row column exchanges in MNA are exactly the same as in the SLNA and MLNA. The major advantages of the MNA are summarized below:

1) A large circuit is partitioned into multiple levels. There is only one main loop in the program. The number of iterations in the main loop is a constant and does not increase with the number of levels. The complexity in each iteration increases linearly with respect to the number of levels in the circuit.

2) The main loop of the MNA has a quadratic convergence rate in most cases. This is important for the overall speedup of the circuit simulation program, especially when the circuits are very large. This convergence rate is faster than the pairwise quadratic convergence of MLNA reported in Lin [5].

3) The nonlinear equations for the main circuit and the subcircuits are not needed in the MNA. Only the parameters of circuit elements are used. This makes the input of the data of a large circuit much easier. The calculations for all the Jacobian matrices can be avoided in MNA which saves CPU time.

4) In our simulation experiments, the speedup is approximately 100, when 500 subcircuits are in the main circuit. The modified bit matrix structure is attractive for large-scale circuit simulation. Only the nonzero elements and bit mask matrix have to be stored in memory. This results in higher efficiency on the Cyber-205 supercomputer.

5) The vector pipelines of Cyber-205 have a long setup time. If the circuit contains a few subcircuits, the speedup would be poor. When the circuit contains many identical subcircuits, a significant speedup can be expected. The larger is the number of subcircuits in the circuit, the higher will be the speedup.

6.2 Suggestions For Further Research

From the results of our simulation experiments, two suggestions are made for continued studies:

1) Since the vectorized programs are written in CYBER 200 FORTRAN, they have to be translated to machine language by the compiler of Cyber-205. Therefore, the overall CPU time needed depends on the efficiency of the compiler. For example, page fault handling will demand CPU time. If the page size and page allocation are reasonable, the page fault occurrences may be reduced. Thus the CPU time will be also reduced. This problem can be alleviated by writing the program in assemble language. This requires us to know the machine architecture and operating system in more detail.

2) The data input/output demands long time delays. Since the Cyber-205 CPU time is expensive, it may be more advantageous to use a small scalar computer to perform the input/output functions. The circuits in our simulations are partitioned into only two levels. Since the number of iterations in the MNA is a constant, higher benefit may be obtained if more levels are used. The latency technique is not considered either. Latency may further improve the efficiency of the MNA on a vector supercomputer [20-23].

LIST OF REFERENCES

[1] N. B. G. Rabbat, A. L. Sangiovanni-Vincentelli, and H. Y. Hsiech, "A Multilevel Newton Algorithm and Latency for the Analysis of Large-Scale Nonlinear Circuit in the Time Domain," *IEEE Trans. on Circuits and Systems* Vol CAS-26, pp. 733-741, Sept. 1979.

[2] J. L. Algiere, and K. Hwang,"Sparse Matrix Techniques for Largr-Ssale Circuit Analysis on the Cyber 205 Vector Processor," School of Electrical Engineering, Purdue University, *TR-EE 83-40,* October 1983.

[3] Chung-Wen Ho, Albert E. Ruehli, and A. Brennan,"The Modified Nodal Approach to Network Analysis," *IEEE Trans. on Circuit and Systems,* Vol. CAS-22, pp. 504-509, June. 1975.

[4] Kai Hwang, "Vectorization Methods for Larg-Scale Circuit Analysis on Vector Supercomputers," School of Electrical Engineering, Purdue University, *TR-EE 83-1,* Jan. 1983.

[5] P. M. Lin, and A. W. Nordsiech, "A Study of the Convergence Rate of a Multilevel Newton Algorithm," School of Electrical Engineering, Purdue Univesity, *TR-EE 82-24,* November. 1982.

[6] L. O. Chua, and P. M. Lin, *Computer-Aided Analysis of Electronic Circuit: Algorithms and Techniques,* Prentice-Hall, 1975.

[7] W. J. Mccalla, and D. O. Pederson, "Elements of Computer-Aided Circuit Analysis," *IEEE Trans. on Circuit Theory,* Vol. CT-18, pp. 14-26, Jan.1971

[8] L. M. Ni and Kai Hwang, "Vector Reduction Techniques for Arithmetic Pipelines," *IEEE Trans. on Computers,* March 1985.

[9] Control Data Corporation,CDC CYBER 200 FORTRAN VERSION 2 Reference Mannal, Oct. 1982.

[10] Kai Hwang, and F. A. Biggs, *Computer Architecture and Parallel Processing,* McGrw-Hill Book Co., New York 1984.

[11] T. Putnam, B. Whitson, and D. Seaman "Introduction to the CYBER 205," PUCC User Services, Purdue Univesity.

[12] Ibrahim N. Hajj, Ping Yang, and Timolthy N. Trich, "Avoiding zero pivots in the Modified Nodal Approach," *IEEE Trans. on Circuits and Systems,* Vol.CAS-28, pp. 271-279, April. 1978.

[13] Robert D Bery, "An Optimal Ordering of Electronic Circuit Equations for a Sparse Matrix Solution," *IEEE Trans. on Circuit Theory,* Vol.C7-18, pp. 40-50, Jan. 1971.

[14] A. W. Nordsiech, and P. L. Lin, "MULNA:A Program for a Multilevel Newton Algorithm," School of Electrical Engineering, Purdue University, *TR-EE 82-23,* August. 1982.

[15] William T. Weeks et al., "Algorithm for ASTAP-A Network Analysis Program," *IEEE Trans. on Circuit Theory,* Vol.CT-20, pp. 628-634, Nov. 1973.

[16] R. K. Brayton, F. G. Gustarson, and G. D. Hachtel, "A New Efficient Algorithm for Solving Differential Algebraic Systems Using Implicit Backward Differential Formulas," *Proc. IEEE,* Vol. 60, pp. 98-108, Jan. 1972.

[17] F. F. Wu, "Solution of Large Scale Networks by Tearing," *IEEE Trans. on Circuits and Systems,* Vol. Cas-23, No. 12, pp. 706-713, December 1976.

[18] A. Sangiovanni-Vincentelli, L. K. Chen, and L. O. Chua, "A New Tearing Approach - Node Tearing Nodal Analysis," *Proc. IEEE Int. Symp. on Circuits and Systems,* pp. 143-147, April 1977.

[19] F. Odeh and D. Zein, "A Semi-Direct method for Modular Circuits," *Proc. 1983 Int'l. Symp. on Circuits and Systems,* pp. 226-229, May 1983.

[20] K. Hwang, "Multiprocessor Supercomputers and Scientific Applications," to appear *IEEE Computer Magazine,* July 1985.

[21] K. Hwang and Y. H. Cheng, "Partitioned Matrix Algorithms for VLSI Arithmetic Systems," *IEEE Trans. Computers,* Oct. 1982.

[22] K. Hwang (Editor), *Supercomputers: Design and Applications,* IEEE Computer Society Press, August, 1984.

[23] M. M. Hassoun and P. M. Lin, "A Study of a Semi-Direct Method for Computer Analysis of Large Scale Circuits," School of Electrical Engineering, Purdue University, TR-EE 84-46, December 1984.

APPENXIX A: INPUT DATA SETS AND SIMULATION RESULTS

The main circuit and subcircuits are all characterized by the linear equations as the form $\mathbf{A}\mathbf{x} = \mathbf{b}$ in MNA. The matrix **A** for the subcircuits is shown in Fig.A(a). Since the symbolic processing and the row column exchange do not be included in our simulations, these processing should be done before the data input. After these processing, the **A** becomes that in Fig.A(b). The symbol $\times$ in Fig.A(b) indicate that there is a fill-in element. The bit mask matrix for subcircuit **BS** is as shown in Fig.A(c). In Fig.A(d) are the vector **b** for subcircuits, the **b** after row exchange, and the bit array bpS, the integer array RS indicating the first nonzero element in each row of **A**. The matrix **A** for main circuit is shown in Fig.A(e). Its dimension can be changed up to 1000. After symbolic processing and row column exchange, **A** becomes that in Fig.A(f). Figure A(g) shows the bit mask matrix for main circuit **B**. The vectors **b** before and after row change, the vectors bg and bp, and the integer array C for the main circuit are shown in Fig.A(h).

Table A1 and A2 list the simulation results of MNA and SLNA as reported in Chapter 3. The large scale circuit simulation results on Cyber-205 are given in Table A3 to A6.

Table A6. Times Needed for User, System and Page Faults Handling

Number of Subcircuits	User CPU Time*(s)		System CPU Time(s)		Net Page Faults(s)	
	Scalar	Vector	Scalar	Vector	Scalar	Vector
1	1.9	1.9	1.0	1.0	11.2	11.9
10	2.0	1.9	1.0	1.0	11.3	11.9
50	2.5	1.9	1.0	1.0	11.3	11.9
100	5.5	2.0	1.0	1.0	11.3	11.9
200	26.0	2.4	1.0	1.0	11.3	11.9
500	345.4	4.6	1.0	1.0	11.4	12.0
1000	–	12.0	–	1.0	–	12.3

*Input/Output Times Are Included

$$
\begin{vmatrix}
\frac{1}{R_1} & 0 & \frac{-1}{R_1} & 0 & 0 & 1 & 1 & 1 & \beta & 0 & \dots & 0 \\
0 & \frac{1}{R_2} & 0 & 0 & \frac{-1}{R_2} & -1 & 0 & 0 & 0 & 0 & \dots & 0 \\
\frac{-1}{R_1} & 0 & \frac{1}{R_1}+\frac{1}{R_4}+\frac{1}{R_5} & \frac{-1}{R_5} & \frac{-1}{R_4} & 0 & \alpha-1 & 0 & 1-\beta & 0 & \dots & 0 \\
0 & 0 & \frac{-1}{R_5} & \frac{1}{R_5}+\frac{1}{R_6} & 0 & 0 & -\alpha & 0 & -1 & 1 & \dots & 1 \\
0 & \frac{-1}{R_2} & \frac{-1}{R_4} & 0 & \frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4} & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \\
g_7^j & 0 & -g_7^j & 0 & 0 & 0 & -1 & 0 & 0 & 0 & \dots & 0 \\
0 & 0 & g_8^j & -g_8^j & 0 & 0 & 0 & 0 & -1 & 0 & \dots & 0 \\
0 & 0 & 0 & G_{10}^j & 0 & 0 & 0 & 0 & 0 & -1 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & & \\
0 & 0 & 0 & G_n^j & 0 & 0 & 0 & 0 & 0 & 0 & \dots & -1
\end{vmatrix}
$$

(e)

$$
\begin{vmatrix}
-1 & 0 & 0 & 0 & 0 & 0 & 0 & G_{10}^j & 0 & 0 & 0 \\
 & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & G_n^j & 0 & 0 & 0 \\
0 \dots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 \dots & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 \dots & 0 & g_7^j & 0 & -g_7^j & 0 & 0 & -1 & 0 & 0 & 0 \\
0 \dots & 0 & 0 & \frac{-1}{R_2} & \frac{-1}{R_4} & \frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4} & 0 & X & 0 & 0 & 0 \\
0 \dots & 0 & 0 & \frac{1}{R_2} & 0 & \frac{-1}{R_2} & -1 & X & 0 & 0 & 0 \\
0 \dots & 0 & \frac{-1}{R_1} & 0 & \frac{1}{R_1}+\frac{1}{R_4}+\frac{1}{R_5} & \frac{-1}{R_4} & 0 & \alpha-1 & 0 & \frac{-1}{R_5} & 1-\beta \\
0 \dots & 0 & \frac{1}{R_1} & 0 & \frac{-1}{R_1} & 0 & 1 & 1 & 1 & X & \beta \\
0 \dots & 0 & 0 & 0 & g_9^j & 0 & 0 & X & 0 & -g_9^j & -1 \\
1 \dots & 1 & 0 & 0 & \frac{-1}{R_5} & 0 & 0 & -\alpha & 0 & \frac{1}{R_5}+\frac{1}{R_6} & -1
\end{vmatrix}
$$

(f)

$$\begin{vmatrix}
\frac{1}{R_1} & 0 & \frac{-1}{R_1} & 0 & 1 & 0 & 0 & 0 \\
0 & \frac{1}{R_2}+\frac{1}{R_3} & \frac{-1}{R_2} & \frac{1}{R_3} & 0 & 1-\alpha & 1-\beta & 1 \\
\frac{-1}{R_1} & \frac{-1}{R_2} & \frac{1}{R_1}+\frac{1}{R_2} & 0 & 0 & -1 & \beta & 0 \\
0 & \frac{-1}{R_3} & 0 & \frac{1}{R_3}+\frac{1}{R_4} & 0 & \alpha & -1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & g_6^j & -g_6^j & 0 & 0 & -1 & 0 & 0 \\
0 & g_7^j & 0 & -g_7^j & 0 & 0 & -1 & 0
\end{vmatrix}$$

(a)

$$\begin{vmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & g_6^j & -g_6^j & 0 & 0 & -1 & 0 & 0 \\
0 & g_7^j & 0 & -g_7^j & 0 & 0 & -1 & 0 \\
\frac{1}{R_1} & 0 & \frac{-1}{R_1} & 0 & 1 & X & 0 & 0 \\
\frac{-1}{R_1} & \frac{-1}{R_2} & \frac{1}{R_1}+\frac{1}{R_2} & 0 & 0 & -1 & \beta & 0 \\
0 & \frac{-1}{R_3} & 0 & \frac{1}{R_3}+\frac{1}{R_4} & 0 & \alpha & -1 & 0 \\
0 & \frac{1}{R_2}+\frac{1}{R_3} & \frac{-1}{R_2} & \frac{-1}{R_3} & 0 & 1-\alpha & 1-\beta & 1
\end{vmatrix}$$

(b)

$$\begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{vmatrix}$$

(c)

$$\begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \\ E_1 \\ E_2 \\ -J_6^j \\ -J_7^j \end{vmatrix} \quad \begin{vmatrix} E_1 \\ E_2 \\ -J_6^j \\ -J_7^j \\ 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} \quad \begin{vmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} \quad \begin{vmatrix} 1 \\ 2 \\ 3 \\ 6 \\ 9 \\ 13 \\ 18 \\ 22 \end{vmatrix}$$

(d)

$$\begin{vmatrix}
1 & & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & \ldots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ldots & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \ldots & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & \ldots & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & \ldots & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & \ldots & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & \ldots & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & \ldots & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & \ldots & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1
\end{vmatrix}$$

(g)

0	$-J_{10}^{j}$	1	0	10
0		⋮	⋮	⋮
0	$-J_{n}^{j}$	1	0	n
$-J_1$	E_1	0	0	1
0	E_2	0	0	2
E_1	$-J_{7}^{j}$	0	1	3
E_2	0	0	0	5
$-J_{7}^{j}$	0	0	0	6
$-J_{8}^{j}$	0	0	0	7
$-J_{10}^{j}$	0	0	0	8
⋮	$-J_{9}^{j}$	0	1	4
$-J_{n}^{j}$	$-J_1$	0	0	9

(h)

Fig. A The input data sets: (a) to (h)

Table A1. Results of the MNA Simulation

Number of iterations (j)	Variables in the main circuit			Varianbes in the subcircuits		
	V_1^j	V_2^j	i_3^j	V_3^j	G_s^j	J_s^j
0	7.0	1.0		0.0	2.0	-1.0
1	7.0	2.666667	-4.333333	2.666667	6.175439	-8.233918
2	7.0	2.123064	-4.876935	1.458028	4.990777	-5.050256
3	7.0	2.011470	-4.988530	1.056501	4.701707	-4.4045713
4	7.0	2.000203	-4.999796	1.001090	4.667315	-4.334630
5	7.0	2.000000	-5.000000	1.000000		

Table A2. Results of the SLNA Simulation

Number of iterations (j)	Variables in the circuit			
	V_1^j	V_2^j	V_3^j	i^j
0	7.0	1.0	0.0	
1	7.0	2.666667	2.666667	-4.333333
2	7.0	2.123064	1.458028	-4.876936
3	7.0	2.011470	1.056501	-4.988530
4	7.0	2.000203	1.001090	-4.999796
5	7.0	2.000000	1.000000	-5.000000

Table A3. Results Obtained in the Main Circuit

Number of subcircuits	Main Circuit Variables				
	i_{10}	V_1	V_2	V_3	V_5
1	0.00023194	15.0	14.3	14.21430961	14.20918220
10	0.00022998	15.0	14.3	14.21217756	14.20724574
50	0.00022328	15.0	14.3	14.20303835	14.19894491
100	0.00021724	15.0	14.3	14.19215019	14.18905557
200	0.00020844	15.0	14.3	14.17156648	14.17036002
500	0.00019112	15.0	14.3	14.11602041	14.11990955
1000	0.00017209	15.0	15.3	14.03710825	14.04823638

Number of subcircuits	Main circuit Variables				
	i_6	i_7	i_8	V_4	i_9
1	0.00090818	0.00128569	-0.0151450	13.49213632	0.01307401
10	0.00092754	0.00128782	-0.01719880	13.47826175	0.01512683
50	0.00101055	0.00129696	-0.02601451	13.43076520	0.02393841
100	0.00110944	0.00130785	-0.03653093	13.38800212	0.03445006
200	0.00129640	0.00132843	-0.05643029	13.32564756	0.05434058
500	0.00180090	0.00138398	-0.11017436	13.20295807	0.10806125
1000	0.00251765	0.00146289	-0.18656576	13.06817581	0.18441980

Table A4. Results Obtained in One Subcircuit

Number of Subcircuits	Subcircuits Variables			
	V_1	V_2	V_3	V_4
1	15.0	13.49213632	13.74011897	12.7147804
10	15.0	13.47826175	13.74139862	12.70100470
50	15.0	13.43076520	13.74577911	12.56384735
100	15.0	13.38800212	13.74972294	12.61139075
200	15.0	13.32564756	13.75547342	12.54948480
500	15.0	13.20295807	13.76678753	12.42768413
1000	15.0	13.06817581	13.77921587	12.29388772

Number of Subcircuits	Subcircuits Variables			
	i_5	i_6	i_7	i_8
1	-0.02519762	-0.00005460	0.02539449	-0.00023194
10	-0.02517203	-0.00005763	0.02536542	-0.00022998
50	-0.02508447	-0.0000680	0.02526592	-0.00022328
100	-0.02500554	-0.00007734	0.02517634	-0.00021724
200	-0.02489053	-0.00009097	0.02504573	-0.00020844
500	-0.02466425	-0.00011777	0.02478873	-0.00019112
1000	-0.02441568	-0.00014721	0.02450643	-0.00017209

Table A5. CPU Times for the Execuation of Various Programs

(a) Vector Code (Cyber 205)

Number of Subcircuits	Update the Subcircuits	Solving the Subcircuits	Update the Main Circuit	Solving the Main Circuit	Substitution Time	Total Time
1	3.24×10^{-4}	1.01×10^{-3}	1.2×10^{-4}	6.53×10^{-4}	2.82×10^{-4}	1.2×10^{-2}
10	3.39×10^{-4}	1.20×10^{-3}	1.51×10^{-4}	1.42×10^{-3}	3.03×10^{-4}	1.7×10^{-2}
50	3.55×10^{-4}	2.01×10^{-3}	2.11×10^{-4}	8.00×10^{-3}	3.98×10^{-4}	5.49×10^{-2}
100	4.23×10^{-4}	3.08×10^{-3}	2.99×10^{-4}	2.37×10^{-2}	5.37×10^{-4}	1.40×10^{-1}
200	4.77×10^{-4}	5.12×10^{-3}	4.46×10^{-4}	7.96×10^{-2}	7.63×10^{-4}	4.32×10^{-1}
500	7.34×10^{-4}	1.13×10^{-2}	9.16×10^{-4}	4.44×10^{-1}	1.50×10^{-3}	2.29
1000	1.13×10^{-3}	2.17×10^{-2}	1.72×10^{-3}	1.72	2.72×10^{-3}	8.76

(b) Scalar Code (Cyber 205)

1	1.60×10^{-4}	9.35×10^{-4}	2.11×10^{-4}	8.7×10^{-4}	2.42×10^{-4}	1.2×10^{-2}
10	9.16×10^{-4}	2.40×10^{-3}	2.90×10^{-4}	4.13×10^{-3}	7.39×10^{-4}	4.24×10^{-2}
50	4.32×10^{-3}	9.21×10^{-3}	7.49×10^{-4}	1.07×10^{-1}	2.99×10^{-3}	6.21×10^{-1}
100	8.58×10^{-3}	1.77×10^{-2}	1.32×10^{-3}	6.7×10^{-1}	5.8×10^{-3}	3.52
200	1.71×10^{-2}	3.47×10^{-2}	2.46×10^{-3}	4.73	1.15×10^{-2}	23.96
500	4.27×10^{-2}	8.58×10^{-2}	5.90×10^{-3}	68.43	2.84×10^{-2}	342.95

(c) Scalar Code (Cyber 205 without use of bit processing)

1	3.40×10^{-4}	1.03×10^{-3}	1.39×10^{-4}	6.57×10^{-4}	2.83×10^{-4}	1.2×10^{-2}
10	1.11×10^{-3}	2.21×10^{-3}	3.2×10^{-4}	3.78×10^{-3}	6.81×10^{-4}	4.06×10^{-2}
50	5.30×10^{-3}	9.04×10^{-3}	8.36×10^{-4}	8.92×10^{-2}	2.94×10^{-3}	5.37×10^{-1}
100	1.05×10^{-2}	1.76×10^{-2}	1.48×10^{-3}	6.75×10^{-1}	5.76×10^{-3}	3.55
200	2.10×10^{-2}	3.46×10^{-2}	2.77×10^{-3}	4.78	1.14×10^{-2}	24.25

(d) Scalar Code (VAX 11/780)

1	1.67×10^{-2}	2.67×10^{-2}	0.0	4.33×10^{-2}	3.33×10^{-3}	4.50×10^{-1}
10	2.67×10^{-2}	6.67×10^{-2}	1.00×10^{-2}	8.33×10^{-2}	2.33×10^{-2}	1.67
50	1.30×10^{-1}	2.37×10^{-1}	1.30×10^{-2}	1.81	6.00×10^{-2}	11.22
100	2.63×10^{-1}	4.58×10^{-1}	1.6×10^{-2}	12.67	1.30×10^{-1}	67.68
200	4.93×10^{-1}	8.60×10^{-1}	3.67×10^{-2}	100.55	2.83×10^{-1}	511.95

APPENDIX B: CYBER 200 FORTRAN INTRINSIC FUNCTIONS

Listed below are the CYBER 200 FORTRAN intrinsic functions used in this report. These description are taken directly from reference [9]

Q8SCNT

Q8SCNT(v) is a specific scalar function that returns the number of 1 bits in the argument. The argument must be a vector of type bit. The result is of type integer.

For example, if bit vector V1 consists of the elements 1 0 0 1 1, the result of Q8SCNT(V1) is 3.

Q8SDOT

Q8SDOT(v1,v2) is a generic scalar function that returns the dot product of the two arguments. The arguments must be vectors and can be of type integer, real, or half-precision. If the arguments have different lengths, the excess elements of the longer argument are ignored. The result is of the same data type as the arguments. The result is the sum of the products of corresponding elements of the vector arguments.

For example, if vector V1 consists of the elements 0 1 3, and vector V2 consists of the elements 2 2 2, the result of Q8SDOT(V1,V2) is (0*2)+(1*2)+(3*2), which is 8.

Q8VCMPRS

Q8VCMPRS(v,cv;u) is a generic vector function that creates a vector consisting of selected elements of the input argument v. Teh input argument v must be a vector and can of type integer, real, or half-precision. The input argument cv, which is used as a control vector, must be a vector of type bit. The output argument can be a vector of the same data type as the input argument, or an integer expression that specifies the length of the vector function result. The input arguments must have the same length. The length of the vector through which the function result is returned is determined by the number of 1 bit in cv.

The function result consists of all of the elements of the input argument v whose corresponding elements in the control vector cv contain a 1 bit.

For example, if input argument V1 is a vector that consists of elements 2 4 6 8, and input argument CV1 is a bit vector that consists of the elements 0 1 0 1, the function reference Q8VCMPRS(V1,CV1;U1) assigns the values 4 8 to the output argument U1.

Q8VPOLY

Q8VPOLY(v1,v2;u) is a generic vector function that computes a polynomial at several points. The input arguments must be two vectors or one scalar and one vector. If a scalar is used as an input argument, the scalar must be the first input argument. The input arguments can be of type real or half-precision. The output argument can be a vector of the same data type as the input arguments, or an integer expression that specifies the length of the vector function result. The input arguments can have different lengths. The length of the vector through which the function result is returned must be the same as the length of the input argument v1, or longer.

The input argument v2 contains the coefficients of the polynomial: the first element of input argument v2 is the coefficient of the highest order term of the polynomial, and the last element of input argument v2 is the coefficient of the lowest order term of the polynomial, which is the constant. The length of input argument v2 determines the order of the polynomial. The order is one less than the number of elements in input argument v2. The input argument v1 contains the points at which the polynomial is to be evaluaed. The value of the first element of input argument v1 is substituted for the variable in the polynomial, the polynomial is evaluated, and the result is placed in the first element of the function result. This is repeated for each element of input argument v1.

For example, if input argument v1 is a vector that consists of the elements 2.0 3.0 5.0, and input argument v2 is a vector that consists of the elements 4.0 2.0 1.0, the function reference Q8VPOLY(V1,V2;U1) assigns the values 21.0 43.0 111.0 to the output argument U1. These values were computed by substituting each element of input argument V1 for the variable in the polynomial defined by the input argument V2. The polynomial is:

$$4x^2 + 2x + 1$$

Q8VSCATR

Q8VSCATR(v,i;u) is a generic vector function that creates a vector consisting of selected elements of the input argument v. The input argument v must be a vector and can be of type integer, real, or half-precision. The input argument i must be a vector of type integer. The output argument can be a vector of the same data type as the input argument v, or an integer expression that specifies the length of the vector function result. The input argument i and the vector through which the function result is returned must have the same length.

Each element of the input argument v corresponds to an element in input argument i. The elements in input argument i indicate to which elements in the function rsult the elements in input argument v are assigned. For example, if an element of i contains a 1, the element of input argument v that corresponds to that element in i is assigned to the first element of the function result. An element of the function result can be assigned more than one value; the last value an element is assigned is the value that it retains.

For example, if input argument V1 is a vector that consists of the elements 2.0 4.0 6.0 8.0, input argument I1 is a vector that consists of the elements 1 4 4 2, and output argument U1 is a vector that consists of the elements 9.0 9.0 9.0 9.0, the function reference Q8VSCATR(V1,I1;U1) assigns the values 2.0 8.0 9.0 6.0 to the output argument U1. The fourth element of the output argument is assigned the value 4.0, but is then reassigned the value 6.0. The third element of the output vector is never assigned; therefor, it retains its previous value.

Q8VXPND

Q8VXPND(v,cv;u) is a generic vector function that creates a vector that consists of the elements of input argument v plus additional elements having the value 0 or 0.0. The input argument v must be a vector of type integer, real, or half-precision. The input argument cv, which is used as the control vector, must be a vector of type bit. The output argument can be a vector of the same data type as the input argument v, or an integer expression that specifies the length of the vector function result. The length of the vector through which the function result is returned must be the same as the length of the input argument cv.

Each element of the function result corresponding to an element in the control vector cv that contains a 0 bit is assigned the value 0. The elements of the function result corresponding to 1 bits in the control vector are assigned values from the input argument v. The leftmost values from input argument v

are used, and any excess values are ignored.

For example, if input argument V1 is a vector that consists of the elements 5.0 5.0 5.0, and input argument CV1 is a bit vector consists of the elements 1 0 0 1, the function reference Q8VXPND(V1,CV1;U1) assigns the values 5.0 0.0 0.0 5.0 to the output argument U1.

APPENDIX C: SIMULATION PROGRAMS IN VECTOR CODE

```
C=============================================================
C
C                    FILE NMAE MMNA
C       THIS PROGRAM IS USED TO DO THE LARGE CIRCUIT
C       ANALYSIS USING THE MMNA ALGORITM. THE NUMBER
C       OF SUBCIRCUITS IN THE CIRCUIT CAN BE FROM 1
C       TO 1000. THE MAIN PROGRAM AND ALL SUBROUTINES
C       ARE WRITTEN IN VECTOR VERSION.
C
C       VARIABLES:
C
C       NZ - A REAL ARRAY CONTAINING THE NONZERO
C            ELEMENTS IN MATRIX A FOR MAIN
C            CIRCUIT.
C
C       NZS - A  REAL ARRAY CONTAINING THE NONZERO
C             ELEMENTS IN MATRIX A FOR SUBCIRCUITS.
C
C       B - A BIT MASK ARRAY FOR MAIN CIRCUIT.
C
C       BS - A BIT MASK ARRAY FOR SUBCIRCUITS
C
C       BR - A REAL ARRAY CONTAINING THE RIGHT HAND
C            SIDE OF THE EQUATION AX=B FOR MAIN
C            CIRCUIT.
C
C       BRS - A REAL ARRAY CONTAINING THE RIGHT HAND
C           OF THE FIRST NONZERO ELEMENT IN EACH ROW OF
C           THE MATRIX A FOR MAIN CIRCUIT.
C
C       R - A NITEGER ARRAY FOR INDICATING THE POSITIONS
C           FO THE FIRST NONZERO ELEMENT IN EACH ROW OF
C           THE MATRIX A FOR MAIN CIRCUIT.
C
C       RS - A INTEGER ARRAY FOR INDICATING THE
C            POSITIONS OF THE FIRST NONZERO ELEMENT IN
C            EACH ROW OF THE MATRIX A FOR SUBCIRCUITS.
C
C       D - A INTEGER ARRAY CONTAINING THE POINTERS OF
C           NONLINEAR DEVICES IN MAIN CIRCUIT.
C
C       DS - A INTEGER ARRAY CONTAINING THE POINTERS OF
C            NONLINEAR DEVICES IN SUBCIRCUITS.
C
C       PO - A REAL ARRAY CONTAINING THE COEFFICIENTS OF
C            POLYNOMIAL FOR FUNCTIONS OF NONLINEAR DEVICES
C            IN WHOLE CIRCUIT.
C
C       X - A RAEL ARRAY CONTAINING THE VARIABLES IN MAIN
C           CIRCUIT.
C
C       XS - A REAL ARRAY CONTAINING THE VARIABLES IN THE
C            SUBCIRCUITS.
C
C       CI - A INTEGER ARRAY INDICATING THE COLUMN ORDER
C            EXCHANGE OF THE MATRIX A FOR MAIN CIRCUIT.
C
C       CIS - A INTEGER ARRAY INDICATING THE COLUMN ORDER
C             EXCHANGE OF THE MATRIX A FOR SUBCIRCUITS.
C
C       NZP - A BIT ARRAY FOR INDICATING THE POSITIONS OF
C             G IN THE NZ.
C
C       NZPS - A BIT ARRAY FOR INDICATIOG THE POSITIONS OF
C              G IN THE NZS.
C=============================================================
```

```
      PROGRAM MMNA(TAPE5=INPUT,TAPE6=OUTPUT)

      ROWWISE B(1100,1100),BS(10,10)

      REAL NZS(1000,50),NZ(3100),PO(20,5),XS(1000,10),BR(1100),
     1 BRS(1000,10),IO(1000),EJ(1000),EG(1000),IBR(1100),IBRS(1000,10),
     1 JTS(1000,4),INZ(3100),INZS(1000,50),X(1100),T1,T2,T3,T4,T5



      INTEGER I,J,K,M,MS,N,NS,NL,NLS,P,Y,BPG,BPP,
     1 D(10,4),DS(4,4),CI(1100),CIS(10),
     1 R(1100),RS(10),NZPC(20),NZPSC(8),
     1 NZQC(1000),BPC(10),BPSC(4),
     1 BQC(1000),BPV(1000)

      BIT B,BS,E(1000),NZP(3100),NZPS(50),NZQ(3100),
     1 BQ(1100),BP(1100),BPS(10)


C     INPUT THE DATA
C     --------------

      CALL INP (IBR,IBRS,X,XS,INZ,INZS,B,BS,R,RS,D,DS,CI,CIS,M,MS,
     1 NZP,NZPC,NZPS,NZPSC,NZQ,NZQC,E,BP,BPC,Y,
     1 BPS,BPSC,BQ,BQC,NL,NLS,N,NS,PO,BPG,BPP,BPV)


C     SET THE NUMBER OF ITERATION
C     ---------------------------


      READ(5,1) K
1     FORMAT(10I4)

      TT1=0.0


      DO 10 L=1,K

C     COPY THE NZ AND NZS
C     -------------------

      NZ(1;M)=IN(1;M)

      DO 3 I=1,MS
      NZS(1,I;Y)=INS(1,I;Y)
3     CONTINUE


C     UPDATA THE  SUBNETWORK
C     ----------------------

      IF (NLS.NE.0) THEN
      TT3=SECOND()
      CALL UPNS (NZS,BRS,NZPS,NZPSC,PO,DS,XS,NS,NLS,CIS,MS,JTS,Y)
      TT4=SECOND()
      T1=TT4-TT3
      ENDIF



C     LU DECOMPOSITION FOR SUBNETWORK
C     -------------------------------

      TT3=SECOND()
      CALL LUS (NZS,BS,RS,NS,Y)
```

```
C       CALCULATE THE EG AND EJ
C       -----------------------

        DO 20 I=1,NS
        BRS(1,I;Y)=0.0
20      CONTINUE

        DO 30 I=1,Y
        BRS(I,BPP)=X(BPV(I))
30      CONTINUE

C       SOLVE THE SUBNETWORK
C       --------------------

        CALL SXY (NZS,BRS,BS,RS,NS,Y,XS)


C       OBTAIN THE I WITH ZERO INPUT
C       ----------------------------

        IO(1;Y)=XS(1,BPG;Y)

C       CALCULATE THE BRS
C       -----------------

        P=0
        DO 40 I=1,NS
        IF (BTOL(BPS(I))) THEN

        P=P+1
        DO 50 J=1,Y
        BRS(J,I)=JTS(J,BPSC(P))
50      CONTINUE

        ELSE
        DO 60 J=1,Y
        BRS(J,I)=IBRS(J,I)
60      CONTINUE

        ENDIF
40      CONTINUE

C       SET THE SUBNETWORK INPUT
C       ------------------------

        DO 70 I=1,Y
        BRS(I,BPP)=X(BPV(I))
70      CONTINUE

C       CALCULATE THE SUBNETWORK
C       ------------------------

        CALL SXY (NZS,BRS,BS,RS,NS,Y,XS)


C       OBTAIN THE EG AND EJ
C       --------------------

        DO 90 I=1,Y
        EG(I)=-IO(I)/X(BPV(I))
        EJ(I)=XS(I,BPG)-IO(I)
90      CONTINUE
```

```
C       COPY THE BR
C       -----------

        BR(1;N)=IBR(1;N)

        TT4=SECOND()
        T2=TT4-TT3

C       UPDATA THE NONLINEAR DEVICES OF THE MAIN NETWORK
C       ------------------------------------------------

        TT3=SECOND()
        IF (NL.NE.0) THEN
        CALL UPNM (NZ,BR,NZP,NZPC,BPC,BP,PO,D,X,N,NL,CI,M)
        ENDIF

C       UPDATA THE INPUT OF THE MAIN NETWORK
C       ------------------------------------

        CALL UPI (NZQ,Y,N,M,NZQC,EG,EJ,E,BQ,BQC,NZ,BR)

        TT4=SECOND()
        T3=TT4-TT3

C       LU DECOMPOSITION AND SOLVE THE MAIN NETWORK
C       -------------------------------------------

        TT3=SECOND()
        CALL LUM (NZ,X,BR,B,R,N)
        TT4=SECOND()
        T4=TT4-TT3

C       SUBSTITUTE TO THE SUBNETWORK
C       ----------------------------

        TT3=SECOND()
        DO 100 I=1,Y
        BRS(I,BPP)=X(BPV(I))
100     CONTINUE

        CALL SXY (NZS,BRS,BS,RS,NS,Y,XS)
        TT4=SECOND()
        T5=TT4-TT3

        TT1=TT1+T1+T2+T3+T4+T5

C       PRINT THE RESULTS OF THIS ITERATION
C       -----------------------------------


        WRITE(6,6) L

        WRITE(6,7) (X(I+Y-1),I=1,N-Y+1)

        WRITE(6,8) (XS(1,I),I=1,NS)

        WRITE(6,9) T1,T2,T3,T4,T5


6       FORMAT (1X,≠***≠//35X,≠THE NUMBER OF ITERATION  ≠,I5//
     1  25X,≠**************************************************≠/)

7       FORMAT (/30X,≠THE VALUE OF VARIABLE IN THE MAIN NETWORK  ≠//
     1  10X,5F15.8//)

8       FORMAT (/30X,≠THE VALUE OF VARIABLE IN FIRST OF SUBNETWORKS ≠//
     1  20X,4F15.8//)
```

```
9       FORMAT (////30X,≠THE VALUE OF THE CPUTIME FOR THIS ITERAYION≠//
     1  10X,5F15.8/////)

10      CONTINUE

        T1=TT1

C       OUTPUT THE FINAL RESULTS
C       ------------------------

        CALL OUT (N,NS,Y,X,XS,T1)

        STOP
        END
C-------------------------------------------------------------
C
C                   FILE NAME UPNS
C
C       THIS SUBROUTINE IS USED TO DO UPDATA THE
C       SUBNETWORKS. ALL OF THE FUNCTIONS OF THE
C       NONLINEAR DEVICE IN SUBCIRCUITS ARE WRITTEN
C       AS THE POLYNOMIAL OF THE VOLTAGE V.
C
C-------------------------------------------------------------

        SUBROUTINE UPNS (NZS,BRS,NZPS,NZPSC,PO,DS,XS,NS,NLS,CIS,
     1  MS,JTS,Y)

        REAL NZS(1000,50),NZSO(1000,8),XS(1000,10),BRS(1000,10),
     1  GS(1000,8),TIS(1000),TXS(1000,10),VS(1000),GTS(1000),
     1  PO(20,5),JTS(1000,4)

        INTEGER DS(4,4),Y,CIS(10),NS,NZPSC(8),
     1  NLS,MS,P,I,J,Z

        BIT NZPS(50)

C       FIND THE V ACROSS THE NONLINEAR DEVICES
C       ---------------------------------------

        DO 10 I=1,NS
        TXS(1,CIS(I);Y)=XS(1,I;Y)
10      CONTINUE

C       CALCULATE THE GS AND JS
C       -----------------------

        P=0
        DO 20 I=1,NLS
        IF(DS(I,4).EQ.0) THEN
        VS(1;Y)=TXS(1,DS(I,1);Y)
        ELSE
        VS(1;Y)=TXS(1,DS(I,1);Y)-TXS(1,DS(I,2);Y)
        ENDIF

C       CALCULATE THE POLYNOMIAL
C       ------------------------

        GTS(1;Y)=0.0
```

```
        DO 30 J=1,10
        GTS(1;Y)=GTS(1;Y)*VS(1;Y)+PO(J,DS(I,3))
30      CONTINUE

C       OBTAIN THE JTS
C       --------------

        TIS(1;Y)=0.0
        DO 35 J=11,20
        TIS(1;Y)=TIS(1;Y)*VS(1;Y)+PO(J,DS(I,3))
35      CONTINUE

        JTS(1,I;Y)=GTS(1;Y)*VS(1;Y)-TIS(1;Y)

C       OBTAIN THE GS
C       -------------

        P=P+1
        GS(1,P;Y)=GTS(1;Y)
        IF(DS(I,4).EQ.0) GO TO 20
        P=P+1
        GS(1,P;Y)=-GTS(1;Y)
20      CONTINUE

C       OBTAIN THE NZSO
C       ---------------

        DO 40 I=1,P
        NZSO(1,I;Y)=GS(1,NZPSC(I);Y)
40      CONTINUE

C       UPDATA THE NZS
C       --------------

        P=0
        DO 50 I=1,MS
        IF (BTOL(NZPS(I))) THEN
        P=P+1
        NZS(1,I;Y)=NZSO(1,P;Y)
        ENDIF
50      CONTINUE

        RETURN
        END
C-------------------------------------------------------------
C
C                   FILE NAME LUS
C
C       THIS SUBROUTINE IS USED TO DO THE LU
C       DECOMPOSITION FOR THE MATEIX A OF THE
C       SUBCIRCUITS.
C
C-------------------------------------------------------------

        SUBROUTINE LUS(NZS,BS,RS,NS,Y)

        ROWWISE BS(10,10)

        REAL NZS(1000,50),SUM(1000)

        INTEGER RS(10),NS,Q,QQ,QL,QU,I,K,P,W,Z,J,Y

        BIT BS
```

```
C       CALCULATE THE FIRST ROW OF U
C       ----------------------------


        Q=1
        DO 2 W=2,NS
        IF (BTOL(BS(1,W))) THEN
        Q=Q+1
        NZS(1,Q;Y)=NZS(1,Q;Y)/NZS(1,1;Y)
        ENDIF
2       CONTINUE


        DO 10 K=2,NS

C       CALCULATE THE KTH COLUMN OF L (TO 20)
C       -------------------------------------

        DO 20 I=K,NS
        IF (BTOL(BS(I,K))) THEN

C       INITIALIZATION
C       --------------

        SUM(1;Y)=0.0

        QL=RS(I)-1
        DO 40 P=1,K-1

C       FIND THE INDEX OF L AND U
C       -------------------------

        IF (BTOL(BS(I,P))) THEN
        QL=QL+1
        IF (BTOL(BS(P,K))) THEN
        QU=RS(P)+Q8SCNT(BS(P,1;K))-1

C       CALCULATE THE SUM
C       -----------------

        SUM(1;Y)=NZS(1,QL;Y)*NZS(1,QU;Y)+SUM(1;Y)

        ENDIF
        ENDIF
40      CONTINUE


C       OBTAIN THE L
C       ------------

        QL=QL+1
        NZS(1,QL;Y)=NZS(1,QL;Y)-SUM(1;Y)
        ENDIF
20      CONTINUE

C       CALCULATE THE KTH ROW OF U
C       --------------------------

        IF (K.NE.NS) THEN

        DO 120 J=K+1,NS
        IF (BTOL(BS(K,J))) THEN

C       INITIALIZATION
C       --------------

        SUM(1;Y)=0.0
```

```
      QL=RS(K)-1
      QQ=QL
      DO 140 P=1,K-1

C     FIND THE INDEX OF U AND L
C     -------------------------

      IF (BTOL(BS(K,P))) THEN
      QL=QL+1
      IF (BTOL(BS(P,J))) THEN
      QU=RS(P)+Q8SCNT(BS(P,1:J))-1

C     FIND THE SUM
C     ------------

      SUM(1:Y)=NZS(1,QL:Y)*NZS(1,QU:Y)+SUM(1:Y)

      ENDIF
      ENDIF
140   CONTINUE

C     OBTAIN THE U
C     ------------

      QQ=Q8SCNT(BS(K,1:J))+QQ
      QL=QL+1
      NZS(1,QQ:Y)=(NZS(1,QQ:Y)-SUM(1:Y))/NZS(1,QL:Y)

      ENDIF
120   CONTINUE
      ENDIF

10    CONTINUE
      RETURN
      END
C-------------------------------------------------------------
C
C                  FILE NAME SXY
C
C     THIS SUBROUTINE IS USED TO SOLVE THE VECTORS
C     Y AND X FOR THE SUBCIRCUITS.
C
C-------------------------------------------------------------

      SUBROUTINE SXY (NZS,BRS,BS,RS,NS,Y,XS)


      ROWWISE BS(10,10)

      REAL NZS(1000,50),XS(1000,10),YS(1000,10),
     1 BRS(1000,10),SUM(1000)

      INTEGER RS(10),NS,L,QQ,I,K,P,
     1 Z,J,DD,Y

      BIT BS


C     FIND THE FIRST ELEMENT OF Y
C     ---------------------------

      YS(1,1:Y)=BRS(1,1:Y)/NZS(1,1:Y)

C     CALCULATE THE Y (TO 20)
C     -----------------------
```

```
      DO 20 K=2,NS

C     INITIAL SUM
C     -----------

      SUM(1:Y)=0.0

C     CALCULATE THE SUM
C     -----------------

      DD=RS(K)-1
      DO 40 J=1,K-1
      IF (BTOL(BS(K,J))) THEN
      DD=DD+1

      SUM(1:Y)=NZS(1,DD:Y)*YS(1,J:Y)+SUM(1:Y)

      ENDIF
40    CONTINUE

C     OBTAIN THE Y
C     ------------

      DD=DD+1
      YS(1,K:Y)=(BRS(1,K:Y)-SUM(1:Y))/NZS(1,DD:Y)

20    CONTINUE

C     CALCULATE THE LAST ELEMENT OF X
C     -------------------------------

      XS(1,NS:Y)=YS(1,NS:Y)

C     CALCULATE THE X (TO 70)
C     -----------------------

      DO 70 K=1,NS-1

C     FIHD THE INDEX OF L
C     -------------------

      L=NS-K

      QQ=RS(L)+Q8SCNT(BS(L,1:L))-1

C     INITIAL THE SUM
C     ---------------

      SUM(1:Y)=0.0

C     CALCULATE THE SUM
C     -----------------

      DO 100 P=L+1,NS
      IF (BTOL(BS(L,P))) THEN
      QQ=QQ+1

      SUM(1:Y)=NZS(1,QQ:Y)*XS(1,P:Y)+SUM(1:Y)

      ENDIF
100   CONTINUE

C     OBTAIN THE X
C     ------------

      XS(1,L:Y)=YS(1,L:Y)-SUM(1:Y)
```

```
70      CONTINUE

        RETURN
        END
C-------------------------------------------------------------
C
C                   FILE NAME UPNM
C
C       THIS SUBROUTINE IS USED TO DO THE UPDATA THE
C       NONLINEAR SEVICES OF THE MAIN NETWORK. ALL OF
C       THE FUNCTIONS OF NONLINEAR DEVICES IN MAIN
C       CIRCUIT ARE WRITTEN AS THE POLYNOMIAL OF
C       VOLTAGE V.
C
C
C-------------------------------------------------------------


        SUBROUTINE UPNM (NZ,BR,NZP,NZPC,BPC,BP,PO,D,X,N,NL,CI,M)

        REAL NZ(3100),X(1100),BR(1100),PO(20,5),G(20),
     1  TX(1100),TI,V,JT(10),GT,NZO(3100),BRO(1100)


        INTEGER NZPC(20),D(10,4),BPC(10),CI(1100),N,
     1  NL,P,I,J,M

        BIT NZP(3100),BP(1100)


C       FIND THE OLD ORDER OF X
C       -----------------------


        TX(1;N)=Q8VSCATR(X(1;N),CI(1;N);TX(1;N))

C       CALCULATE THE G AND JT (TO 20)
C       ------------------------------

        P=0
        DO 20 I=1,NL

C       FIND THE V ACROSS THE NONLINEAR DEVICES
C       ---------------------------------------

        IF (D(I,4).EQ.0) THEN
        V=TX(D(I,1))
        ELSE
        V=TX(D(I,1))-TX(D(I,2))
        ENDIF

C       CALCULATE THE POLYNOME
C       ----------------------

        GT=0.0
        DO 30 J=1,10
        GT=GT*V+PO(J,D(I,3))
30      CONTINUE

        TI=0.0
        DO 40 J=11,20
        TI=TI*V+PO(J,D(I,3))
40      CONTINUE


C       OBTAIN THE JT
C       -------------
```

```
      JT(I)=GT*V-TI

C     OBTAIN THE G
C     ------------

      P=P+1
      G(P)=GT
      IF (D(I,4).EQ.0) GO TO 20
      P=P+1
      G(P)=-GT
20    CONTINUE

C     UPDATA THE NZ
C     -------------

      NZO(1:P)=Q8VSCATR(G(1:P),NZPC(1:P):NZO(1:P))
      G(1:P)=NZO(1:P)

      NZ(1:M)=NZ(1:M)+Q8VXPND(G(1:P),NZP(1:M):NZO(1:M))

C     UPDATA THE BR
C     -------------

      BRO(1:NL)=Q8VSCATR(JT(1:NL),BPC(1:NL):BRO(1:NL))
      JT(1:NL)=BRO(1:NL)

      BR(1:N)=BR(1:N)+Q8VXPND(JT(1:NL),BP(1:N):BRO(1:N))

      RETURN
      END
C-------------------------------------------------------------
C
C                  FILE NAME UPI
C
C     THIS SUBROUTINE IS USED TO UPDATA THE INPUT
C     OF THE MAIN NETWORK. ASSUME ALL OF THE
C     SUBCIRCUITS HERE HAVE ONLY TWO PORTS.
C
C-------------------------------------------------------------

      SUBROUTINE UPI (NZQ,Y,N,M,NZQC,EG,EJ,E,BQ,BQC,NZ,BR)

      REAL NZ(3100),BR(1100),EG(1000),
     1 EJ(1000),NZO(3100),EO(2000),BRO(1100)

      INTEGER P,I,Y,N,M,NZQC(2000),BQC(1000)

      BIT NZQ(3100),BQ(1100),E(1000)

C     CALCULATE THE EO
C     ----------------

      P=0
      DO 10 I=1,Y
      P=P+1
      EO(P)=EG(I)
      IF (.NOT.(BTOL(E(I)))) GO TO 10
      P=P+1
      EO(P)=-EG(I)
10    CONTINUE

C     UPDATA THE NZ
```

```
C       -------------

        NZO(1;P)=Q8VSCATR(EO(1;P),NZQC(1;P);NZO(1;P))
        EO(1;P)=NZO(1;P)

        NZ(1;M)=NZ(1;M)+Q8VXPND(EO(1;P),NZQ(1;M);NZO(1;M))

C       UPDATA THE BR
C       -------------

        BRO(1;Y)=Q8VSCATR(EJ(1;Y),BQC(1;Y);BRO(1;Y))
        EJ(1;Y)=BRO(1;Y)

        BR(1;N)=BR(1;N)+Q8VXPND(EJ(1;Y),BQ(1;N);BRO(1;N))

        RETURN
        END
C-------------------------------------------------------------
C
C                   FILE NAME LUM
C
C       THIS SUBROUTINE IS USED TO DO THE LU
C       DECOMPOSITION FOR MATRIX A FOR MAIN
C       CIRCUIT AND SOLVE THE VECTORS Y AND X.
C
C-------------------------------------------------------------

        SUBROUTINE LUM (NZ,X,BR,B,R,N)

        ROWWISE B(1100,1100)

        REAL NZ(3100),X(1100),Y(1100),BR(1100),SUM(1100),
     1  TU(1100),TL(1100),NZT(1100)

        INTEGER R(1100),N,I,J,K,Q,QL,QQ,QU,P,DD,W

        BIT B


C       FIND THE FIRST ELEMENT OF Y
C       ---------------------------

        Y(1)=BR(1)/NZ(1)

C       CALCULATE THE FIRST ROW OF U
C       ----------------------------

        Q=Q8SCNT(B(1,2;N-1))
        NZ(2;Q)=NZ(2;Q)/NZ(1)

C       DO LU DECOMPOSITION AND CALCULATE Y (TO 20)
C       -------------------------------------------

        DO 20 K=2,N

C       GENERATE THE VECTOR TU
C       ----------------------

        TU(1;K-1)=0.0
        DO 30 I=1,K-1
        IF (BTOL(B(I,K))) THEN
        QU=R(I)+Q8SCNT(B(I,1;K))-1
        TU(I)=NZ(QU)
        ENDIF
30      CONTINUE

C       CALCULATE THE KTH COLUMN OF L (TO 40)
C       -------------------------------------
```

```
      DO 40 I=K,N
      IF (BTOL(B(I,K))) THEN
      QL=R(I)

C     CALCULATE THE SUM
C     -----------------

      TL(1:K-1)=Q8UXPND(NZ(QL:K-1),B(I,1:K-1):TL(1:K-1))
      SUM(1)=Q8SDOT(TL(1:K-1),TU(1:K-1))

C     OBTAIN THE L
C     ------------

      QL=Q8SCNT(B(I,1:K-1))+QL
      NZ(QL)=NZ(QL)-SUM(1)
      ENDIF
40    CONTINUE

C     CALCULATE THE Y
C     ---------------

      QL=R(K)

C     GENERATE THE VECTOR TL
C     ----------------------

      TL(1:K-1)=Q8UXPND(NZ(QL:K-1),B(K,1:K-1):TL(1:K-1))

C     CALCULATE THE SUM
C     -----------------

      SUM(1)=Q8SDOT(TL(1:K-1),Y(1:K-1))
      DD=Q8SCNT(B(K,1:K-1))+QL

C     OBTAIN THE Y
C     ------------

      Y(K)=(BR(K)-SUM(1))/NZ(DD)

C     CALCULATE THE KTH ROW OF U
C     --------------------------

      IF (K.NE.N) THEN
      QL=R(K)-1
      DD=N-K
      SUM(1:DD)=0.0
      DO 80 J=1,K-1
      IF (BTOL(B(K,J))) THEN
      QL=QL+1
      QU=R(J)+Q8SCNT(B(J,1:K))

C     CALCULATE THE SUM
C     -----------------

      TU(1:DD)=Q8UXPND(NZ(QU:DD),B(J,K+1:DD):TU(1:DD))
      WHERE (B(K,K+1:DD)) SUM(1:DD)=SUM(1:DD)+TU(1:DD)*NZ(QL)
      ENDIF
80    CONTINUE

C     OBTAIN THE U
C     ------------

      NQU=Q8SCNT(B(K,K+1:DD))
      QU=QL+1
      NZT(1:NQU)=Q8UCMPRS(SUM(1:DD),B(K,K+1:DD):NZT(1:NQU))
      NZ(QU+1:NQU)=(NZ(QU+1:NQU)-NZT(1:NQU))/NZ(QU)
```

```
      ENDIF
20    CONTINUE

C     CALCULATE THE X
C     ---------------

      X(N)=Y(N)

      DO 120 K=1,N-1

C     CALCULATE THE SUM
C     -----------------

      L=N-K

      QQ=R(L)+Q8SCNT(B(L,1;L))
      TU(1;K)=Q8VXPND(NZ(QQ;K),B(L,L+1;K);TU(1;K))
      SUM(1)=Q8SDOT(TU(1;K),X(L+1;K))

C     OBTAIN THE X
C     ------------

      X(L)=Y(L)-SUM(1)

120   CONTINUE

      RETURN
      END
C-------------------------------------------------------------
C
C                FILE NAME OUT
C
C     THIS SUBROUTINE IS USED TO OUTPUT THE
C     FINAL RESULTS OF MMNA PROGRAM.
C
C-------------------------------------------------------------

      SUBROUTINE OUT (N,NS,Y,X,XS,T1)

      REAL T1,X(1100),XS(1000,10)

      INTEGER N,NS,I,Y

      WRITE (6,4) Y

      WRITE(6,1) (X(I+Y-1),I=1,N-Y+1)

      WRITE(6,2) (XS(1,J),J=1,NS)

      WRITE(6,3) T1

1     FORMAT (/25X,≠THE FINALE RESULTS OF THE MAIN NETWORK≠//
     1  10X,5F15.8/)

2     FORMAT (25X,≠THE FINALE RESULTS IN THE FIRST OF SUBNETWORKS≠//
     1  20X,4F15.8/)

3     FORMAT (////35X,≠THE TOTAL CPU TIME≠//35X,≠T =≠,F15.10/)

4     FORMAT (////30X,≠THE NUMBER OF SUBNETWORKS IS≠,I8//25X,
     1  ≠**************************************************≠///)

      RETURN
      END
```

APPENDIX D: SIMULATION PROGRAMS IN SCALAR CODE

```
C=============================================================
C
C                    FILE NMAE MMNA
C          THIS PROGRAM IS USED TO DO THE LARGE CIRCUIT
C          ANALYSIS USING THE MMNA ALGORITM. THE NUMBER
C          OF SUBCIRCUITS IN THE CIRCUIT CAN BE FROM 1
C          TO 1000. THE MAIN PROGRAM AND ALL SUBROUTINES
C          ARE WRITTEN IN SCALAR VERSION.
C
C          VARIABLES:
C
C          NZ - A REAL ARRAY CONTAINING THE NONZERO
C               ELEMENTS IN MATRIX A FOR MAIN
C               CIRCUIT.
C
C          NZS - A  REAL ARRAY CONTAINING THE NONZERO
C                ELEMENTS IN MATRIX A FOR SUBCIRCUITS.
C
C          B - A BIT MASK ARRAY FOR MAIN CIRCUIT.
C
C          BS - A BIT MASK ARRAY FOR SUBCIRCUITS
C
C          BR - A REAL ARRAY CONTAINING THE RIGHT HAND
C               SIDE OF THE EQUATION AX=B FOR MAIN
C               CIRCUIT.
C
C          BRS - A REAL ARRAY CONTAINING THE RIGHT HAND
C              OF THE FIRST NONZERO ELEMENT IN EACH ROW OF
C              THE MATRIX A FOR MAIN CIRCUIT.
C
C          R - A NITEGER ARRAY FOR INDICATING THE POSITIONS
C              FO THE FIRST NONZERO ELEMENT IN EACH ROW OF
C              THE MATRIX A FOR MAIN CIRCUIT.
C
C          RS - A INTEGER ARRAY FOR INDICATING THE
C               POSITIONS OF THE FIRST NONZERO ELEMENT IN
C               EACH ROW OF THE MATRIX A FOR SUBCIRCUITS.
C
C          D - A INTEGER ARRAY CONTAINING THE POINTERS OF
C              NONLINEAR DEVICES IN MAIN CIRCUIT.
C
C          DS - A INTEGER ARRAY CONTAINING THE POINTERS OF
C               NONLINEAR DEVICES IN SUBCIRCUITS.
C
C          PO - A REAL ARRAY CONTAINING THE COEFFICIENTS OF
C               POLYNOMIAL FOR FUNCTIONS OF NONLINEAR DEVICES
C               IN WHOLE CIRCUIT.
C
C          X - A RAEL ARRAY CONTAINING THE VARIABLES IN MAIN
C              CIRCUIT.
C
C          XS - A REAL ARRAY CONTAINING THE VARIABLES IN THE
C               SUBCIRCUITS.
C
C          CI - A INTEGER ARRAY INDICATING THE COLUMN ORDER
C               EXCHANGE OF THE MATRIX A FOR MAIN CIRCUIT.
C
C          CIS - A INTEGER ARRAY INDICATING THE COLUMN ORDER
C                EXCHANGE OF THE MATRIX A FOR SUBCIRCUITS.
C
C          NZP - A BIT ARRAY FOR INDICATING THE POSITIONS OF
C                G IN THE NZ.
C
C          NZPS - A BIT ARRAY FOR INDICATIOG THE POSITIONS OF
C                 G IN THE NZS.
C=============================================================
```

```
      PROGRAM MMNA(TAPE5=INPUT,TAPE6=OUTPUT)

      ROWWISE B(1100,1100),BS(10,10)

      REAL NZS(1000,50),NZ(3100),PO(20,5),XS(1000,10),BR(1100),
     1 BRS(1000,10),IO(1000),EJ(1000),EG(1000),IBR(1100),IBRS(1000,10),
     1 JTS(1000,4),INZ(3100),INZS(1000,50),X(1100),T1,T2,T3,T4,T5

      INTEGER I,J,K,M,MS,N,NS,NL,NLS,P,Y,BPG,BPP,
     1 D(10,4),DS(4,4),CI(1100),CIS(10),
     1 R(1100),RS(10),NZPC(20),NZPSC(8),
     1 NZQC(2000),BPC(10),BPSC(4),
     1 BQC(1000),BPV(1000)

      BIT B,BS,E(1000),NZP(3100),NZPS(50),NZQ(3100),BP(1100),BPS(10),
     1 BQ(1100)

C     INPUT THE DATA
C     --------------

      CALL INP (IBR,IBRS,X,XS,INZ,INZS,B,BS,R,RS,D,DS,CI,CIS,M,MS,
     1 NZP,NZPC,NZPS,NZPSC,NZQ,NZQC,E,BP,BPC,Y,
     1 BPS,BPSC,BQ,BQC,NL,NLS,N,NS,PO,BPG,BPP,BPV)

C     SET THE NUMBER OF ITERATION
C     ---------------------------

      READ(5,1) K
1     FORMAT(10I4)

      TT1=0.0

      DO 10 L=1,K

C     COPY THE NZ AND NZS
C     -------------------

      DO 4 I=1,M
      NZ(I)=INZ(I)
4     CONTINUE

      DO 3 I=1,MS
      DO 3 J=1,Y
      NZS(J,I)=INZS(J,I)
3     CONTINUE

C     UPDATA THE NONLINEAR DEVICES OF THE SUBNETWORK
C     ----------------------------------------------

      IF (NLS.NE.0) THEN
      TT3=SECOND()
      CALL UPNS (NZS,BRS,NZPS,NZPSC,PO,DS,XS,NS,NLS,CIS,MS,JTS,Y)
      TT4=SECOND()
      T1=TT4-TT3
      ENDIF

C     LU DECOMPOSITION FOR SUBNETWORK
```

```
C       ------------------------------

        TT3=SECOND()
        CALL LUS (NZS,BS,RS,NS,Y)

C       CALCULATE THE EG AND EJ
C       -----------------------

        DO 20 I=1,NS
        DO 20 J=1,Y
        BRS(J,I)=0.0
20      CONTINUE

        DO 30 I=1,Y
        BRS(I,BPP)=X(BPV(I))
30      CONTINUE

C       SOLVE THE SUBNETWORK
C       --------------------

        CALL SXY (NZS,BRS,BS,RS,NS,Y,XS)


C       OBTAIN THE I WITH ZERO INPUT
C       ----------------------------

        DO 35 I=1,Y
        IO(I)=XS(I,BPG)
35      CONTINUE

C       CALCULATE THE BRS
C       -----------------

        P=0
        DO 40 I=1,NS
        IF (BTOL(BPS(I))) THEN

        P=P+1
        DO 50 J=1,Y
        BRS(J,I)=JTS(J,BPSC(P))
50      CONTINUE

        ELSE
        DO 60 J=1,Y
        BRS(J,I)=IBRS(J,I)
60      CONTINUE

        ENDIF
40      CONTINUE

C       SET THE SUBNETWORK INPUT
C       ------------------------

        DO 70 I=1,Y
        BRS(I,BPP)=X(BPV(I))
70      CONTINUE

C       CALCULATE THE SUBNETWORK
C       ------------------------

        CALL SXY (NZS,BRS,BS,RS,NS,Y,XS)


C       OBTAIN THE EG AND EJ
```

```
C       --------------------

        DO 90 I=1,Y
        EG(I)=-IO(I)/X(BPV(I))
        EJ(I)=XS(I,BPG)-IO(I)
90      CONTINUE

C       COPY THE BR
C       -----------

        DO 95 I=1,N
        BR(I)=IBR(I)
95      CONTINUE

        TT4=SECOND()
        T2=TT4-TT3

C       UPDATA THE NONLINEAR DEVICES OF THE MAIN NETWORK
C       ------------------------------------------------

        TT3=SECOND()
        IF (NL.NE.0) THEN
        CALL UPNM (NZ,BR,NZP,NZPC,BPC,BP,PO,D,X,N,NL,CI,M)
        ENDIF

C       UPDATA THE INPUT OF THE MAIN NETWORK
C       ------------------------------------

        CALL UPI (NZQ,Y,N,M,NZQC,EG,EJ,E,BQ,BQC,NZ,BR)

        TT4=SECOND()
        T3=TT4-TT3

C       LU DECOMPOSITION AND SOLVE THE MAIN NETWORK
C       -------------------------------------------

        TT3=SECOND()
        CALL LUM (NZ,X,BR,B,R,N)
        TT4=SECOND()
        T4=TT4-TT3

C       SUBSTITUTE TO THE SUBNETWORK
C       ----------------------------

        TT3=SECOND()
        DO 100 I=1,Y
        BRS(I,BPP)=X(BPV(I))
100     CONTINUE

        CALL SXY (NZS,BRS,BS,RS,NS,Y,XS)
        TT4=SECOND()
        T5=TT4-TT3

        TT1=TT1+T1+T2+T3+T4+T5

C       PRINT THE RESULTS OF THIS ITERATION
C       -----------------------------------


        WRITE(6,6) L

        WRITE(6,7) (X(I+Y-1),I=1,N-Y+1)

        WRITE(6,8) (XS(1,I),I=1,NS)

        WRITE(6,9) T1,T2,T3,T4,T5
```

```
6       FORMAT (1X,≠***≠//35X,≠THE NUMBER OF ITERATION  ≠,I5//
     1  25X,≠**************************************************≠/)

7       FORMAT (/30X,≠THE VALUE OF VARIABLE IN THE MAIN NETWORK  ≠//
     1  10X,5F15.8//)

8       FORMAT(/30X,≠THE VALUE OF VARIABLE IN FIRST OF SUBNETWORKS ≠//
     1  20X,4F15.8//)

9       FORMAT (////30X,≠THE VALUE OF THE CPUTIME FOR THIS ITERAYION≠//
     1  10X,5F15.8/////)


10      CONTINUE

        T1=TT1


C       OUTPUT THE FINAL RESULTS
C       ------------------------


        CALL OUT (N,NS,Y,X,XS,T1)

        STOP
        END
C-------------------------------------------------------
C
C                   FILE NAME UPNS
C
C       THIS SUBROUTINE IS USED TO DO THE UPDATA
C       SUBCIRCUITS. ALL OF THE FUNCTIONS OF THE
C       NONLINEAR DEVICES IN SUBCIRCUITS ARE WRITTEN
C       AS THE POLYNOMIAL OF THE VOLTAGE V.
C
C-------------------------------------------------------


        SUBROUTINE UPNS (NZS,BRS,NZPS,NZPSC,PO,DS,XS,NS,NLS,CIS,
     1  MS,JTS,Y)

        REAL NZS(1000,50),XS(1000,10),BRS(1000,10),PO(20,5),
     1  NZSO(8),GS(8),TXS(1000,10),VS,GTS,JTS(1000,4)

        INTEGER NZPSC(8),DS(4,4),CIS(10),NS,
     1  NLS,MS,P,I,J,Z,Y

        BIT NZPS(50)

C       FIND THE OLD ORDER OF XS
C       ------------------------


        DO 10 I=1,NS
        DO 10 Z=1,Y
        TXS(Z,CIS(I))=XS(Z,I)
10      CONTINUE


C       CALCULATE THE GS AND JS
C       -----------------------


        DO 20 Z=1,Y
        P=0
        DO 25 I=1,NLS
        IF (DS(I,4).EQ.0) THEN
        VS=TXS(Z,DS(I,1))
        ELSE
```

```
        VS=TXS(Z,DS(I,1))-TXS(Z,DS(I,2))
        ENDIF

C       CALCULATE THE POLYNOME
C       ----------------------

        GTS=0.0
        DO 30 J=1,10
        GTS=GTS*VS+PO(J,DS(I,3))
30      CONTINUE

C       OBTAIN THE JTS
C       --------------


        TIS=0.0
        DO 35 J=11,20
        TIS=TIS*VS+PO(J,DS(I,3))
35      CONTINUE

        JTS(Z,I)=GTS*VS-TIS

C       OBTAIN THE GS
C       -------------

        P=P+1
        GS(P)=GTS
        IF (DS(I,4).EQ.0) GO TO 25
        P=P+1
        GS(P)=-GTS
25      CONTINUE

C       OBTAIN THE NZSO
C       ---------------

        DO 40 I=1,P
        NZSO(I)=GS(NZPSC(I))
40      CONTINUE

C       UPDATA THE NZS
C       --------------

        P=0
        DO 20 I=1,MS
        IF (BTOL(NZPS(I))) THEN
        P=P+1
        NZS(Z,I)=NZSO(P)
        ENDIF

20      CONTINUE

        RETURN
        END
C-------------------------------------------------------
C
C                   FILE NAME LUS
C
C       THIS SUBROUTINE IS USED TO DO THE LU
C       DECOMPOSITION FOR THE MATRIX A OF THE
C       SUBCIRCUITS.
C
C-------------------------------------------------------


        SUBROUTINE LUS(NZS,BS,RS,NS,Y)

        ROWWISE BS(10,10)
```

```
      REAL NZS(1000,50),SUM(1000)

      INTEGER RS(10),NS,Q,QQ,QL,QU,I,K,P,W,Z,J,Y
      BIT BS

C     CALCULATE THE FIRST ROW OF U
C     ----------------------------

      Q=1
      DO 2 W=2,NS
      IF (BTOL(BS(1,W))) THEN
      Q=Q+1
      DO 4 Z=1,Y
      NZS(Z,Q)=NZS(Z,Q)/NZS(Z,1)
4     CONTINUE
      ENDIF
2     CONTINUE

C     DO LU DECOMPOSITION  (TO 10)
C     ----------------------------------------


      DO 10 K=2,NS

C     CALCULATE THE KTH COLUMN OF L (TO 20)
C     ---------------------------------

      DO 20 I=K,NS
      IF (BTOL(BS(I,K))) THEN

C     INITIAL SUM
C     -----------

      DO 30 Z=1,Y
      SUM(Z)=0.0
30    CONTINUE

      QL=RS(I)-1
      DO 40 P=1,K-1

C     FIND THE INDEX OF L AND U
C     -------------------------

      IF (BTOL(BS(I,P))) THEN
      QL=QL+1
      IF (BTOL(BS(P,K))) THEN
      QU=RS(P)-1
      DO 50 KK=1,K
      IF (BTOL(BS(P,KK))) QU=QU+1
50    CONTINUE

C     CALCULATE SUM
C     -------------

      DO 60 Z=1,Y
      SUM(Z)=NZS(Z,QL)*NZS(Z,QU)+SUM(Z)
60    CONTINUE

      ENDIF
      ENDIF
40    CONTINUE

C     OBTAIN THE L
C     ------------

      QL=QL+1
      DO 70 Z=1,Y
```

```
      NZS(Z,QL)=NZS(Z,QL)-SUM(Z)
70    CONTINUE

      ENDIF
20    CONTINUE


C     CALCULATE THE KTH ROW OF THE U
C     ------------------------------

      IF (K.NE.NS) THEN
      DO 120 J=K+1,NS
      IF (BTOL(BS(K,J))) THEN

C     INITIAL SUM
C     -----------

      DO 130 Z=1,Y
      SUM(Z)=0.0
130   CONTINUE

      QL=RS(K)-1
      DO 140 P=1,K-1

C     FIND THE INDEX OF U AND L
C     -------------------------

      IF (BTOL(BS(K,P))) THEN
      QL=QL+1
      IF (BTOL(BS(P,J))) THEN


      QU=RS(P)-1
      DO 150 KK=1,J
      IF (BTOL(BS(P,KK))) QU=QU+1
150   CONTINUE

C     FIND THE SUM
C     ------------

      DO 160 Z=1,Y
      SUM(Z)=NZS(Z,QL)*NZS(Z,QU)+SUM(Z)
160   CONTINUE

      ENDIF
      ENDIF
140   CONTINUE

C     OBTAIN THE U
C     ---------------

      QQ=QL
      QL=QL+1
      DO 170 KK=K,J
      IF (BTOL(BS(K,KK))) QQ=QQ+1
170   CONTINUE

      DO 180 Z=1,Y
      NZS(Z,QQ)=(NZS(Z,QQ)-SUM(Z))/NZS(Z,QL)
180   CONTINUE
      ENDIF
120   CONTINUE
      ENDIF
10    CONTINUE


      RETURN
```

```
      END
C-----------------------------------------------------
C
C                  FILE NAME SXY
C
C     THIS SUBROUTINE IS USED TO SOLVE THE VECTORS
C     Y AND X FOR THE SUBCIRCUITS.
C
C-----------------------------------------------------

      SUBROUTINE SXY (NZS,BRS,BS,RS,NS,Y,XS)

      ROWWISE BS(10,10)

      REAL NZS(1000,50),XS(1000,10),YS(1000,10),
     1 BRS(1000,10),SUM(1000)

      INTEGER RS(10),NS,L,QQ,K,P,
     1 Z,J,DD,Y

      BIT BS

C     FIND THE FIRST ELEMENT OF Y
C     ---------------------------

      DO 10 Z=1,Y
      YS(Z,1)=BRS(Z,1)/NZS(Z,1)
10    CONTINUE

C     CALCULATE THE Y (TO 20)
C     -----------------------

      DO 20 K=2,NS

C     INITIAL SUM
C     -----------

      DO 30 Z=1,Y
      SUM(Z)=0.0
30    CONTINUE

C     CALCULATE THE SUM
C     -----------------

      DD=RS(K)-1
      DO 40 J=1,K-1
      IF (BTOL(BS(K,J))) THEN
      DD=DD+1

      DO 50 Z=1,Y
      SUM(Z)=NZS(Z,DD)*YS(Z,J)+SUM(Z)
50    CONTINUE

      ENDIF
40    CONTINUE


C     OBTAIN THE Y
C     ------------

      DD=DD+1
      DO 20 Z=1,Y
      YS(Z,K)=(BRS(Z,K)-SUM(Z))/NZS(Z,DD)

20    CONTINUE

C     CALCULATE THE LAST ELEMENT OF X
C     -------------------------------
```

```
      DO 60 Z=1,Y
      XS(Z,NS)=YS(Z,NS)
60    CONTINUE

C     CALCULATE THE X (TO 70)
C     -----------------------

      DO 70 K=1,NS-1

C     FIND THE INDEX OF L
C     -------------------

      L=NS-K
      QQ=RS(L)-1

      DO 80 P=1,L
      IF (BTOL(BS(L,P))) QQ=QQ+1
80    CONTINUE

C     INITIAL THE SUM
C     ---------------

      DO 90 Z=1,Y
      SUM(Z)=0.0
90    CONTINUE

C     CALCULATE THE SUM
C     -----------------

      DO 100 P=L+1,NS
      IF (BTOL(BS(L,P))) THEN
      QQ=QQ+1

      DO 110 Z=1,Y
      SUM(Z)=NZS(Z,QQ)*XS(Z,P)+SUM(Z)
110   CONTINUE

      ENDIF
100   CONTINUE

C     OBTAIN THE X
C     ------------

      DO 120 Z=1,Y
      XS(Z,L)=YS(Z,L)-SUM(Z)
120   CONTINUE

70    CONTINUE

      RETURN
      END
C-------------------------------------------------------
C
C                   FILE NAME UPNM
C
C     THIS SUBROUTINE IS USED TO DO THE UPDATA
C     MAIN NETWORK FOR NONLINEAR DEVICES. ALL OF
C     FUNCTIONS OF NONLINEAR DEVICES IN MAIN
C     CIRCUIT ARE WRITTEN AS THE POLYNOMIAL OF
C     VOLTAGE V.
C
C-------------------------------------------------------

      SUBROUTINE UPNM (NZ,BR,NZP,NZPC,BPC,BP,PO,D,X,N,NL,CI,M)

      REAL NZ(3100),X(1100),BR(1100),PO(20,5),G(20),
```

```
     1  TX(1100),TI,V,JT(10),GT,NZO(20)

        INTEGER D(10,4),CI(1100),N,
     1  NL,P,I,J,M,NZPC(20),BPC(10)

        BIT NZP(3100),BP(1100)

C       FIND THE OLD ORDER OF X
C       -----------------------


        DO 10 I=1,N
        TX(CI(I))=X(I)
10      CONTINUE

C       CALCULAE THE G AND JT  (TO 20)
C       ------------------------------

        P=0
        DO 20 I=1,NL
        IF (D(I,4).EQ.0) THEN
        V=TX(D(I,1))
        ELSE
        V=TX(D(I,1))-TX(D(I,2))
        ENDIF

C       CALCULATE THE POLYNOME
C       ----------------------

        GT=0.0
        DO 30 J=1,10
        GT=GT*V+PO(J,D(I,3))
30      CONTINUE

C       OBTAIN THE JT
C       -------------

        TI=0.0
        DO 35 J=11,20
        TI=TI*V+PO(J,D(I,3))
35      CONTINUE

        JT(I)=GT*V-TI


C       OBTAIN THE G
C       ------------

        P=P+1
        G(P)=GT
        IF (D(I,4).EQ.0) GO TO 20
        P=P+1
        G(P)=-GT
20      CONTINUE

C       OBTAIN THE NZO
C       --------------

        DO 40 I=1,P
        NZO(I)=G(NZPC(I))
40      CONTINUE

C       UPDATA THE NZ
C       -------------
```

```
      P=0
      DO 50 I=1,M
      IF (BTOL(NZP(I))) THEN
      P=P+1
      NZ(I)=NZO(P)
      ENDIF
50    CONTINUE

C     UPDATA THE BR
C     -------------

      P=0
      DO 60 I=1,N
      IF (BTOL(BP(I))) THEN
      P=P+1
      BR(I)=JT(BPC(P))
      ENDIF
60    CONTINUE

      RETURN
      END
C-------------------------------------------------------------
C
C                    FILE NAME UPI
C
C     THIS SUBROUTINE IS USED TO UPDATA THE INPUT
C     OF THE MAIN CIRCUIT. ASSUME ALL OF THE
C     SUBCIRCUITS HERE HAVE ONLY TWO PORTS.
C
C-------------------------------------------------------------

      SUBROUTINE UPI (NZQ,Y,N,M,NZQC,EG,EJ,E,BQ,BQC,NZ,BR)

      REAL NZ(3100),BR(1100),EG(1000),
     1 EJ(1000),NZO(2000),EO(2000),BRO(1000)

      INTEGER NZQC(2000),BQC(1000),
     1 P,I,Y,N,M

      BIT NZQ(3100),BQ(1100),E(1000)

C     CALCULATE THE EO
C     ----------------

      P=0
      DO 10 I=1,Y
      P=P+1
      EO(P)=EG(I)
      IF (.NOT.(BTOL(E(I)))) GO TO 10
      P=P+1
      EO(P)=-EG(I)
10    CONTINUE

C     OBTAIN THE NZO
C     --------------

      DO 20 I=1,P
      NZO(I)=EO(NZQC(I))
20    CONTINUE

C     UPDATA THE NZ
C     -------------

      P=0
      DO 30 I=1,M
```

```
      IF (BTOL(NZQ(I))) THEN
      P=P+1
      NZ(I)=NZO(P)
      ENDIF
30    CONTINUE

C     OBTAIN THE BRO
C     --------------

      DO 40 I=1,Y
      BRO(I)=EJ(BQC(I))
40    CONTINUE

C     UPDATA THE BR
C     -------------

      P=0
      DO 50 I=1,N
      IF (BTOL(BQ(I))) THEN
      P=P+1
      BR(I)=BRO(P)
      ENDIF
50    CONTINUE

      RETURN
      END
C-------------------------------------------------------------
C
C                  FILE NAME LUM
C
C     THIS SUBROUTINE IS USED TO DO THE LU
C     DECOMPOSITION FOR MATRIX A FOR MAIN
C     CIRCUIT AND SOLVE THE VECTORS Y AND X.
C
C-------------------------------------------------------------

      SUBROUTINE LUM (NZ,X,BR,B,R,N)

      ROWWISE B(1100,1100)

      REAL NZ(3100),X(1100),Y(1100),BR(1100),SUM,
     1 TU(1100),TL(1100)

      INTEGER R(1100),N,I,J,K,QL,QU,Q,
     1 QQ,P,DD,W

      BIT B

C     FIND THE FIRST ELEMENT OF Y
C     ---------------------------

      Y(1)=BR(1)/NZ(1)

C     CALCULATE THE FIRST ROW OF U
C     ----------------------------

      Q=1
      DO 10 W=2,N
      IF (BTOL(B(1,W))) THEN
      Q=Q+1
      NZ(Q)=NZ(Q)/NZ(1)
      ENDIF
10    CONTINUE

C     DO LU DECOMPOSITION AND CALCULATE X Y (TO 20)
C     ---------------------------------------------
```

```
      DO 20 K=2,N

C     GENERATER THE VECTOR TU
C     -----------------------

      DO 30 I=1,K-1
      IF (BTOL(B(I,K))) THEN
      QU=R(I)-1
      DO 35 P=1,K
      IF (BTOL(B(I,P))) QU=QU+1
35    CONTINUE
      TU(I)=NZ(QU)
      ELSE
      TU(I)=0.0
      ENDIF
30    CONTINUE

C     CALCULATE THE KTH COLUMN OF L (TO 40)
C     -------------------------------------

      DO 40 I=K,N
      IF (BTOL(B(I,K))) THEN
      QL=R(I)-1

C     CALCULATE THE SUM
C     -----------------

      SUM=0.0
      DO 50 P=1,K-1
      IF (BTOL(B(I,P))) THEN
      QL=QL+1
      IF (BTOL(B(P,K))) SUM=NZ(QL)*TU(P)+SUM
      ENDIF
50    CONTINUE

C     OBTAIN THE L
C     ------------

      QL=QL+1
      NZ(QL)=NZ(QL)-SUM
      ENDIF
40    CONTINUE

C     CALCULATE THE Y
C     ---------------

      SUM=0.0
      DD=R(K)-1
      DO 60 J=1,K-1
      IF (BTOL(B(K,J))) THEN
      DD=DD+1
      SUM=NZ(DD)*Y(J)+SUM
      ENDIF
60    CONTINUE

      Y(K)=(BR(K)-SUM)/NZ(DD+1)

C     GENERATE THE VECTOR TL
C     ----------------------

      QL=R(K)-1
      DO 70 I=1,K-1
      IF (BTOL(B(K,I))) THEN
      QL=QL+1
      TL(I)=NZ(QL)
      ELSE
```

```
      TL(I)=0.0
      ENDIF
70    CONTINUE

C     CALCULATE THE KTH ROW OF U (TO 80)
C     ----------------------------------

      IF (K.NE.N) THEN
      DO 80 J=K+1,N
      IF (BTOL(B(K,J))) THEN

C     CALCULATE THE SUM
C     -----------------

      SUM=0.0
      DO 90 I=1,K-1
      IF (BTOL(B(I,J))) THEN
      QU=R(I)-1
      DO 100 P=1,J
      IF (BTOL(B(I,P))) QU=QU+1
100   CONTINUE

      SUM=NZ(QU)*TL(I)+SUM
      ENDIF

90    CONTINUE

C     OBTAIN THE U
C     ------------

      QU=QL
      DO 110 P=K,J
      IF (BTOL(B(K,P))) QU=QU+1
110   CONTINUE

      NZ(QU)=(NZ(QU)-SUM)/NZ(QL+1)

      ENDIF
80    CONTINUE
      ENDIF
20    CONTINUE

C     CALCULATE THE X
C     ---------------

      X(N)=Y(N)

      DO 120 K=1,N-1

C     FIND THE INDEX OF U
C     -------------------


      L=N-K
      QQ=R(L)-1
      DO 130 P=1,L
      IF (BTOL(B(L,P))) QQ=QQ+1
130   CONTINUE

C     CALCULATE THE SUM
C     -----------------

      SUM=0.0
      DO 140 P=L+1,N
      IF (BTOL(B(L,P))) THEN
      QQ=QQ+1
      SUM=NZ(QQ)*X(P)+SUM
      ENDIF
```

```
140     CONTINUE

C       OBTAIN THE X
C       ------------

        X(L)=Y(L)-SUM
120     CONTINUE

        RETURN
        END
C-------------------------------------------------------
C
C                   FILE NAME OUT
C
C       THIS SUBROUTINE IS USED TO OUTPUT THE
C       FINAL RESULTS OF MMNA PROGRAM.
C
C-------------------------------------------------------

        SUBROUTINE OUT (N,NS,Y,X,XS,T1)

        REAL T1,X(1100),XS(1000,10)

        INTEGER N,NS,I,Y

        WRITE (6,4) Y

        WRITE(6,1) (X(I+Y-1),I=1,N-Y+1)

        WRITE(6,2) (XS(1,J),J=1,NS)

        WRITE(6,3) T1

1       FORMAT (/25X,≠THE FINALE RESULTS OF THE MAIN NETWORK≠//
     1  10X,5F15.8//)

2       FORMAT (/25X,≠THE FINALE RESULTS IN THE FIRST OF SUBNETWORKS≠//
     1  20X,4F15.8//)

3       FORMAT (////35X,≠THE TOTAL CPU TIME≠//35X,≠T =≠,F18.10/)

4       FORMAT (////30X,≠THE NUMBER OF SUBNETWORKS IS≠,I8//25X,
     1  ≠**************************************************≠///)

        RETURN
        END
```

APPENDIX E: THE PROGRAM FOR GENERATING INPUT DATA SETS

This program generates the input data sets for 500 identical subcircuits in the circuit being simulated.

```
C-----------------------------------------------------------------
C
C                       FILE NAME INP
C
C       THIS SUBROUTINE IS USED TO GENERATE THE DATA OF THE
C       EXAMPLE CIRCUIT. LET THE SYMBOLICA PROCESSING AND THE
C       ROW COLUMN EXCHANGING HAVE BEEN DONE HERE.
C
C-----------------------------------------------------------------

        SUBROUTINE INP (IBR,IBRS,X,XS,INZ,INZS,B,BS,R,RS,D,DS,CI,CIS,
     1  M,MS,NZP,NZPC,NZPS,NZPSC,NZQ,NZQC,E,BP,BPC,Y,BPS,BPSC,
     1  BQ,BQC,NL,NLS,N,NS,PO,BPG,BPP,BPV)

        ROWWISE B(1100,1100),BS(10,10)

        REAL INZ(3100),INZS(1000,50),IBR(1100),IBRS(1000,10),X(1100),
     1  XS(1000,10),PO(20,5)

        INTEGER N,NS,NL,NLS,M,MS,Y,BPG,BPP,R(1100),RS(10),D(10,4),
     1  DS(4,4),CI(1100),CIS(10),NZPC(20),NZPSC(8),NZQC(2000),BPC(10),
     1  BQC(1000),BPV(1000),BPSC(4),BB(10),RR(10)

        BIT B,BS,NZP(3100),NZQ(3100),
     1  E(1000),NZPS(50),BQ(1100),BPS(10),BP(1100)


        READ(5,1) Y
1       FORMAT(10I4)

        READ(5,1) (CI(I+Y),I=1,9)

        READ(5,1) (RR(I),I=1,9)

        READ(5,1) (RS(I),I=1,8)

        M=3*Y+35
        MS=27
        N=Y+9
        NS=8
        NL=2
        NLS=2

        DO 5 I=1,N
        DO 5 J=1,N
        B(I,J)=B≠0≠
5       CONTINUE

        DO 6 I=1,Y
        B(I,I)=B≠1≠
        B(I,Y+8)=B≠1≠
        B(N,I)=B≠1≠
6       CONTINUE

        DO 7 I=1,9
        READ(5,1) (BB(K),K=1,9)
        DO 7 J=1,9
7       IF (BB(J).NE.0) B(I+Y,J+Y)=B≠1≠

        DO 8 I=1,8
        READ(5,1) (BB(K),K=1,8)
        DO 8 J=1,8
8       IF (BB(J).NE.0) BS(I,J)=B≠1≠

        DO 9 I=1,Y
        R(I)=2*I-1
        INZ(2*I-1)=-1.0
```

```
      INZ(2*I)=0.0
9     INZ(2*Y+I+31)=1.0

      READ(5,2) (INZ(2*Y+I),I=1,31)
      READ(5,2) (INZ(3*Y+I+31),I=1,4)
      READ(5,2) (INZS(1,I),I=1,27)
2     FORMAT(5F10.5)

      DO 10 I=1,2
      READ(5,1) (D(I,J),J=1,4)
      READ(5,1) (DS(I,J),J=1,4)
10    CONTINUE

      DO 11 I=1,M
      NZP(I)=B≠0≠
11    NZQ(I)=B≠0≠

      DO 12 I=1,N
      BP(I)=B≠0≠
12    BQ(I)=B≠0≠

      DO 13 I=1,8
      CIS(I)=I
13    BPS(I)=B≠0≠

      DO 14 I=1,27
14    NZPS(I)=B≠0≠

      DO 15 I=1,Y
      CI(I)=I+9
      NZQC(I)=I
      NZQ(2*I)=B≠1≠
      BPU(I)=Y+8
      BQ(I)=B≠1≠
      BQC(I)=I
15    E(I)=B≠0≠

      DO 16 I=1,4
      NZPC(I)=I
      NZPSC(I)=I
      BPC(I)=I
16    BPSC(I)=I

      NZP(2*Y+4)=B≠1≠
      NZP(2*Y+5)=B≠1≠
      NZP(2*Y+28)=B≠1≠
      NZP(2*Y+30)=B≠1≠

      NZPS(3)=B≠1≠
      NZPS(4)=B≠1≠
      NZPS(6)=B≠1≠
      NZPS(7)=B≠1≠

      BP(Y+3)=B≠1≠
      BP(Y+8)=B≠1≠

      BPS(3)=B≠1≠
      BPS(4)=B≠1≠

      DO 17 I=1,Y
      DO 17 J=1,8
      IBRS(I,J)=0.0
17    XS(I,J)=0.0

      DO 18 I=1,N
      IBR(I)=0.0
18    X(I)=0.0
```

```
      BPG=8
      BPP=2

      IBR(Y+1)=15.0
      IBR(Y+2)=0.7
      IBR(N)=-1.0E-9

      IBRS(1,1)=15.0

      XS(1,1)=15.0
      XS(1,2)=5.0
      XS(1,3)=7.0
      XS(1,4)=4.0

      DO 19 I=1,Y
      DO 20 J=1,27
20    INZS(I,J)=INZS(1,J)
      XS(I,2)=XS(1,2)
      XS(I,1)=XS(1,1)
      XS(I,3)=XS(1,3)
      XS(I,4)=XS(1,4)
      IBRS(I,1)=IBRS(1,1)
19    CONTINUE

      DO 21 I=1,9
21    R(I+Y)=2*Y+RR(I)

      X(Y+1)=15.0
      X(Y+3)=6.0
      X(Y+2)=14.3
      X(Y+8)=5.0

      DO 22 I=1,20
      DO 22 J=1,2
22    PO(I,J)=0.0

      PO(10,1)=0.001
      PO(19,1)=0.001
      PO(20,1)=0.0005
      PO(2,2)=2.205
      PO(11,2)=0.245
      PO(10,3)=0.0002
      PO(19,3)=0.0002
      PO(20,3)=-0.000005

      RETURN
      END
```

```
500
  1   2   3   5   6   7   8   4   9
  1   2   4   7  11  15  21  28  32
  1   2   3   6   9  13  18  22
  1   0   0   0   0   0   0   0   0
  1   1   0   0   0   0   0   0   0
  1   0   1   0   0   1   0   0   0
  0   1   1   1   0   1   0   0   0
  0   1   0   1   1   1   0   0   0
  1   0   1   1   0   1   0   1   1
  1   0   1   0   1   1   1   1   1
  0   0   1   0   0   1   0   1   1
  0   0   1   0   0   1   0   1   1
  1   0   0   0   0   0   0   0
  0   1   0   0   0   0   0   0
  0   1   1   0   0   1   0   0
  0   1   0   1   0   0   1   0
  1   0   1   0   1   1   0   0
```

```
 1   1   1   0   0   1   1   0
 0   1   0   1   0   1   1   0
 0   1   1   1   0   1   1   1
 1.0         1.0        -1.0          0.0         0.0
-1.0        -0.01       -0.1          0.1101      0.0
 0.01       -0.01       -1.0          0.0        -0.00001
 0.10002    -0.1        -0.5         -0.00001     0.01
 0.00001    -0.00001     1.0          1.0         1.0
 0.0         0.99        0.0          0.0         0.0
-1.0
-0.00001    -0.5         0.00101     -1.0
 1.0         1.0         0.0          0.0        -1.0
 0.0         0.0        -1.0          0.02       -0.02
 1.0         0.0        -0.02        -0.00001     0.02001
-1.0         0.99       -0.00001      0.00201     0.5
-1.0         0.00002    -0.00001     -0.00001     0.5
 0.01        1.0
 1   3   1   1
 2   3   3   1
 3   4   2   1
 2   4   2   1
```