ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

DESIGN OF ELECTRICAL NETWORKS THROUGH COMPUTER-AIDED OPTIMIZATION TECHNIQUES

A Thesis presented to the School of Graduate Studies in partial fulfillment of the requirements for the Master of Science Degree in Electrical Engineering

By

Solomon Zewde

Addis Ababa

June 1993

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

DESIGN OF ELECTRICAL NETWORKS THROUGH COMPUTER-AIDED OPTIMIZATION TECHNIQUES

*A **Thesis** presented to the School of Graduate Studies in partial fulfillment of the requirements for the Master of Science Degree in Electrical Engineering*

By

Solomon Zewde

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUTE STUDIES

DESIGN OF ELECTRICAL NETWORKS THROUGH COMPUTER-AIDED OPTIMIZATION TECHNIQUES

BY
SOLOMON ZEWDE

APPROVAL BY BOARD OF EXAMINERS:

DR. GIRMA MULLISA
Chairman,
Department Graduate Committee

DR. WOLDE-GHIORGIS W.
Advisor

DR. GASHAW TEFERA
External Examiner

DR. TESFAYE BAYOU
Internal Examiner

Acknowledgements

First and foremost of all, I would like to express my deepest gratitudes to my thesis advisor Dr. Wolde-Ghiorghis W., Associate Professor of Electrical Engineering at the Electrical Engineering Department of the Addis Ababa University, for his follow-ups, pivotal comments and practical suggestions throughout this study.

I also thank Dr. Girma Mullisa, Head, Department of Electrical Engineering, Addis Ababa University, who, as I see the matter now, has given me an intact background on optimization theory -- both in class and through the informal discussions I had with him. Besides, his excellency has always been wiling to help and furnished me with the book COMPUTER ORIENTED CIRCUIT DESIGN, without which an even more effort would have been required to materialize the meat of the study.

I have benefitted from the thoughtful comments of Professor Samuel Lakeou, Head, Department of Electrical Engineering, University of the District of Columbia, Washington D.C., USA, concerning the overall content of the study and my preliminary seminar on the subject during his visit to the Addis Ababa University in May 1992. Dr. Samuel has also sent me the original memo of SPICE together with a printout of the source program. I must therefore extend my respects and sincere thanks to him.

Dr.-Ing. Hailu Ayele, the Dean, Faculty of Technology, Addis Ababa University, has been very helpful. I reserve special words of thanks to him. I also thank the staff of the Computer Center of the Faculty of Technology, Addis Ababa University, and Ato Bluye Hadis of the NCR corporation for the various forms of help they rendered in making this study both a success and a reality.

On the other hand, I will be failing in duty if I do not record my immeasurable indebtedness to the Ethiopian Freight Transport Corporation for allowing me this rather unique opportunity of following an M.Sc. program at the Addis Ababa University.

Finally, I thank all those others upon whose shoulders I had to stand with that rather ambitious hope of looking any further.

List of Tables and Abbreviations

A. List of Tables

B. List of Abbreviations

CAD	Computer-Aided Design;
VCCS	Voltage Controlled Current Source;
VCVS	Voltage Controlled Voltage Source;
CCCS	Current Controlled Current Source;
CCVS	Current Controlled Voltage Source;
v-i	voltage and current;
NAM	Nodal Admittance Matrix;
KCL	Kirchhoff's Current Law;
KVL	Kirchhoff's Voltage Law;
det	determinant;
diag	diagonal;
RIM	Reduced Incidence Matrix;
RHS	Right Hand side;
LHS	Left Hand Side;
Re(.)	Real part of;
Im(.)	Imaginary part of;
p.d.	positive definite;
LM	Levenberg - Marquardt;
P-POND	A Pascal Program for Optimal Network Design.

List of Figures

Abstract

The objective of the study reported in this thesis has been to develop a unified algorithm for the computer-aided design of a wide range of electrical network configurations. While the design has been based on well-known techniques established earlier by other researchers, the present work has made significant contributions in simplifying the uses of nodal equation formulation method, graph theoretic concepts and topological formulas, as well as applications of a modified least p-th Taylor method, component damping techniques, sensitivity analyses and related design concepts. The basic computer-aided design approach for a selected network was realized by first generating a voltage gain transfer function from the network connectivity details. Information concerning the gradient vector necessary in the course of an optimization process was then derived through a direct method of sensitivity analysis, without performing the normally needed first order partial differentiations. Comparison of a transfer function against desired response was preceded by sampling of the latter at selected frequency points and the whole adjustment was eventually automated via the least p-th Taylor method of optimization. It has been established that this approach avoided the time consuming calculation of a Hessian matrix that is usually required for performing network optimization. Convergence properties of the least p-th Taylor method were improved through a use of Fletcher's modification of the classical Levenberg-Marquardt method together with component damping techniques. The concept of dynamic memory allocation has also been exploited. The resulting computer-aided network design technique was therefore efficient in terms of both processing time and memory requirements and the entire package has been termed as a Pascal program for optimal network design or P-POND. The algorithm has been fully tested in its use for designing both passive and active network types by specifying initial network parameters and input-output relations.

Table of Contents

1. GENERAL INTRODUCTION

Optimization problems are at the core of Engineering Design [1] and the electrical trade is no exception. In fact, electrical network design quite often calls for optimization in the design parameters and parameter adjustments are common exercises. In the past, these steps were performed [in vain for large systems] without resorts to the computer, primarily because of the absence from the scene of the tool itself and, to a lesser extent, because of the lack of suitable algorithms for such adaptations. Today, many design areas are almost completely handled by the computer. A number of analysis and design programs have already made their ways into the market and the results achieved are extraordinarily attractive from various considerations.

The subject of this study, namely the design of electrical networks through computer-aided optimization techniques, is variously known as Computer-Aided Network Optimization [3], Fully Automated Network Design [4], or, simply, Automated Design [5]. Some authors use the terms *non-linear programming* and *computer-aided redesign* [42: p.219]. The basic procedure is iterative in nature and the need for it is felt ''in the frequent case when the synthesis procedures of classical circuit theory are, for some reason, inapplicable'' [2: p.191]. Although the task may be accomplished equally well in both of the frequency and time domain approaches, only the former shall be considered here.

In the present study, it can be stated at the outset that there are three steps in the computer-aided design formulations of electrical networks. These are:

1) formulation of the network structure;
2) formulation of some suitable error functional which accommodates the design criteria and serves as a measure of the error between the actual and desired network responses; and,
3) analysis of the network and use of some suitable optimization scheme to minimize the error functional and come out with optimum values of the design parameters.

Of these three steps, the first is usually considered as being the relatively most difficult and the third as falling in between the other two, closer to the first but farther away from the second. The interrelationships among the steps have been formalized by numerous researchers. The works of Dawson [2], Branin [2], Temes [2], Temes and Calahan [3], Rohrer [4], Director [5], Bandler [7], Calahan [8], Spence [9], Brayton [10], Temes and LaPatra [27], Wing and Behar [36], Cuthbert[42], Adby and Dempster [43], and Fletcher [44] are contributions towards those ends. A flow chart of an optimization process that is based upon the above divisions is included in Appendix A of this thesis.

Step 1, or formulation of the network structure, may be done in a variety of ways and consits of the equation formulation method and the input means to the computer. These issues are discussed in sections 2.1 and 2.2 of the present study, where overviews of input devices and the three classes of the topological, sparse-tableau, and modified tableau methods of network equation formulation are presented.

Step 2, which principally involves formulation of a suitablly defined error functional, is based upon the results of step 1 and, in its turn, forms a basis for step 3.

In step 3, a multi-dimensional optimization procedure is used to find the optimal values of the design parameters. In other words, sets of repeated analyses are performed on the network of step 1 until the conditions of step 2 are satisfied.

The study is thus divided into six chapters. These are General Introduction, Basic Approaches to *CAD* of an Electrical Network, On the Theory of Network Optimization, Computer Algorithm Development Schemes, Applications, and Conclusions.

The theory concerning the implementation of steps 2 and 3 above shall be presented in sections 2.3, 2.4 and 2.5. The remaining parts of chapter 2 are addressed to the questions of equation formulation and ways of compiling network information as described above.

In chapter 3, a study of the theory of network optimization is presented.

The development of the computer program written in Turbo Pascal version 5.0 is discussed in chapter 4.

Chapter 5 contains two application examples: design of a gain equalizer and an example that deals with Guillemin's parallel ladder network. These are meant to represent both of the active and passive design examples. The basic results obtained for the networks are also subjected to relevant comments in the same chapter.

Finally, in chapter 6, conclusions are also included.

In following the above outline, the present study first uses the keyboard and/or an ascii file for feeding network information to the computer. There is also an option for graphic input. Study of the network then proceeds in two entirely independent lines: study of the topological structure and study of the algebraic structure. The *nodal equation method* and *topological formulas* are employed to obtain the *voltage gain transfer function* by first obtaining the networks N_{-i} and N_{-j} from the original network. Networks N_{-i} and N_{-j} have their nodes i and j shorted to the datum node.

After treatments of the theories of sensitivity and optimization, the analysis and optimization stages are coupled using the direct method of sensitivity analysis. This provides gradient information on the performance function in a form of a Jacobian matrix. Of the various optimization methods in this thesis, gradient methods are shown to be most efficient and, out of these, a multi-variable optimization process that uses *the damped least p-th Taylor method* and combines the advantages of the basic first and second order methods of Steepest Descent and Newton's method is utilized to determine the optimal component values for a desired response. The comparison of the desired and computed transfer functions is effected only after sampling the former at selected frequency points.

The need for the damped least p-th method arises since the ordinary least p-th Taylor method is prone to the problems of divergence and floating-point overflow. The former problem is an outcome of the approximations on the Hessian matrix which is required in the course of the optimization while the latter is due to large and/or small numbers occurring in the calculations. The overall effect is that poor initial conditions are not tolerated when component damping techniques and Fletcher's modification of the classical Levenberg - Marquardt algorithm are not utilized to solve these problems.

2. BASIC APPROACHES TO *CAD* OF AN ELECTRICAL NETWORK

2.1 Methods of Compiling Network Information

The first thing that comes to the mind in any computer-oriented network study is usually the input mechanism through which data is fed to the computer. A variety of devices are available for this purpose, the common ones being: the keyboard, the light pen, the tablet and the mouse.

The alphanumeric keyboard is by far the most used input device in computer-aided network design. Its attractiveness lies in the low cost involved. The network of Fig. 2.1, for instance may be coded as shown in Fig. 2.2.

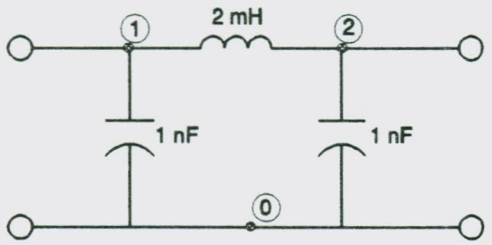


Fig. 2.1: Conventional Network Description.

```
C  1  0  1E-09
L  1  2  2E-03
C  2  0  1E-09
```

Fig. 2.2: A Coding Scheme for the Network of Fig. 2.1.

Circled numbers in Fig. 2.1 represent node numbers.

An alternative coding scheme using the alphanumeric input is provided by the so-called *wiring operators* or *wiring functions*. These operators have the same kind of interpretation as arithmetic operators in an algebraic expression and were first extensively used in the circuit analysis program called MARTHA written in the APL language and operated from time-shared computer terminals [17: pp. 245-249].

Wiring operators are better suited to two-port network descriptions and, even then, the description may not yield an exact resemblance with the conventional circuit diagram since each and every element of the network has to be described in a two terminal

format. The network of Fig. 2.1 above, for example, must first be converted to the form shown in Fig. 2.3 below before the *wiring-operator-description* of Fig. 2.4 is obtained.

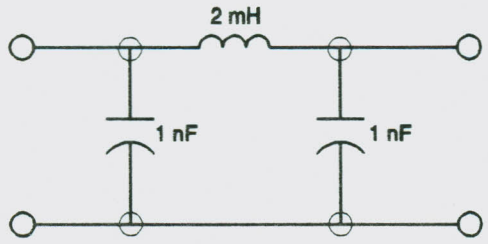


Fig. 2.3: Network of Fig. 2.1 Prepared for Wiring-Operator-Description.

(WP C 1E-09) WC
(WS L 2E-03) WC
(WP C 1E-09)
where: WP = wires in parallel;
WS = wires in series;
WC = wires in cascade.

Fig. 2.4: Wiring-Operator-Description for Fig. 2.1.

The alphanumeric input may also be organized into an ASCII file. The number and location of nodes, the orientations and natures of components, and other related items are fed to the computer by appropriate blends of characters. In the present study, we shall exploit this strategy along with graphical ways of feeding information to the computer.

The light pen, tablet and mouse are devices for what is called *graphic input* which got its conception from the fact that the natural language of the engineer in network design is the schematic. A graphic input may consist of either of the above mentioned three devices with the help of which the designer feeds the circuit data to the computer directly in a schematic form.

Programs like SPICE (Simulation Program with Integrated Circuit Emphasis) use both the alphanumeric and graphic inputs. Other programs like GINA (Graphical Input for Network Analysis)[2] are devoted mainly to graphic input - output. In GINA, the designer enters his schematic by using the light-pen and a set of graphic instructions called *light buttons*. The CRT face is divided into two areas: a *display* area at the center of the screen, and a *control area* at the bottom and that contains the *light buttons*. A command is *picked* from the control area and tracked to the display area by means of the button POSITION. When the circuit schematic is completed, the designer *picks* the ANALYZE button and the computer outputs the network frequency response directly in graphic form. Thus, apart from certain amount of non-display information such as

element value and type of analysis required, the computer - designer communication is entirely graphic.

Graphic input is extremely efficient, versatile and quite fascinating. Unfortunately, it requires special hardware for its implementation and shall not be a topic of further attention here.

2.2 METHODS OF NETWORK EQUATION FORMULATION

2.2.1 Basic Considerations

Consider an electrical network constructed by interconnecting b branch elements. Since each element has a current through and a voltage across it, there are a total of 2b branch variables that need be determined, b voltages and b currents.

To approach the above problem systematically, the information embodied in an electrical network is categorized under two independent structures: a topological structure and an algebraic structure, which represent two independent aspects of the physical network. The topology represents the interconnections between the various branch elements while the algebraic structure has to do with the associated constitutive or v-i relationships. The identification of these two aspects as independent mathematical entities is *fundamental* to the study of electrical network behavior and could be viable since Kirchhoff's laws are true irrespective of the nature of the elements, i.e., these laws are valid whether the elements are linear, non-linear, active, passive, time-variant, time-invariant and etc.

Study of the topological structure of an electrical network reveals that the voltage and current functions of the network are constrained by Kirchhoff's current and voltage laws, KCL and KVL, and, for a network of n nodes there are exactly (n-1) independent KCL equations and (b-n+1) independent KVL equations. The use of KCL in conjunction with KVL therefore yields a total of b independent equations. The algebraic structure also yields a total of b independent equations, thus making the total number of independent equations 2b.

Now, 2b equations are sufficient to solve for the 2b unknowns but this is a substantial task. Ways that do not demand the consideration of all voltages and currents as unknown variables must be sought. This idea leads to the concept of a *basis*, in which branch variables are determined in terms of a smaller subset, the basis, from which an independent system of equations may be constructed.

Three groups of different equation formulation methods shall be studied:

(i) - methods using the topological approach;

(ii) - the sparse-tableau approach; and,

(iii) - the modified-tableau approach.

The former group may further be divided into the next eight methods:

(i) - nodal formulation method;

(ii) - mesh formulation method;

(iii) - tieset method;

(iv) - cutset method;

(v) - hybrid method;

(vi) - nodal method with voltage sources;

(vii) - modified nodal method of Ho et al; and,

(viii) - reduced modified nodal approach.

Topological methods use sets of bases voltages and currents in terms of which all other voltages and currents may be expressed. The sparse-tableau approach includes all network information in non-reduced form. The modified-tableau approach, on the other hand, combines topological concepts with a modification of the tableau formulation.

We shall investigate in brief each of the above ten methods of equation formulation. But first the theory of network topology, a definition of an electrical network itself, and introduction of branch conventions are necessary.

2.2.2 Graph Theoretic Concepts

Euler, the great Swiss Mathematician, wrote perhaps the first paper on Graph Theory back in 1736 when seeking a solution to the celebrated *Königsberger bridge problem*. However, it was the German Physicist Gustav Robert Kirchhoff who first

founded the *Theory of Graphs* or *Network Topology* in its present form in 1847 for its specific application to electrical networks. Graph theory was also used by Sir William Hamilton in 1859 and Maxwell in 1892. The original application of the theory to electrical networks was restricted to passive networks and extension to active networks began only later in 1957.

In Graph theory, the connection features of a network are studied independently of the algebraic relations between the *through* and *across*[12: p.8] variables because "...Kirchhoff's laws are independent of the nature of the elements" [14: p.382]. Hence, we can ignore the nature of the elements and replace them by line segments or *edges* [*branches*] with points or *nodes* at their ends. The result of this practice is a *network graph*, or simply *graph*. More formally, a graph is defined as "a set of nodes and branches with the condition that each branch terminates at each end into a node" [14: p.382].

A node and an edge are *incident* with each other if the former is an end point of the latter. The number of edges incident at a node make the *degree* or *valency* of the node. A *path* between nodes i and j is an ordered sequence of edges in the graph in which every node is of degree two, except nodes i and j, which are of degree 1. Any closed path in a network is called a *loop* or *circuit*.

A graph G, therefore, consists of two sets: a finite set N of nodes and a finite set E of edges; and may be denoted as G(N,E). The symbols n_1, n_2, n_3,... and e_1, e_2, e_3,... shall be used, respectively, to represent nodes and edges. In pictorial representations, a node number shall appear inside a small circle while edge numbers shall be left free.

Thus, edge e_l may be denoted as $e_l = (n_i, n_j)$. As an example, let

$$N = \{ n_1,n_2,n_3,n_4,n_5,n_6 \};$$
$$E = \{ e_1,e_2,e_3,e_4,e_5 \};$$
$$e_1 = (n_1,n_2);$$
$$e_2 = (n_1,n_4);$$
$$e_3 = (n_5,n_6);$$
$$e_4 = (n_1,n_2);$$
$$e_5 = (n_5,n_5).$$

Then, the graph G(N,E) is represented as in Fig. 2.5 next.

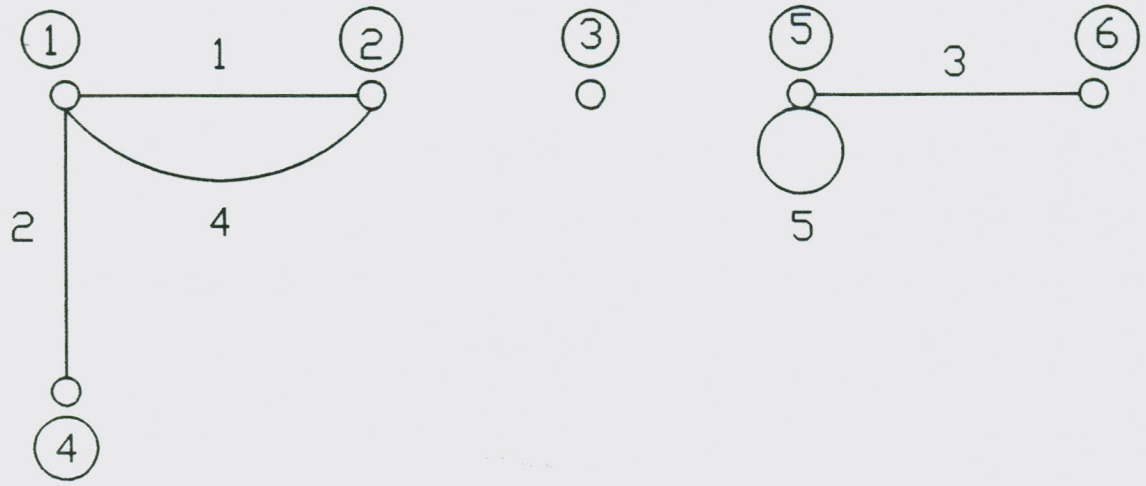


Fig. 2.5: Graph G(N, E).

Since the definition of a graph does not require the nodes to be distinct, more than one edge in E may have the same pair of nodes although the elements of E are still distinct. All edges having the same pair of nodes are called *parallel edges*. Another important observation is that the nodes of an edge need not be distinct. In that case, one gets a *self-looped* edge at the node being considered. Also, a graph may contain a node without an edge. Such graphs are called *degenerate* or *empty* and possess a valency of zero. A node of degree 0 is distinguished as an isolated node while a graph with no nodes [and hence no edges] is known as a *null graph*.

A graph may be *connected*, i.e., there will be a path between every pair of nodes. Otherwise, by definition, it is said to be *disconnected*. Fig. 2.5 above may be seen as an example of a disconnected graph with three parts or *subgraphs*. A subgraph G is another graph S every node of which is a node of G and every edge of which is an edge of G. It is always possible to form S by deleting from G some edges and/or some nodes.

An *oriented* graph is one whose branches are directed and gets its orientation from the reference system used for currents and voltages. The "associated reference direction" convention discussed in section 2.2.4 below shall be used in this work.

Another basic definition is that of a *tree*: a "key concept in the theory of graphs" [20: p.71]. A tree is defined as "...any set of branches that is just sufficient in number

to connect all of the nodes" [15: p.7]. A tree has all the graph nodes but no closed path.

An edge of a tree is called *a tree branch*. The remaining edges are called *chords* or *links*. The term *branch* includes both *links* and *tree branches*. The graph for the bridged-T network of Fig. 2.6 is indicated in Fig. 2.7 and has the subgraphs of Fig. 2.8.

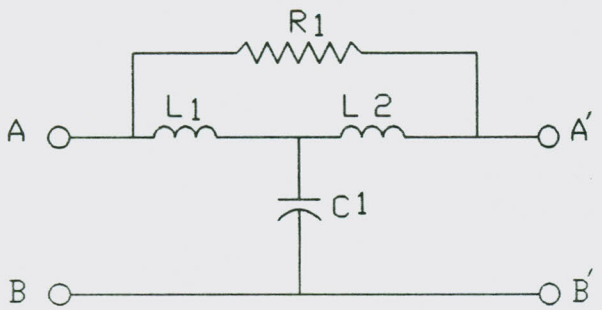


Fig. 2.6: Bridged-T Network.

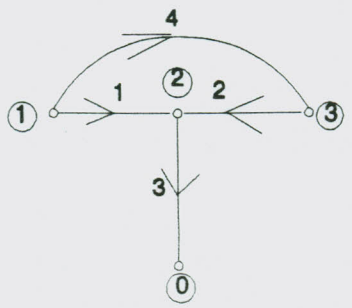


Fig. 2.7: Graph of Fig. 2.6.

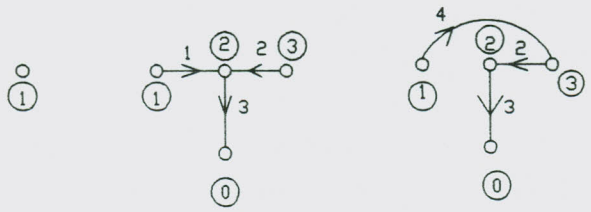


Fig. 2.8: Subgraphs of Fig. 2.6.

The first subgraph in Fig. 2.8 is degenerate. The rest are trees having {1,2,3} and {4,2,3} as their tree branches, respectively. The first edge of the first tree is a link for the second. Edge 4, a tree branch for the latter, is a link for the former tree. **Appendix B** of this thesis contains some useful theorems concerning trees.

Consider next the notion of *k-trees*. Each piece of a k-tree is a tree by itself and the k-tree consists of a total of k trees that are in k pieces. One or more pieces may contain isolated nodes. Although there are *1-tree*, *2-tree*, *3-tree*, and so on, consideration of only the former two is sufficient for our purpose.

A *1-tree* is the same as a tree. A *2-tree* divides a graph into two parts and is defined as "... a pair of unconnected, circuitless subgraphs, each subgraph being connected, which together include all the vertices of the graph. one (or, in trivial graphs, both) of the subgraphs may consist of an isolated vertex" [11: p.159]. Fig. 2.9 on the following page shows a graph and some of its 2-trees.

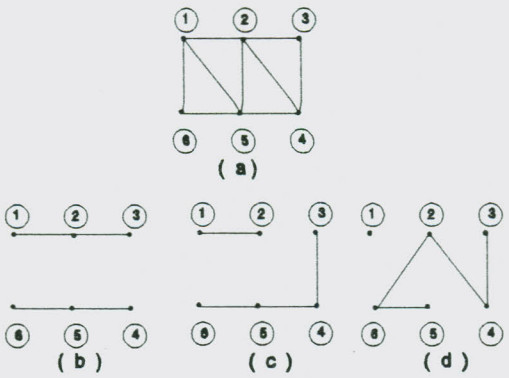
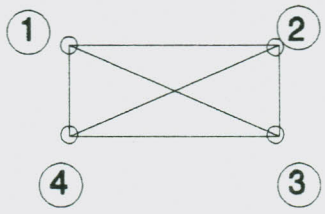


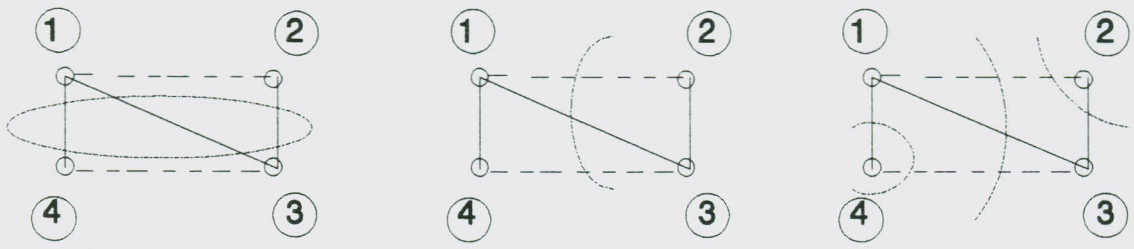
Fig. 2.9: A Graph and some of its two-trees.

A k-tree is denoted by T_k and nodes in different connected parts of a graph by blends of sub and super-scripts. Thus, $T_{k\ ab,cd}$ shows that the nodes a and b are in one part while nodes c and d are in the other. We next look at the concepts of *cut-set* and *tie-set*.

Cut, basic-cut and cut-set of a graph are shown in Fig. 2.10 next:



(a) Graph of a network



(b) a cut (c) a basic cut (d) a cut-set

Fig. 2.10: Cut, basic cut, and cut-set of a graph.

A *cut* is closed surface that breaks a network into two pieces. If the cut cuts just

one tree branch then it is a *basic cut*. A set of basic cuts, one for each tree branch, forms a *cutset*. Removal of branches of a cut-set "...causes the graph to become unconnected into exactly two connected subgraphs" and "...the removal of any proper subset of this set leaves the graph connected" [20, p.83].

Since each tree branch, together with certain link branches, defines a basic cut, a network can have as many basic cuts as the number of its tree branches and, by the first theorem in **Appendix B**, this number is equal to n-1. Also, since there are no loops in a tree, KVL cannot be used to express the tree branch voltages. Tree branch voltages, therefore, form a basis set of network variables.

The presence of each fixed voltage source in a tree reduces the number of basis voltages in a network by one, as a fixed voltage source constrains one node voltage relative to the other. With that consideration, the number of independent voltages in any given network is equal to n-1-v, where v is the number of fixed voltage sources.

A loop that traverses just one link is called a *basic loop* and a set of basic loops, one of which traverses each link, forms a *tieset*. These are shown in Fig. 2.11 below.

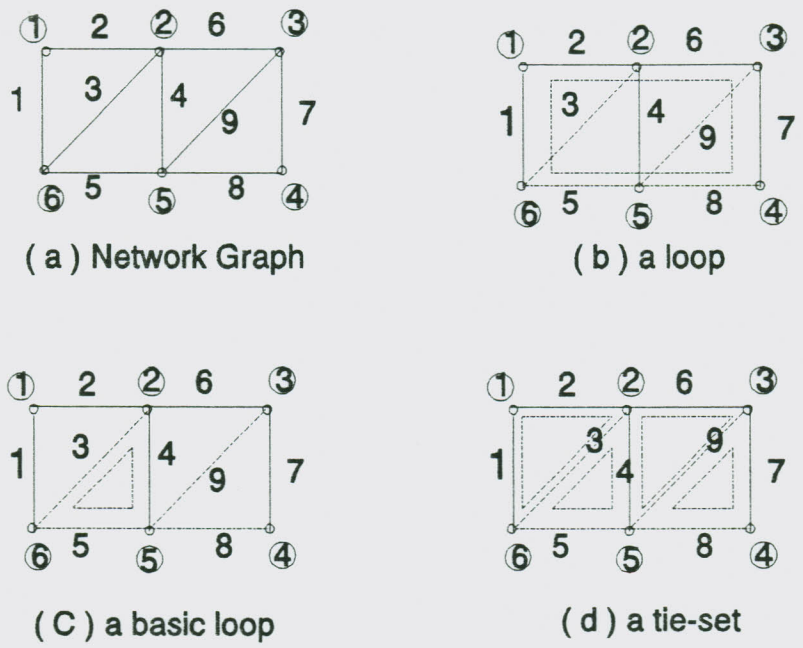


Fig. 2.11: Loop, basic loop and tie-set of a network graph.

The number of basic loops in a network is equal to the number of links in that network and, since there are n-1 tree branches, this number is equal to b-n+1. Also, a given basic loop is associated with a unique link current and, moreover, KCL is satisfied at each node by the link currents. The link currents therefore represent a basis set of network currents.

The presence of fixed current sources must also be accounted for. The inclusion of each fixed current source as link branches constrains one link current and reduces the number of basis currents by one. The total number of independent currents in a network therefore becomes b-n+1-i, where i is the number of independent current sources.

Graph Theory is necessary to collect network information into a set of matrices that efficiently handle the book-keeping chore arising out of network complexities. The four topological matrices of interest here are: the *reduced incidence matrix A* , the *node-to datum path matrix* (or the *path matrix*) B_T, the *reduced circuit matrix C*, and the *reduced cutset matrix D*. The corresponding *complete* matrices are A_a, B_a, C_a, and D_a and contain redundant information.

The *complete incidence matrix* is defined as follows:

> The *complete incidence matrix* $A_a = [a_{ij}]$, of an oriented graph with n nodes and b edges, is a matrix with b rows and n columns. Each row corresponds to an edge i, and each column to a node j, such that
>
> $a_{ij} = 1$, if i is incident at j and away from it;
>
> $a_{ij} = -1$, if i is incident at j and toward it;
>
> $a_{ij} = 0$, if i is not incident at j.

In computer work, the incidence matrix is the standard method used to describe the interconnection of the elements [14: p.386]. The major properties of A_a are:

(i) Every row of A_a has two non-zero entries one +1 and the other -1. This is evident as there are two distinct ends for every branch.

(ii) As a consequence of property (i), any column of A_a can be deleted without information loss.The matrix obtained in this way is called the *reduced incidence matrix* (RIM) and is denoted by **A**. The vertex corresponding to the column deleted is identified as the reference or datum vertex.

(iii) Another consequence of property (i) is that the sum of all columns of A_a

results a row of zeros. Hence, the rank of A_a is at most n-1. The rank of A is also n-1.

Other properties of the RIM are summarized in **Appendix C** of this thesis. The RIM for the bridged-T network of Fig. 2.6 is indicated in Fig. 2.12:

		nodes		
		1	2	3
	1	1	-1	0
branches	2	0	-1	1
	3	0	1	0
	4	1	0	-1

Fig. 2.12: The RIM for the network of Fig. 2.6.

The path matrix B_T has tree branches included in passing from each node to the reference node. B_T is also an edge-node matrix, with the nodes in columns and the edges in rows. The path matrix for the last subgraph in Fig. 2.8 is shown in Fig. 2.13 next.

		nodes		
		1	2	3
	4	1	-1	0
branches	2	0	1	-1
	3	0	0	1

Fig. 2.13: The Path-Matrix for the last sub-graph in Fig. 2.8.

The complete circuit matrix, denoted by $C_a = [c_{a,ij}]$, is a branch-loop matrix whose

columns show the branches included in each loop. It is defined as follows:

$c_{a,ij}$ = 1, if branch i is contained in loop j and the loop and branch references agree;

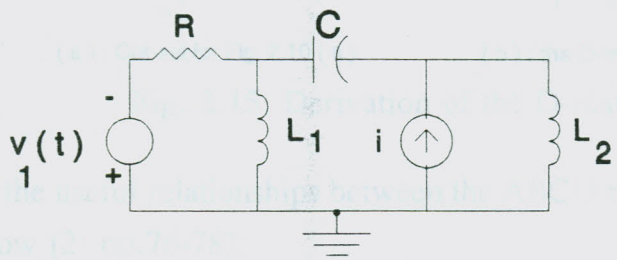
$c_{a,ij}$ = -1, if branch i is contained in loop j but the loop and branch references do not agree;

$c_{a,ij}$ = 0, if branch i is not contained in loop j.

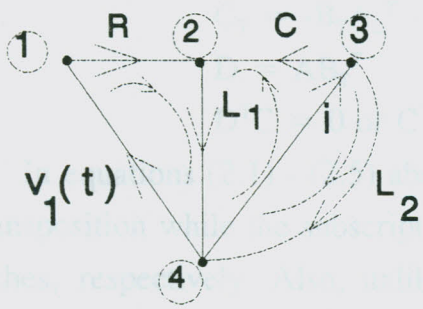
The *reduced circuit matrix C* is defined as follows:

> The *reduced circuit matrix* $C = [c_{ij}]$ of an oriented graph is a matrix consisting of b rows and *l* basic loops showing in columns the branches i included in each basic loop j such that:
>
> c_{ij} = 1, if i is in j and loop and branch references agree;
>
> c_{ij} = -1, if i is in j but the two references do not agree;
>
> c_{ij} = 0, if i is not in j.

As an example, consider Fig. 2.14 below.



(a) : example network



(b) : graph of (a) showing basic loops

branches	basic loops R	C	L_2
$v_1(t)$	1	0	0
R	1	0	0
L_1	1	1	0
C	0	1	0
L_2	0	0	1
i	0	1	1

(c): the c - matrix for (a)

Fig. 2.14: Derivation of the C-matrix.

2.2.3 Definition of an Electrical Network

As a prelude to further development, consider first the three fundamental laws of electrical network theory:

$$A^T i(t) = 0 \tag{2.6}$$

$$C^T v(t) = 0 \tag{2.7}$$

$$v(t) = L\frac{d\ i(t)}{dt} + R\ i(t) + D\int_0^t i(x)dx + e(t) + V_c(0^+) \tag{2.8}$$

where

L, R, D are matrices of order b x b each, b = no. of branches;

e(t) = driving function(current and/or voltage generator);

$V_c(0^+)$ = initial condition of the system(network);

Equations (2.6) and (2.7) above are Kirchhoff's two laws, KCL and KVL, in their *most original* forms and represent the topological properties of an electrical network while the integro-differential equation (2.8) defines the v-i relations for each network element.

A network may thus be characterized by the entries in the matrices L, R, and D as shown in Table 2.1 next.

Table 2.1: A Classification Scheme for Electrical Networks.

Entries in L, R, and D	Network type	Remarks
1. symmetric	reciprocal	otherwise non-reciprocal
2. independent of branch voltages and currents	linear	otherwise linear
3. functions of time but not branch voltages & currents	linear time variable	-
4. positive semidefinite or definite with e(t) equal to zero	passive	otherwise active
5. constants only	Linear time invariant	-

On the other hand, since branch voltages are linear combinations of node voltages and branch currents are linear combinations of mesh currents, there are two alternative and mathematically equivalent forms of equations (2.6) and (2.7). This idea leads to the concepts of *node* and *loop transformations*.

Node transformations are described by

$$v(t) = A\, v_n(t) \tag{2.9}$$

where $v_n(t)$ is a vector of node-to-datum voltages.

Loop transformations are described by

$$i(t) = C\, i_m(t) \tag{2.10}$$

where $i_m(t)$ is a vector of mesh currents.

The equivalence of equations (2.9) and (2.10) to equations (2.6) and (2.7) may be shown as follows:

$C^T v(t) = C^T A\, v_n(t)$, pre-multiplying (2.9) by C^T.

But, from equation (2.2), $C^T A$ is a null matrix.

Hence, the result in equation (2.6).

Similarly, using the node transformation equation,

$A^T i(t) = A^T C\, i_m(t)$.

But, from equation (2.2) again, $A^T C$ is a null matrix.

Hence, the result in equation (2.7).

An electrical network may now be defined in precise terms:

Definition. *An electrical network is a directed linear graph with two bounded voltage and current functions of time $v_b(t)$ and $i_b(t)$ associated with each branch and satisfying the three fundamental laws of electrical network theory.*

2.2.4 Branch Conventions

The *associated reference direction convention*, in which a positive current enters an edge by the terminal marked with a plus voltage sign and leaves the same edge by the terminal marked with a minus voltage sign, shall be used to account for the voltage and current references. This choice of current direction immediately fixes the voltage polarity thus rendering the indication of polarity, either by means of an arrow or by + and - signs, redundant. Hence, along with topological considerations, we replace the associated (combined) references of Figs.

2.16(a) and (b) next by that of Fig. 2.16(c) below.

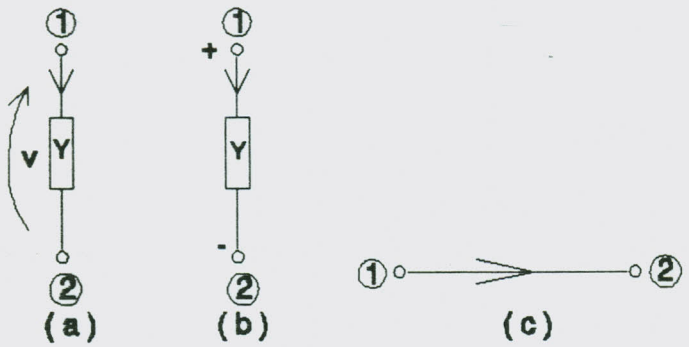


Fig. 2.16: The Associated Reference Direction Convention.

Another useful assumption concerns that of a general branch. The most general branch shall be assumed to consist of both voltage and current sources as per *Kron's branch convention* [2: p.78]. This is reproduced in Fig. 2.17 next with some revisions in notations:

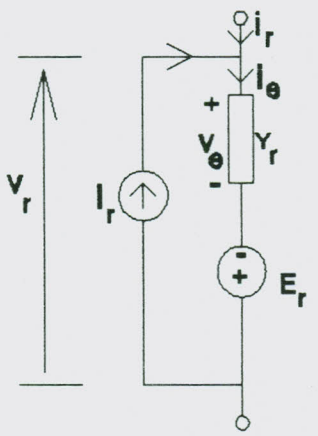


Fig. 2.17: Kron's branch convention.

i_r, v_r = current and voltage variables of the r-th branch.
I_r, E_r = current and voltage-source vectors of the r-th branch;
i_e, v_e = current through and voltage across Y_r;
Y_r = the r-th branch admittance element;

2.2.5 Nodal Formulation Method

We now explore the various equation formulation methods.

The nodal formulation method shall be considered here. The method does not need the selection of trees and is based upon n-1 node-to-datum voltages which form a basis of branch voltages.

From Fig. 2.17, for the r-th branch, $i_r = i_e - I_e = Y_r v_e - I_e$.

And, for the whole network, $i(t) = Y\, v(t) - I(t)$.

Pre-multiplying the last result by A^T yields

$$A^T i(t) = A^T Y\, v(t) - A^T I(t). \tag{2.11}$$

Using equations (2.6) and (2.9), equation (2.11) becomes:

$$A^T Y A\, v_n = A^T I(t). \tag{2.12}$$

Letting $A^T I(t) = J_n$ (= *the nodal excitation current vector*) and $A^T Y A = Y_n$ (= the *Nodal Admittance Matrix* or *NAM* for short), equation (2.12) reduces to the *basic node voltage equation*:

$$Y_n v_n = J_n \tag{2.13}$$

The *branch admittance matrix Y* for the entire network may be filled by inspection. The diagonal entries correspond to self-admittances and the off-diagonal terms to transadmittances between pairs of branches. Equality between corresponding off-diagonal terms indicates mutual admittances. Active devices(dependent sources) are symbolized by off-diagonal terms.

The matrix Y_n may also be filled efficiently. Let all circuit admittances be denoted by y_{kl}^{ij}. When i=k and j=l, a self-admittance is obtained between node i and j. When $i \neq k$ and $j \neq l$, y_{kl}^{ij} is the mutual admittance connected between nodes i and j and controlled by nodes k to l. Each admittance element is entered into four positions. In positions (i,k) and (j,l), the element is added to the entry, and in positions (j,k) and (i,l) it is subtracted from the entry.

Nodal formulation is applicable only to voltage-controlled current-defined branch relations. Due provisions must therefore be made to convert multi-terminal devices to this form when these devices lack the representation. Voltage sources, for instance, are replaced by their Norton equivalent as in Fig. 2.18. Similarly, a VCVS may be converted to a VCCS as in Fig. 2.19.

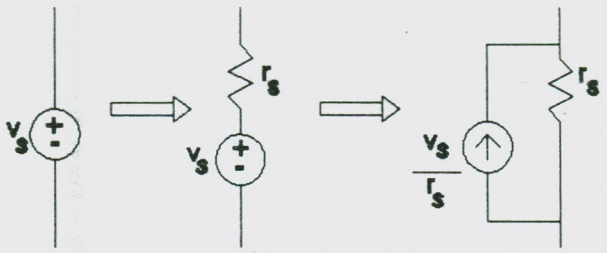


Fig. 2.18: Transformation of an Independent Voltage Source.

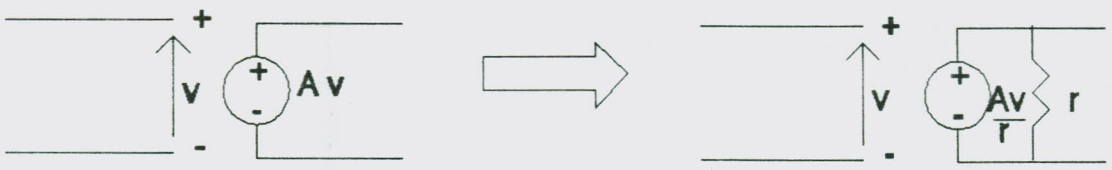


Fig. 2.19: Transformation of a VCVS.

The advantages of the nodal formulation method include its simplicity and familiarity, and avoidance of tree selection. Also, determination of network functions requires only partial solution of the *basic node voltage equation*, a very useful fact in network optimization.

2.2.6 Mesh Formulation Method

The mesh method is applicable only to planar graphs[17: p.68], graphs that can be mapped onto a plane such that no two edges have a point in common that is not a vertex[11: p.40]. Selection of a network tree is not required. The method is the dual of the nodal formulation method and its development proceeds in the same way as in the nodal method.

2.2.7 Cut-set Method

This is the general case of the nodal formulation method. But, unlike the nodal method, the n-1 tree branch voltages are the unknowns here and form a basis in terms of which all other voltages are described.

Let D^T take the role of A^T in equation (2.11) above. Then,

$$D^T i(t) = D^T Y v(t) - D^T I(t). \quad (2.14)$$

Pre-multiplying both sides of equation (2.10) by D^T gives

$$D^T i(t) = D^T C i_m(t). \quad (2.15)$$

Now, from equation (2.5), $D^T C = 0$. Hence,

$$D^T i(t) = 0. \quad (2.16)$$

Thus, substituting (2.16) in (2.14),

$$D^T Y v(t) = D^T I(t). \quad (2.17)$$

Since tree branch voltages form a basis set, $v(t) = D v_T(t)$. And, letting $D^T I(t) = J_T$,

$$D^T Y D v_T(t) = J_T. \quad (2.18)$$

The product $D^T Y D$ is identified as the cut-set admittance matrix. Denoting it by Y_T, the *basic cut-set equation* is obtained:

$$Y_T v_T = J_T. \quad (2.19)$$

As with the nodal formulation method, the success of the cut-set method also depends on the existence of voltage-defined current controlled branch relations.

2.2.8 Tie-set Method

The tie-set method is the dual of the cut-set method and a general case of the mesh formulation method. It is based upon the fact that loop currents or, more precisely, link currents, form a basis for all other branch currents. The success of the method depends on the existence of current-controlled voltage defined branch relations. Again, if a device does not possess these forms, then necessary transformations must be effected.

From Fig. 2.17, for the r-th branch and with $Z_r = 1/Y_r$,

$$v_r = v_e - E_r = Z_r i_e - E_r \quad (2.20)$$

And, for the whole network,

$$v(t) = Z i(t) - E. \quad (2.21)$$

Pre-multiplying the RHS and LHS of the last result by C^T, then using equations (2.7) and (2.10), letting $C^T E = E_m$ (the loop voltage-source vector), and denoting $C^T Z C$ by Z_m (the loop impedance matrix), one gets the *basic tie-set equation*:

$$Z_m i_m = E_m. \quad (2.22)$$

2.2.9 Hybrid Method

Resistors, inductors, capacitors, and gyrators have both impedance and admittance representations. VCCSes, on the other hand, have only admittance while the CCVSes have only impedance representations. Transformers, NCs, VCVSes, and CCCSes have only a mixed, or hybrid, representation. Any network equation formulation method must therefore consider these facts to accommodate a wide class of electrical networks.

The hybrid method of equation formulation starts from the premise that "... Ohm's law may be expressed in a hybrid form involving the admittance matrix for certain branches of the network and the impedance matrix for the rest of the branches" [2: p.84]. The admittance branches are studied by the cut-set method and the impedance branches by the tie-set method. The practice therefore requires selection of a network tree which is considered to be the "critical part of the formulation algorithm" [22: p.74] since certain precautions are imposed upon the tree selection process. Independent voltage sources, for instance, cannot be tree links, and independent current sources cannot be tree branches.

Once cut-set and tie-set methods have been applied to the entire network, the resulting hybrid equations are then coupled together using a suitable topological matrix [2: pp.83-86].

2.2.10 Nodal Method with Voltage Sources

In sections 2.2.10 - 2.2.14, we look at some of the methods that are used to retain the advantages of the nodal formulation method while removing its limitations.

In the *nodal method with voltage sources*, a provision for grounded voltage sources is added to the nodal method by partitioning the *basic node voltage equation* into *source* and *reduced equations* [22: pp.62-64]. Nodes are also renumbered to identify the datum node, nodes that are connected to grounded voltage sources, and the remaining nodes. The result is the partitioned set of equations:

$$\begin{bmatrix} I_v \\ \hline 0 \end{bmatrix} + \left[\begin{array}{c|c} Y_{ss} & Y_{sr} \\ \hline Y_{rs} & Y_{rr} \end{array}\right] \begin{bmatrix} V_s \\ \hline V_s \end{bmatrix} = \begin{bmatrix} J_s \\ \hline J_r \end{bmatrix} \tag{2.23}$$

where I_v is a vector of voltage-source branch currents, the vector v_s contains the grounded voltage sources, v_r is the unknown set of reduced node voltages, and the vectors J_s and J_r are

similarly defined.

The unknowns in equation (2.23) above are the entries in v_r and I_v and these are determined by first partitioning equation (2.23) such that v_s is known. Then,

$$v_r = y_{rr}^{-1} [J_r - y_{rs} v_s]$$
$$I_v = J_s - y_{ss} v_s - y_{sr} v_r \qquad (2.24)$$

2.2.11 Modified Nodal Method of Ho et al

First developed by C.W. Ho, A.E. Ruehli, and P.A. Brennan[23], the modified nodal method of Ho et al accommodates the inclusion of voltage-defined branches and current-controlled branches in "a much easier fashion"[22: p.64].

The principle of the method is this:

- number first all non-grounded nodes and use zero for ground;
- place all elements which have admittance description into an nxn matrix the same way as in the nodal formulation;
- for all other elements add rows and columns according to a simple stamp:
- the additional row expresses the constitutive equation of the element;
- the additional column takes care of the currents modifying KCL at the respective nodes.

As an example, consider the network of Fig. 2.20 next.

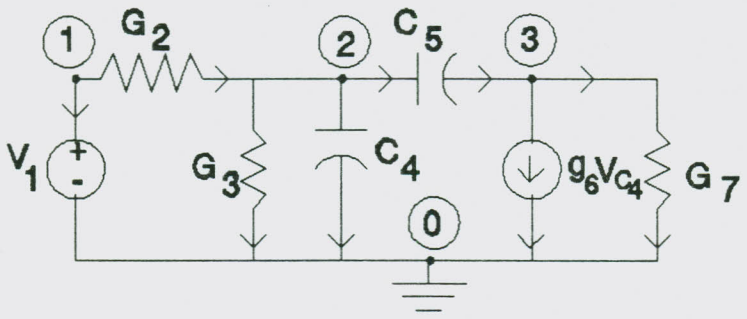


Fig. 2.20: A network to illustrate the Ho et al method.

To include the independent voltage source of Fig. 2.20 in the formulation method of Ho et al, note first that $v_1 = E$, where E, in general, is a voltage-defined branch voltage vector. Note also that there is a current through branch 1 and that modifies KCL at node 1, the positive node of branch 1. Hence, the equation formulation takes the form of equation (2.25) next:

$$\left[\begin{array}{ccc|c} G_2 & -G_2 & 0 & 1 \\ -G_2 & G_2+G_3+SC_4+SC_5 & -SC_5 & 0 \\ 0 & g_6-SC_5 & SC_5+G_7 & 0 \\ \hline 1 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ \hline I \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ 0 \\ \hline E \end{array}\right]$$

...(2.25)

The general form of the modified nodal formulation method is:

$$\left[\begin{array}{c|c} Y_n & B \\ \hline C & D \end{array}\right] \left[\begin{array}{c} V_n \\ \hline I_v \end{array}\right] = \left[\begin{array}{c} J_n \\ \hline E \end{array}\right] \qquad (2.26)$$

where:

Y_n , V_n, J_n are quantities as indicated in section (2.2.5);

I_v = vector of voltage-defined branch currents;

B, C, D are sub-matrices that account for the necessary modification in the nodal formulation method.

In the particular case when there are no voltage-defined or current-controlled branch relations, equation (2.26) reduces to the *basic node voltage* relation of equation (2.13).

2.2.12 The Reduced Modified Nodal Approach

The unknowns in the modified nodal method of Ho et al are the n-1 non-datum node voltages and the v voltage-defined branch currents. These make a total of n-1+v unknowns. The matrix associated with the method is therefore larger in size than that in the nodal formulation method. Two other problems with the modified nodal method are the existence of one or more zero-valued diagonal entries in the associated matrix and "...the necessity of formulating the branch relations prior to adopting a branch current as a network variable".

To overcome the above difficulties, Lee and Park have suggested a new method that they call the reduced modified nodal approach (RMNA) [24].

The network elements allowed in the modified nodal method and the RMNA are the same. The latter, however, formulates the equations in terms of only node voltages and

controlling currents thus resulting in a smaller matrix size. It is to be recalled that in the modified nodal method, in addition to the nondatum node voltages, the voltage source currents and the controlling currents are chosen as additional network variables.

The RMNA also uses three types of pivotings (namely, node renumbering, row interchange and column interchange) and these are reported to always guarantee non-zero diagonal entries. There is no need of pivoting once the RMNA equations are formulated. This is not the case with the modified nodal approach and pivoting may be called upon even after formulating the relevant network equations.

2.2.13 The Sparse-Tableau Method

The concept of the tableau includes all network information in non-reduced form and, instead of obtaining a more compact set of equations, the complete set is solved [25]. There are four advantages claimed out of this act: 1)generality, 2)simplicity, 3)speed, and 4)problem size.

The claim of simplicity arises from the use Gaussian elimination in the main computational task. The approach is considered general since "any system of differential and algebraic equations can be handled". The operations executed are the necessary ones only and this adds to the speed of the method. Large problem sizes can also be handled since storage is saved by not storing 0's and 1's.

The tableau definition requires the introduction of what are known as the *unknown vector*, ω, the *tableau operator*, F, and the tableau *error vector*, $f(\omega,t)$.

The unknown vector ω is defined as:

$$\omega = \mathbf{col(V,i,v,q,\Phi,p)} \tag{2.27}$$

where V = node voltage vector;

i = branch current vector;

v = branch vector;

q = generic name to denote energy storage;

Φ = magnetic flux; and,

p = electronic charge.

The operator F is an algebraic operator which operates to the right on ω to form $f(\omega,t)$. Thus,

$$F(.,.,.,.,.,t)\ [V \quad i \quad v \quad q \quad \Phi \quad p]^T = f(V,i,v,q,\Phi,p,t) \tag{2.28}$$

The network equations may now be stated as:

$$f(\omega,t) = 0 \tag{2.29}$$

and are determined using equations (2.6) and (2.9), the branch constitutive relations, and the dynamic relations describing the generalized energy storage elements.

On the other hand, linearization of the tableau operator F around an operating point ω gives the *tableau matrix* F_L, i.e.,

$$F_L(\omega,t) = \frac{\partial f(\omega,t)}{\partial \omega} \tag{2.30}$$

This allows the alternative tableau definition:

$$F_L(\omega,t)[\omega] = 0 \tag{2.31}$$

The tableau matrix F_L, in general, has the form in Table 2.2.

Table 2.2: General Form of the Tableau Matrix.

0	A^T	0	0
A	0	-I	0
0	Z	Y	B
0	E		-d/dt

It is to be noted in Table 2.2 that the sub-matrices A, B, and E all have elements ± 1 or 0, and I is the identity matrix.

2.2.14 The Modified-Tableau Method [26]

The unknowns in the original tableau formulation were seen to be the voltage across each element, currents through each element, node voltages, capacitor charges, and inductor fluxes. The associated matrix was obviously very large and very sparse.

The modified-tableau method formulates the network equations in terms of branch voltages and currents only by combining topological concepts with a modification of the tableau

formulation of Hachtel et al. The resulting associated matrix has a diminished size when compared to its sparse-tableau counterpart.

The cut-set and tie-set methods, in conjunction with the branch relations, are used to obtain the network equations after selecting a network tree. The tree selection itself is done by forcing all independent voltage sources to be tree branches and all independent current sources to be tree links; the remaining tree branches are chosen in a manner designed to minimize the number of non-zero coefficients in the cutset matrix.

2.2.15 Summary

In section 2.2 so far, systematic methods for applying the fundamental laws of electrical network theory have been examined. These methods yield simultaneous equations useful in both analysis and design. Although inspection, Thevenin's Theorem and other shortcuts may be applied for networks of moderate size, the value of the methods of section 2.2 lies in their generality and ease for computer programming.

The topological methods of equation formulation are based upon the determination of basis voltages or currents and yield reduced sets of equations. These advantages, however, must be weighted against the capacity to entertain a wide range of electrical networks. The sparse-tableau and modified tableau approaches, on the other hand, provide general formulation methods at the expense of larger matrix sizes.

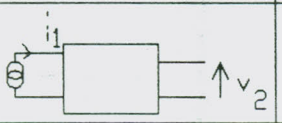
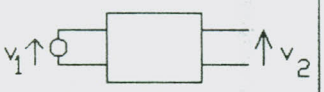
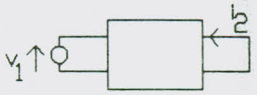
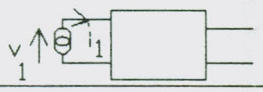
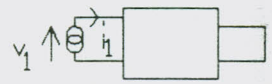
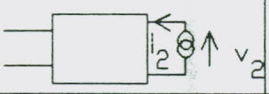
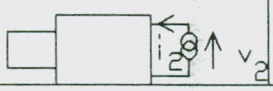
The optimal design of electrical networks in the frequency domain is a procedure which calls for the extraction of network functions for eventual comparisons with desired responses. In the nodal formulation method, network functions are obtained as partial solutions of the *basic node voltage equation* and this approach has some peculiar advantages such as avoidance of the usual cancellations inherent in the evaluation of determinants. Development of the theory in terms of mesh-current equations could be taken but this has not received much use in the literature since it needs the determination of the chords of a tree, an extra step beyond the determination of the tree branches that is required for the nodal approach. For these reasons, the nodal formulation method shall be used throughout this thesis.

2.3 Simplified Use of Topological Formulas

2.3.1 General Formulation

The name *Topological Formulas* refers to a class of formulas used to derive network functions by inspection from the network without actually expanding the various determinants and cofactors. Determination of these functions involves evaluation of the NAM Δ and its cofactors Δ_{jj} and Δ_{ij} as shown in Table 2.3 next [17: p.283]:

Table 2.3: Network Functions and their NAM solutions.

Network	Network Function	Definition	NAM Solution
	Open Circuit Transfer Impedance	$\left(\frac{V_2}{I_1}\right)_{I_2=0}$	$\frac{\Delta_{1n}}{\Delta}$
	Open Circuit Voltage Gain	$\left(\frac{V_2}{V_1}\right)_{I_2=0}$	$\frac{\Delta_{1n}}{\Delta_{11}}$
	Short Circuit Current Gain	$\left(\frac{I_2}{I_1}\right)_{V_2=0}$	$\frac{-\Delta_{1n}}{\Delta_{nn}}$
	Short Circuit Transfer Admittance	$\left(\frac{I_2}{V_1}\right)_{V_2=0}$	$\frac{-\Delta_{1n}}{\Delta^{2,n-1}_{2,n-1}}$
	Open Circuit Input Admittance	$\left(\frac{I_1}{V_1}\right)_{I_2=0}$	$\frac{\Delta}{\Delta_{11}}$
	Short Circuit Input Admittance	$\left(\frac{I_1}{V_1}\right)_{V_2=0}$	$\frac{\Delta_{nn}}{\Delta^{2,n-1}_{2\ n-1}}$
	Open Circuit Output Admittance	$\left(\frac{I_2}{V_2}\right)_{I_1=0}$	$\frac{\Delta}{\Delta_{nn}}$
	Short Circuit Output Admittance	$\left(\frac{I_2}{V_2}\right)_{V_1=0}$	$\frac{\Delta_{11}}{\Delta^{2,n-1}_{2\ n-1}}$

where Δ = det(Y_n), Δ_{ij} = the i-jth cofactor of Δ, $\Delta^{2,n-1}_{2,n-1}$ = det(matrix of rows and columns 2,3,...,n-1, of Y_n).

A useful theorem in this process is *the Binet-Cauchy Theorem*:

Let P be of order (m,n) and Q of order (n,m) be matrices of elements. Let m < n. Then,

det PQ = Σ(Product terms of corresp. majors of P and Q), where the summation is over all such majors.

A *major* is the determinant of the largest square submatrix, in this case of order m. The phrase *corresponding major* implies that if columns $j_1, j_2, \ldots, j_m$ of P are chosen for a major, rows $j_1, j_2, \ldots, j_m$ of Q should be chosen to form the corresponding major.

Consider next *tree-admittance product* and *2-tree admittance product*. These refer, respectively, to admittance products of tree and 2-tree branches. The product of an isolated vertex in a 2-tree is defined as 1. A 2-tree $T_{2\,i,i}$ where the same vertex i is in different connected parts is defined to have a zero product.

In the *traditional* methods of determinant evaluation, such as *cofactor expansion*, a network function is found after so many calculations and there is cancellation of terms. When using topological formulas, however, there is no subtraction of terms and every term evaluated appears in the final result. For this reason, the latter are also called *minimum-effort* formulas.

Evaluations of Δ and its cofactors shall next be studied by first assuming passivity and no mutual inductances. These assumptions shall soon be dropped for the general case.

2.3.2 Case I: Evaluation of Δ

The determinant of the NAM can be set as

$$\Delta = \det Y_n = \det(A^T Y A)$$
$$= \sum (\text{products of corresponding majors of } A^TY \text{ and } A).$$

Since $Y_n = \text{diag}[y_j]$, $j = 1, 2, \ldots, b$, A^TY and A^T are identical except that each column j of A^TY is multiplied by y_j. By the first two properties in **Appendix C**, the non-zero majors of A^T correspond to trees and have values ± 1. So, the non-zero majors of A^TY correspond to trees and have values $(\pm 1)(y_{b1}y_{b2}\ldots y_{bk})$, where $k = n$ and y_{bk} is the k-th tree branch admittance. The corresponding majors of A have also values ± 1. All other non-tree majors are zero. Thus,

$$\Delta = \sum_{\substack{all \\ trees}} (\textit{tree admittance products}) \tag{2.32}$$

Equation (2.32) is called Maxwell's Rule or Formula.

2.3.3 Case II: Evaluation of Δ_{ij}

Δ_{jj} is the cofactor of an entry of Y_n on the main diagonal. It is obtained by deleting the

i-th row and i-th column of Y_n and then taking the determinant of the resultant matrix. The same may be obtained if one omits row j of A^T and column j of A from A^TYA. Letting A_{-j} to be the RIM with row j deleted, we have:

$$\Delta_{jj} = |A_{-j}{}^T Y A_{-j}|$$
$$= \sum_{trees\ of\ N_{-j}} (\text{tree admittance products of } N_{-j})$$

where N_{-j} is the network obtained from the original network N by deleting row j of A^T and column j of A or, what is the same thing, by short-circuiting node j to the reference node r.

Network N_{-j} has one less node(hence one less tree branch in a tree) than N, i.e., trees of N_{-j} cannot be trees of N. Also, since N_{-j} cannot contain a loop, nodes j and r must be in two separate parts. Consequently, we have:

$$\Delta_{jj} = \sum_{\substack{all \\ 2-trees}} (T_{2\,j,r}\ admittance\ products) \tag{2.33}$$

2.3.4 Case III: Evaluation of Δ_{ij}

To form Δ_{ij}, delete row i and column j of Y_n. This yields

$$\Delta_{ij} = \det(A_{-i}{}^T Y A_{-j})$$
$$= \sum (\text{products of corresponding majors of } A_{-i}{}^T Y \text{ and } A_{-j}).$$

The non-zero majors of $A_{-i}{}^TY$ correspond to 2-trees (i,r) while those for A_{-j} correspond to 2-trees(j,r). The only terms that contribute to the cofactor Δ_{ij} must be the common 2-trees of (i,r) and (j,r). Moreover, since a 2-tree has only two separate parts, both i and j must be in one part and r in the other. It therefore follows that

$$\Delta_{ij} = \sum_{\substack{all \\ 2-trees}} (T_{2\,ij,r}\ admittance\ products) \tag{2.34}$$

Note that the formula for the asymmetrical cofactor(Δ_{ij}) contains the formula for the symmetrical cofactor(Δ_{ii}) as a special case. This is established by letting i=j in equation (2.34) above.

2.3.5 Case IV: The General Case

The discussions for the evaluation of the NAM determinant and its various cofactors may be generalized using the Coates - Mayeda technique [11: pp.181-193]. The central theme of this technique is to *modify* the network graph such that NAM can be written as

$$Y_n = A_I^T \, Y \, A_V \tag{2.35}$$

where

Y = branch admittance matrix of the new graph and *is diagonal*;

A_I = the incidence matrix of the current graph;

A_V = the incidence matrix of the voltage graph; and,

Y_n is the same as the NAM of the original graph and A_I and A_V are taken with *the same reference vertex*.

It is also useful to distinguish between three types of edges: *ordinary edges*, *current edges* and *voltage edges*. The term ordinary edges refers to passive branches while the latter two respectively refer to current receiving branches and voltage sending branches.

We shall now define *current-*, *voltage-* and *complete-graphs* of an electrical network. A network graph containing all ordinary and current edges is called a *current* or *i-graph*. A *voltage* or *v-graph*, on the other hand, is one that is constructed from ordinary edges and voltage edges. The current and voltage graphs together form *the complete-graph* of a network. For ordinary edges, the v-i graphs are identical and so rows of A_I and A_V are identical. For current and voltage edges, if row k of A_I corresponds to a current element, then row k of A_V corresponds to a voltage element.

The notion of a *complete-graph* gives rise to the notions of *complete-tree* and *complete-tree admittance product*. The former is a set of edges with weights $y_{b1}, y_{b2}, \ldots, y_{bk}$ that constitute trees common to both the v-i graphs [11: p.85] while the latter refers to the product $y_{b1}y_{b2}\ldots y_{bk}$ of the admittances of a complete-tree. A complete-tree may also be called a *common-tree*.

Considering the evaluation of Δ for the general case:

$\Delta = \det(A_I^T \, Y \, A_V)$

$= \sum$ (products of corresponding majors of $A_I^T Y$ and A_V).

Matrices A_I and A_V possess all the familiar properties of RIMs. Their nonsingular submatrices, in particular, correspond to trees and have determinants ± 1 and so a major of $A_I^T Y$ is non-singular if and only if rows of A_I correspond to a tree of the i-graph. Then, $A_I^T Y$ will have value $(\pm 1)(y_{b1}, y_{b2}, \ldots, y_{bk})$ where k = n-1 and b1, b2, ..., bk are the branches of the tree. The corresponding major A_V has the value ± 1 if the rows of A_V^T correspond to a tree of the v-graph. The determinant of Y_n, by the Binet-Cauchy Theorem, is made of *common-* or *complete-*

trees of the v-i graphs:

$$\Delta = \sum_{\substack{all \\ complete \\ trees}} (\pm 1)(\textit{complete-tree admittance products}) \tag{2.36}$$

Evaluation of the cofactor Δ_{ij} of the NAM determinant is achieved similarly by first defining what are known as *complete (common) 2-trees* and *complete 2-tree admittance products*. The edges y_{b1}, y_{b2}, ..., y_{bk}, where k = n-2 and bk indicates the k-th branch, constitute a *complete 2-tree* $T_{2\,j,r}^{i,r}$ of the graph if the current-edges with these weights constitute a 2-tree separating nodes i and r of the i-graph, and voltage-edges with these weights constitute a 2-tree separating nodes j and r of the v-graph. The product $y_{b1}y_{b2}...y_{bk}$ of a complete 2-tree is a *complete 2-tree admittance product*.

Now, to evaluate Δ_{ij}, delete row i from A_I^T and column j from A_V. Denoting the resultant RIMs by $A_{I_{-i}}^T$ and $A_{V_{-j}}$,

$$\Delta_{ij} = \det(A_{I_{-i}}^T \, Y \, A_{V_{-j}})$$

Since nonsingular sub-matrices of $A_{I_{-i}}^T$ and $A_{V_{-j}}$ correspond one-to-one to the 2-trees of the v-i graphs, the contributing terms to Δ_{ij} must be complete 2-trees of these graphs. Hence,

$$\Delta_{ij} = \sum_{\substack{all \\ complete \\ 2\text{-}trees}} (\pm 1)(\textit{complete 2-tree admittance products}) \tag{2.37}$$

In the implementation of the above results for passive networks, the original network N, the networks N_{-i} and N_{-j} with nodes i and j shorted to the datum node are first formed. Δ is then determined from trees generated for N, Δ_{ii} from trees generated for N_{-i}, and Δ_{ij} from common-trees generated for N_{-i} and N_{-j}. Active networks are similarly treated by forming the networks N_I and N_V and their shorted-node counterparts. Δ is then found from common trees of N_I and N_V; Δ_{ii} from common trees of $N_{I_{-i}}$ and $N_{V_{-i}}$; and Δ_{ij} from common 2-trees of $N_{I_{-i}}$ and $N_{V_{-j}}$.

Also, for passive networks,

$$N_I \equiv N_V \equiv N,$$

$$N_{I_{-i}} \equiv N_{V_{-i}} \equiv N_{-i}, \text{ and}$$

$$N_{V_{-j}} \equiv N_{-j}.$$

2.4 The Tree Generation Problem

As shown so far, the use of topological formulas requires tree, 2-tree, complete-tree, and complete 2-tree admittance products and this takes us to the tree generation problem. More specifically, the tree generation problem is concerned with the generation of non-duplicating network trees, the rapid test of edge sets, and the avoidance of edge sets that predictably form loops. The solution of these problems, together with tests for obtaining numerator and denominator polynomials of a network function is the main aspect of any efficient topological method.

By property C4 of **Appendix C**, the number of network trees is equal to $\det(A\ A^T)$ and a total of that many must be generated for any given network to evaluate the NAM determinant Δ. The evaluations of the cofactors Δ_{ii} and Δ_{ij} of Δ also require the generation of trees whose number is determined from the determinant of the corresponding submatrices.

In this thesis, we shall use the *Cyclic Tree Generation Algorithm* of P.R. Adby [17: p.299]:

> **STEP 1**: List the branches connected to each node, ensuring that the set of branches at the top of the list for each node is unique and does not contain parallel branches.
>
> **STEP 2**: Cyclically step down the list from the previously selected branch set changing node N most rapidly and node 1 least rapidly.
>
> **STEP 3**: If a duplicate or parallel branch is introduced, continue cyclic generation at the highest numbered node for that branch until a selection is obtained.

Thus, to generate trees, one prepares the *branch-node list* and forms all possible edge combinations. Each edge-set with no repeating edge is accepted as a *possible tree* while the others are rejected. When this process is over, all *possible trees* are checked for loop formation and those that do not form loops become *network trees*.

As an example, consider evaluation of trees for Fig. 2.6 on page 10:

The branch-node list is:

	Nodes		
Branches	1	1	2
	4	2	4
		3	

As can be seen from the branch-node list, there are two branches connected to each of the first and last nodes while this number is three for the second node.

Tree generation follows in the next manner:

$$A\,A^T = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & -1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{bmatrix}$$

$$\text{and hence } \left| A\,A^T \right| = \begin{vmatrix} 2 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 2 \end{vmatrix} = 3.$$

i.e., there are 3 trees for the indicated network.

Edge-set	**Remarks**
{ 1,1,2 }	x, edge is duplicated.
{ 1,1,4 }	x, edge is duplicated.
{ 1,2,2 }	x, edge is duplicated.
{ 1,2,4 }	✓, possible tree.
{ 1,3,2 }	✓,possible tree.
{ 1,3,4 }	✓, possible tree.
{ 4,1,2 }	✓, possible tree.
{ 4,1,4 }	x, edge is duplicated.
{ 4,2,2 }	x, edge is duplicated.
{ 4,2,4 }	x, edge is duplicated.
{ 4,3,2 }	✓, possible tree.
{ 4,3,4 }	x, edge is duplicated.

Note: x = reject; ✓ = accept.

That makes a total of 12 selections of the branch-node list.

A formula for the number of selections may now be devised:

Let the total number of edges connected to a particular node i be max(i). Then, the total number of selections S obtainable from the branch-node list of n nodes is given by:

$$S=\prod_{i=1}^{n} \max(i) \tag{2.38}$$

In the present case, S = max(1) x max(2) x max(3) = 12, in harmony with the result above. Seven selections contain duplicated edges. The five possible trees are therefore

{1,2,4}
{1,3,2}
{1,3,4}
{4,1,2}
{4,3,2}.

Edge-sets {1,2,4} and {4,1,2} represent the same edge and must be rejected from the list as they correspond to a loop. That leaves a total of only three trees for the network as ascertained earlier by the value of $\det(AA_T)$.

The three trees are:

{1,3,2}
{1,3,4}
{4,3,2}.

The NAM determinant is then

$$\Delta = y_1 y_3 y_2 + y_1 y_3 y_4 + y_4 y_3 y_2$$

$$= \frac{1}{sL_1} + sC_1 + \frac{1}{sL_2} + \frac{1}{sL_1R_1}sC_1 + \frac{sC_1}{sL_2}$$

$$= \frac{sC_1R_1 + S^2C_1L_2 + s^2C_1L_1}{s^2L_1L_2R_1}$$

The various cofactors of the NAM are evaluated analogously by considering common-trees and common 2-trees of the relevant networks.

The tree testing procedure may also be facilitated by means of *Grassman's Algebra*. This contribution has been made by P.R. Adby in a tutorial paper [18]. The author, while preserving the essentials of his *cyclic tree generation algorithm* in [17], expresses that algorithm in terms of the outer product using *Grassman's Algebra* and demonstrates the uniqueness of the tree generation. By applying simplification to the evaluation of the outer products the author then comes out with a useful *tree test algorithm* the reproduction of which shall be deferred until the next brief introduction on the use of Grassman's Algebra for the evaluation of determinants.

The procedure is summarized in four steps [18]:

STEP 1: For each column, list the non-zero terms as vectors of row numbers and form the corresponding outer product;
STEP 2: Assign signs to the outer product terms as $(-1)^q$ where q is the number of transpositions of adjacent elements to put the terms in natural order.
STEP 3: Translate outer product terms to determinant elements.
STEP 4: Insert numbers and evaluate.

In Adby's words, "The evaluation of determinants using Grassman's Algebra is based on the summation of all possible combinations of products taken one from each column of a matrix and one from each row. Grassman's Algebra formalizes this summation by organizing the products while omitting all zero terms."[18].

As an example, consider the next matrix.

$$\begin{bmatrix} 1 & 0 & 6 & 2 \\ 0 & 0 & 4 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 3 & 2 & 0 \end{bmatrix}$$

Following the above four steps, we have:

STEP 1. *Outer Product Evaluation.*

Outer product = {1,3}^{3,4}^{1,2,4}^{1}
= {13,14,33,34}^{11,21,41}
= {13,14,34}^{21,41}
= {1321,1341,1421,1441,3421, 3441}
= {3421};

where:

- the caret(^) signifies that outer products are taken two sets at a time.
- the outer[*caret*] product is defined as $u \hat{} v = u\ v^T$ and read as *u caret v*.
- numbers show row numbers of non-zero entries for each column;
- all *outer product terms* with repeated row numbers are omitted.
- the set {3421} is an *outer product term* and each entry in it is a *term*. The outer product in this case is made up of only one *outer product term*.

STEP 2. Sign Assignments.

Outer Product	Transpositions	q	Result
3421	3241 → 3214 → 1234	3	-3421;

where:

q = minimum value of the number of transpositions;

Result = outer product term with associated sign.

STEP 3. Translation of Terms into Determinant Elements.

Result	Determinant Elements
-3421	$-a_{31}.a_{42}.a_{23}.a_{14}$

where:

row numbers come from the corresponding outer product term, and column numbers come from position of term in a the outer product.

STEP 4. Evaluation of Determinant.

$$\text{determinant} = -\, a_{31}.a_{42}.a_{23}.a_{14}$$
$$= -\, 1 \times 3 \times 4 \times 2$$
$$= -\, 24.$$

This value may be checked in any convenient manner.

Evaluation of a determinant using Grassman's Algebra when there are two or more outer product terms is done by following steps 1-3 outlined above and finally summing all determinant-element-products finally at step 4. The advantage of the approach when large systems are to be handled is that it safely avoids the zero valued entries.

Application of Grassman's Algebra to the incidence matrix indicates that the outer product terms are $\pm$ 1. The determinants of majors of the incidence matrix, on the other hand, are $\pm$ 1 or 0. As a result [18]:

> ... outer products containing an even number of terms, or no terms, must cancel to zero.
>
> Conversely, if the outer product contains an even number of terms the determinant must be +1 or -1, and the major corresponds to a network tree.
>
> Furthermore, the number of terms in the outer product for a tree must be one since it is always possible to renumber the nodes and branches of a tree such that its major is always a lower triangular matrix. Alternatively, any major of the incidence matrix corresponding to a tree can be put into lower triangular form by

row and column interchanges.

So, in accordance with the above observations, majors corresponding to non-branch sets are either not generated or generated an even number of times with signs such that the branch admittance products cancel. This, however, does not rule out the need for a tree testing algorithm: there are cases where common trees of different networks or different parts of an active network must be selected.

Consider again the bridged-T network of Fig. 2.2. The outer product including all rows of the reduced incidence matrix is:

$$\begin{aligned}\text{Outer Product} &= \{1,4\}\hat{}\{1,2,3\}\hat{}\{2,4\}\\ &= \{11,12,13,41,42,43\}\hat{}\{2,4\}\\ &= \{12,13,41,42,43\}\hat{}\{2,4\}\\ &= \{122,124,132,134,412,414,422,424,432,434\}\\ &= \{124,132,134,412,432\}\end{aligned}$$

As a passing remark, the *outer product* here is made up of five *outer product terms* each of which consists of three *terms*. Also, what we are referring here as outer product terms are what were earlier known as edge-sets.

Signs may also be included by noting that q takes the values 0, 1, 0, 2, and 3, respectively, for the indicated outer product terms. The result, inserting signs from $(-1)^q$, is therefore:

$$\{124,-132,134,412,-432\}.$$

The outer product terms {132}, {134}, and {432} are not duplicated and so correspond to tree branches. The outer product terms {124} and {412} represent branches that form loops and they must cancel.

Tree testing using Grassman's Algebra is based upon the formation of outer products with less redundant terms and this is done through cancellation. For the bridged-T network, for instance, the set {132} has been shown to correspond to a network tree. We now show, using Grassman's Algebra, that this is certainly correct.

Accordingly, the outer product was shown to be {1,4}^{1,2,3}^{2,4}. Now, if we cancel edge 4 from the vectors appearing in the outer product, the result is {1}^{1,2,3}^{2}. Since the first and third vectors are single valued, their entries cannot be used again and must be canceled from the second vector. This gives {1}^{3}^{2} = {132}, as required.

The *tree test algorithm* may now be stated:

> "*...vectors can be canceled to a single term each if the branch set forms a tree*" [18].

3. On the Theory of Network Optimization

3.1 Problem Formulation

The design of electrical networks on the computer requires transformation of the original design problem into an equivalent *mathematical optimization problem*. This is done by first describing the network behavior in terms of a function that embodies the design criteria and compares the network characteristics with these criteria. If the results so obtained fail to satisfy the desired specifications, the design parameters are changed iteratively and in a continuous manner till the optimum performance is attained. The net effect parallels the bread-board design practice. To formulate the design problem in mathematical terms, let:

F = a desired response;

F^j = the response after j adjustments or iterations;

$\mathfrak{w} = [\mathfrak{w}_1\ \mathfrak{w}_2\ \ldots\ \mathfrak{w}_n]^T$ = vector of response sample points;

$c = [c_1\ c_2\ \ldots\ c_n]^T$ = parameter vector of n components;

$c^j = [c_1^j\ c_2^j\ \ldots\ c_n^j]^T$ = value of c in the j-th iteration;

Then, the computer-aided optimal design of electrical networks can be formulated as a problem of minimizing an error criterion E, which is the value of a function of the n system components, by adjusting these components, i.e.,

$$\text{minimize } E = f(c_1, c_2, \ldots, c_n). \tag{3.1}$$

The error criterion E is a scalar function and is variously known as *objective function*, *cost function*, *performance function*, *performance index* or *optimality index*. It measures the error between the desired network response F and the actual network response F^j after j adjustments or iterations. This is why the object of a computer-aided design program may be stated as one of adjusting the network elements to "... reduce, that is, minimize, the value of the performance function" [28: p.623].

The function E may be obtained from a *weighted error function* $\epsilon(\mathfrak{w},c)$ in several ways as shall be undertaken in section 3.3 below. The error function ϵ is defined as

$$\epsilon(\mathfrak{w},c^j) = w(\mathfrak{w})\,[F(\mathfrak{w}) - F^j(\mathfrak{w},c^j)], \tag{3.2}$$

where $w(\mathfrak{w}) \geq 0$ is a *weighing* or *penalty function* that depends on the response sample points $\mathfrak{w}$. It is used to emphasize or deemphasize the difference between F and F^j at selected values of the latter.

Equation (3.2) describes the performance of the network at any single response sample point. This may be either frequency or time or both, although the concern in this thesis shall be on the former only.

E^j indicates the performance of the network in the j-th iteration. For convenience, the iteration counter j shall sometimes be dropped while a subscript attached to E shall indicate the type of error criterion used. Also, the notations $E^j(\omega^j,c_i^j)$ and $F^j(\omega^j,c_i^j)$ shall be used to indicate the dependencies of E^j and F^j on ω^j and the n adjustable components c_i^j, i=1,2,...,n.

The usual definition of the performance function is such that $E \geq 0$ and $E = 0$ indicates a zero error, i.e., an exact match between the actual and desired responses. Its formulation may involve one or more of gain, phase, delay, bandwidth, input impedance, noise figure, dynamic range, or the like.

3.2 Sampling and its Usefulness

The definition of an error criterion for a computer-aided optimization work requires that m sample points ω_i and m weights w_i at these points be chosen. Sampling, in particular, is necessary since the optimization problem must be solved numerically using discrete numbers rather than continuous functions. Hence, all functions of ω must be sampled or discretized into m sample points.

The choice of the weighing factors w_i merits some care since this may determine not only the final minimum but also the rate of convergence by altering the contouring. Experience has shown that the best procedure is to choose the w_i so that the final tolerated weighted errors are approximately equal [44] .

Sampling is also necessary since it replaces all differentiations and integrations with respect to the response sample vector through the relations of equations (3.3) and (3.4) next:

$$\frac{df(\omega)}{d\omega} \rightarrow \frac{f(\omega_i) - f(\omega_{i-1})}{\omega_i - \omega_{i-1}} \tag{3.3}$$

$$\int_a^b F(\omega)\, d\omega \rightarrow \sum_{\omega_i=a}^{b} F(\omega_i) \tag{3.4}$$

The sample points, in most cases, are selected equally spaced in the sampling interval and the size of the number m is determined by the problem at hand. This number must neither be too large nor too small. For a rapidly varying F, for instance, m must be large enough but,

if it is too large, the sampling process will be slowed down. The sample points must also be chosen densely enough so that neither F nor F^j changes significantly between two adjacent samples. A rule of thumb is that at least 2n samples be employed [3, 43], where n is the number of parameters to be adjusted. It is usual to choose 5n sample points [27: p.453].

3.3 Error Criteria Used in Network Design

In order to express the error E introduced in section 3.1 above as a single quantity, we use the p-norm of the weighted error function $\epsilon(c,\mathfrak{w})$, where p is an integer. The p-norm, when $\mathfrak{w}$ is a continuous variable, is defined as:

$$\|\epsilon\|_p = \left(\int_{w1}^{w2} | \epsilon(c,w) |^p \, dw \right)^{\frac{1}{p}}, \qquad 1 \le p \le \infty \tag{3.5}$$

When $\mathfrak{w}$ takes discrete values only, equation (3.5) takes the form:

$$\| \epsilon \|_p = \sum_{i=1}^{m} |\epsilon_i(c,\mathfrak{w})^p|^{\frac{1}{p}}, \qquad 1 \le p \le \infty \tag{3.6}$$

The p-norm, just as any other vector norm, must satisfy the following three conditions for vector norms:

1. $\|\epsilon\| > 0$ for $\epsilon \neq 0$, and $\|\epsilon\| = 0 \Rightarrow x = 0$;
2. $\| h\epsilon \| = |h| \, \|\epsilon\|$ for any scalar h; and,
3. $\|x + y\| \le \|x\| + \|y\|$ (triangle inequality).

The three common values of p are 1, 2, and ∞, and the corresponding norms are called the one-, two-, and infinity-norms.

Equation (3.6) may be put in the equivalent form:

$$(\| \epsilon \|_p)^p = \sum_{i=1}^{m} |\epsilon_i(c,\mathfrak{w})^p|, \qquad 1 \le p \le \infty \tag{3.7}$$

Designating the LHS of Equation (3.7) by E_p and dropping c and $\mathfrak{w}$, we have:

$$E_p = \sum_{i=1}^{m} |\epsilon_i|^p \quad \text{(least p-th error criterion)} \tag{3.8}$$

Equation (3.8) is used in conjunction with the least p-th Taylor method of optimization to be discussed in section 3.6 below. The minimum obtained using the least p-th error criterion is the best approximation for the p-th norm of the error function. Also, to avoid mutual cancellation of positive and negative errors, even values of p are used in equation (3.8).

A value of p = 2 in equation (3.8) leads to the commonly used least-squares error criterion:

$$E_2 = \sum_{i=1}^{m} e_i^2 \quad \text{(least squares error criterion)} \tag{3.9}$$

Allowing p to be infinitely large, on the other hand, leads to direct minimax formulation of the error criterion:

$$E_\infty = \max_{i=1}^{m} |e_i| \quad \text{(min-max or Chebychev error criterion)} \tag{3.10}$$

The min-max or Chebychev error criterion suggests the minimization of the largest absolute error in the frequency range of interest. The criterion is not without problems, however. This is because of the fact that the error E jumps discontinuously as one value of i changes to another in the optimization process, thus rendering the derivatives of E with respect to the design parameters undefined. The least squares error criterion usually provides a better behaved function of the parameter vector c than the Chebychev [3: p.1834].

Another useful error criterion is the *maximally flat error criterion*:

$$E_{MF} = \text{(order of lowest non-zero error derivative at } \omega_0)^{-1}$$

$$= \lim_{\omega \to \omega_o} \frac{\ln|\omega - \omega_0|}{\ln|e|}$$

$$= \lim_{\omega \to \omega_0} \frac{\Delta x}{x - x_0} \quad \text{(maximally flat error criterion)} \tag{3.11}$$

In the maximally flat error criterion, agreement between F and F^j is sought by comparing these functions and their first n-1 derivatives for some $\omega = \omega_0$ [27: p.513].

Nearly all error criteria used in electrical network design fall into one of the four relations of equations (3.8)-(3.11) above.

3.4 The Concept of Sensitivity Analysis

Sensitivity is a measure of the change brought on a response function due to variations in its constituent parameters. Although a well-worn subject, the topic has received "new impetus" because of the general availability of the digital computer [33].

In network optimization, sensitivity provides the vital link between analysis and design by furnishing gradient information in a form of a Jacobian matrix of the performance function. This is necessary to arrive at the modified set of parameter values.

Network sensitivity may be *absolute*, *relative* or *semi-relative*. Absolute sensitivity is also called *unnormalized sensitivity* or, simply, *sensitivity* while the names *normalized* and *semi-normalized sensitivities* are used, respectively, in places of the latter two.

To define absolute network sensitivity in a general way, let f represent a network response (which may be frequency-, transient- or dc-response, or the like) such that $f = f(c_i)$, a function of the n components c_i, $i = 1, 2, \ldots, n$. Let also incremental changes Δc_i in the component values cause a corresponding incremental change Δf in the network response function.

The change Δf, when expanded into a Taylor series about the nominal value of f, becomes:

$$\Delta f = \sum_{i=1}^{n} \frac{\partial f}{\partial C_i}\Delta C_i + \frac{1}{2!}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 f}{\partial C_i \partial C_j}\Delta C_i \Delta C_j + \frac{1}{3!}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} \frac{\partial^3 f}{\partial C_i \partial C_j \partial C_k}\Delta C_i \Delta C_j \Delta C_k + \ldots \tag{3.12}$$

And, considering the first order terms only,

$$\Delta f = \sum_{i=1}^{n} \frac{\partial f}{\partial C_i}\,\Delta C_i \tag{3.13}$$

The absolute sensitivity of f to the component C_i is defined to be the partial derivative term appearing in equation (3.13) above. Thus,

$$S_{C_i}^{f} = \frac{\partial f}{\partial C_i} = \lim_{\Delta C_i \to 0} \frac{\Delta f}{\Delta C_i} \tag{3.14}$$

And, when generalized to include all components,

$$S_C^f = [S_{C_1}^f \quad S_{C_2}^f \quad \ldots \quad S_{C_n}^f] = [\frac{\partial f}{\partial C_1} \quad \frac{\partial f}{\partial C_2} \quad \frac{\partial f}{\partial C_n}] \tag{3.15}$$

The RHS of equation (3.15) above may be identified as being the gradient of the response function. Hence, we have:

$$S = \begin{bmatrix} \nabla f_1 \\ \nabla f_2 \\ \cdots \\ \nabla f_m \end{bmatrix} \tag{3.20}$$

Matrix S, which has the structure of a Jacobian matrix, is called sensitivity matrix. It can be represented more compactly as:

$$S = \left[\frac{\partial f_i}{\partial C_j} \right] \quad , (i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, m). \tag{3.21}$$

The notation (i = 1, 2, ..., n; j = 1, 2, ..., m) signifies that integer variable i, which represents row numbers, varies most. The quantity j denotes column numbers.

Apart from the first, second and higher order sensitivities may also be defined by including more terms from the Taylor series expansion of equation (3.12). The resulting matrices, however, are not convenient to handle and are only seldom used.

It may also prove profitable to work with normalized values of f and C. This results in relative and semi-relative sensitivities:

Relative sensitivity: $$S^{f,r}_{C_i,r} = \frac{\partial \ln f}{\partial \ln C_i} \tag{3.22}$$

Semi-relative sensitivity in f: $$S^{f,r}_{C_i} = \frac{\partial \ln f}{\partial C_i} \tag{3.23}$$

Semi-relative sensitivity in C: $$S^{f}_{C_i,r} = \frac{\partial f}{\partial \ln C_i} \tag{3.24}$$

Simplifying equations (3.22) -(3.24) and substituting $\frac{\partial f}{\partial C_i} = S^f_{C_i}$, we have:

Relative sensitivity: $$S^{f,r}_{C_i,r} = \frac{C_i}{f} S^f_{C_i} \tag{3.25}$$

Semi-relative sensitivity in f: $$S^{f,r}_{C_i} = \frac{1}{f} S^f_{C_i} \tag{3.26}$$

Semi-relative sensitivity in C: $$S^{f}_{C_i,r} = C_i S^f_{C_i} \tag{3.27}$$

From the last three equations it follows that,

$$S^{f}_{C_i} = \frac{S^{f,r}_{C_i} S^{f}_{C_p,r}}{S^{f,r}_{C_p,r}} \tag{3.28}$$

Equation (3.28) relates the absolute sensitivity to the relative and semi-relative sensitivities. It states that the absolute sensitivity is equal to the product of the semi-relative sensitivities diminished by the relative sensitivity.

The above relations for relative and semi-relative sensitivities were concerned with the individual component level and there is the implicit understanding of a continuous response function. The relations can be developed further to include the entire parameter vector and apply for the discrete case as well. To achieve that end, let S^r, $S^{r,f}$, and $S^{r,C}$ respectively denote the relative sensitivity matrix and the semi-relative sensitivity matrices in f and C. Consider next the matrix expansion of equation (3.25). Then,

$$S^r = \begin{bmatrix} S^{f_1,r}_{C,r} \\ S^{f_2,r}_{C,r} \\ \vdots \\ S^{f_m,r}_{C,r} \end{bmatrix} = \begin{bmatrix} \frac{c_1 S^{f_1}_{c_1}}{f_1} & \frac{c_2 S^{f_1}_{c_2}}{f_1} & \cdots & \frac{c_n S^{f_1}_{c_n}}{f_1} \\ \frac{c_1 S^{f_2}_{c_1}}{f_2} & \frac{c_2 S^{f_2}_{c_2}}{f_2} & \cdots & \frac{c_n S^{f_2}_{c_n}}{f_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{c_1 S^{f_m}_{c_1}}{f_m} & \frac{c_2 S^{f_m}_{c_2}}{f_m} & \cdots & \frac{c_n S^{f_m}_{c_n}}{f_m} \end{bmatrix} \tag{3.29}$$

Or, in compact notation,

$$S^r = \left[\frac{c_i S^{f_j}_{C_i}}{f_j} \right] \tag{3.30}$$

Equation (3.30) represents the relative sensitivity matrix. Similarly, the semi-relative sensitivity matrices may be shown to be of the forms:

$$S^{r,f} = \left[\frac{1}{f_j} S^{f_j}_{C_i} \right] \tag{3.31}$$

$$S^{r,C} = \left[c_i S^{f_j}_{C_i} \right] \tag{3.32}$$

The importance of relative and semi-relative sensitivities lies in the fact that they are necessary to compare sensitivities to different components and minimize overall network sensitivity. For instance, one theorem of RC active network design has it that the sum of relative sensitivities in a network is unity for impedance or admittance functions and zero for gain functions [17: p. 397].

Sensitivity analysis is usually effected in one of the *finite difference* [29, 17], *nodal* [17], *adjoint network* [17, 29, 31, 34], and *direct* [29, 17] methods. In addition to these three methods there is also a fourth one: the method of *perturbation*, where each element value is changed in turn by a certain percentage of its nominal value and the network reanalyzed. The reliability of this method, however, is open to question since, first of all, the amount by which each element value should be perturbed to obtain good results is not decidedly known. Secondly, the method is also inefficient, specially for large networks, as it requires a total of n network analyses, one for each element.

The finite difference method of sensitivity evaluation uses standard difference formulae from numerical analysis to approximate the partial derivatives of a function f with respect to the k-th design parameter. Thus, considering first order sensitivities,

$$\frac{\partial f}{\partial c_k} \approx \frac{f(c_k + \Delta c_k) - f(c_k - \Delta c_k)}{2\Delta c_k} \tag{3.33}$$

$$\frac{\partial f}{\partial c_k} \approx \frac{\Delta f}{\Delta c_k} = \frac{f(c_1, \ldots, c_k + \Delta c_k, \ldots, c_n) - f(c_1, \ldots, c_k, \ldots, c_n)}{\Delta c_k} \tag{3.34}$$

In equation (3.33), two analyses are performed for each component making the total number of analyses 2n. In equation (3.34), on the other hand, one circuit analysis is made per component with an increment added to each component in turn. These add to n for n components and, together with the nominal response, a total of n+1 analyses are required in the latter case.

The accuracy of the finite difference method is higher when the Δc_i are small ($<1\%$ for most circuits [17]) but then the difference of two nearly equal quantities has to be taken and this causes numerical inaccuracies. Also, excessive number of analyses are required as shown in the previous paragraph. For these reasons, the method is computationally inefficient for large networks and is not usually recommended.

In the nodal method of sensitivity evaluation, the notation y_{kl}^{ij} on p.21 earlier for an admittance element is convenient to use. The absolute sensitivity $S_{y_{kl}}^{T_{ba}}$ of a transfer function T_{ab}

between an input node a and an output node b of an n-node network to an admittance element y_{kl}^{ij} is then defined as:

$$S_{y_{kl}^{ij}}^{T_{ba}} = \frac{\partial T_{ba}}{\partial y_{kl}^{ij}} \tag{3.35}$$

After some manipulations [17: pp. 403 - 404], equation (3.35) takes the form of equation (3.36) next, where sensitivity is shown as a product of two factors obtained from the elements of Z:

$$S_{y_{kl}^{ij}}^{T_{ab}} = -(Z_{bi}-Z_{bj})\ (Z_{ka}-Z_{la}) \tag{3.36}$$

where Z_{bi}, Z_{bj}, Z_{ka}, and Za_{la} are identified with the entries in matrix Z, the inverse of the NAM.

Once the inverse NAM is obtained, a single network is sufficient to compute the required sensitivities using the nodal method of sensitivity analysis. The method, however, has the disadvantage of relying on matrix inversion which is usually considered as an inefficient procedure.

One approach to circumvent the inconveniences produced by the nodal method of sensitivity analysis is to use the *adjoint network method*. This is based upon Tellegen's Theorem which, when generalized, states that if $v_1(t)$, $v_2(t)$, ..., $v_{b(t)}$ are the b branch voltages of a b-branch network and $\hat{i}_1(t)$, $\hat{i}_2(t)$, ... , $\hat{i}_b(t)$ are the b branch currents of another b-branch network that has the same topology as the first, then, for all times t and τ, [28: p.145],

$$\sum_{k=1}^{b} v_k(t)\ \hat{i}_k(\tau) = 0 \tag{3.37}$$

The validity of Tellegen's Theorem even if applied to two different networks is the success behind the adjoint network method. In this method, a related (or *adjoint* or *companion*) network to the original one is first derived via a set of rules. Two sets of network analyses are then performed, one on the original network and another on the adjoint, to ascertain all branch currents and voltages. Absolute sensitivities are finally discerned as products of corresponding branch currents and/or voltages. An increase in computational efficiency is realized by recognition of the fact that the nodal admittance matrix of the adjoint network is the transpose of the nodal admittance matrix of the originally specified network [32]. The only disadvantage of the method, if any, is the need to deal with two separate networks.

In the direct method of sensitivity analysis, on the other hand, network behavior is

expressed directly in terms of the components. As a result, there is no need for the adjoint or companion model. Also, all sensitivities are found by direct differentiations. The method has special attraction when used in conjunction with topological methods where network functions are generated as multilinear functions of the network elements [30]. In this case, all partial differentiations necessary to extract sensitivities are replaced by the RHS of equation (3.38) next:

$$\frac{\partial T_j}{\partial C_i} = \frac{\text{tree admittance products of } T_j \text{ containing } C_i}{C_i}, \tag{3.38}$$

where T_j is a term that contributes to the transfer function.

In practice, the need to perform partial differentiations in the quest for sensitivities is bypassed when using equation (3.38). This great simplification shall be put into use in this thesis.

So, to proceed, consider the design of an n-node network to realize the complex voltage gain transfer function

$$K = A + jB \tag{3.39}$$

where A and B are the real and imaginary parts of the gain. The phase of this function in radian units is given by

$$\Phi_{rad} = \tan^{-1}\frac{B}{A} \tag{3.40}$$

Consider also the gains G_{nep} in nepers and G_{dB} in decibels and the phase Φ_{deg} in degrees:

$$G_{nep} = \ln(K) \tag{3.41}$$

$$G_{dB} = \frac{20}{\ln(10)} G_{nep} \tag{3.42}$$

$$\Phi_{deg} = \frac{180}{\pi} \Phi_{rad} \tag{3.43}$$

Absolute sensitivity formulas for K, G_{nep}, G_{dB}, Φ_{rad} and Φ_{deg} shall now be developed. The frequency response sensitivity from the gain in nepers and the phase in radian units shall also be developed. We begin with the sensitivity of the complex gain to a change in a component C_i:

$$\begin{aligned} S_{C_i}^{k} &= \frac{\partial K}{\partial C_i} \\ &= \frac{\partial A}{\partial C_i} + j\frac{\partial B}{\partial C_i} \\ &= Re\ S_{C_i}^{K} + j\ Im\ S_{C_i}^{K} \end{aligned} \tag{3.44}$$

where

$$\begin{aligned} Re\ S_{C_i}^{k} &= \frac{\partial A}{\partial C_i} \\ Im\ S_{C_i}^{K} &= \frac{\partial B}{\partial C_i}\ . \end{aligned} \tag{3.45}$$

The sensitivity of the gain in nepers is, by definition,

$$\begin{aligned} S_{C_i}^{G_{nep}} &= \frac{\partial\ G_{nep}}{\partial\ C_i} \\ &= \frac{\partial\ \ln|K|}{\partial\ C_i}\ . \end{aligned}$$

Hence,

$$S_{C_i}^{G_{nep}} = \frac{1}{|K|}\ \frac{\partial|K|}{\partial C_i}\ . \tag{3.46}$$

But

$$\begin{aligned} \frac{\partial|K|}{\partial C_i} &= \frac{\partial\sqrt{A\ +\ jB}}{\partial C_i} \\ &= \frac{1}{|K|}\left(\frac{A\partial A}{\partial C_i}\ +\ \frac{B\partial B}{\partial C_i}\right). \end{aligned} \tag{3.47}$$

So that equation (3.46) becomes

$$S_{C_i}^{Gnep} = \frac{1}{|\ K\ |^{2}}\left(\frac{A\ \partial A}{\partial C_i}\ +\ \frac{B\ \partial B}{\partial C_i}\right). \tag{3.48}$$

Or, using equation (3.45) in equation (3.48),

$$S_{C_i}^{G_{nep}} = \frac{1}{|\ K\ |^{2}}\left(\ A\ Re\ S_{C_i}^{k}\ +\ B\ Im\ S_{C_i}^{K}\ \right). \tag{3.49}$$

Once the real- and imaginary-part sensitivities of the complex gain are available, equation

(3.49) is used to evaluate the sensitivity of the gain in nepers .

The sensitivity of the phase in radians is, by definition,

$$S_{C_i}^{\Phi_{rad}} = \frac{\partial \Phi_{rad}}{\partial C_i}$$

$$= \frac{\partial \left(\tan^{-1} \frac{B}{A} \right)}{\partial C_i} ,$$

which, when evaluated, leads to

$$S_{C_i}^{\Phi_{rad}} = \frac{A^2}{A^2 + B^2} \left(\frac{A \frac{\partial B}{\partial C_i} - B \frac{\partial A}{\partial C_i}}{A^2} \right)$$

$$= \frac{1}{A^2 + B^2} \left(A \frac{\partial B}{\partial C_i} - B \frac{\partial A}{\partial C_i} \right)$$

Using equation (3.45) and $A^2 + B^2 = |K|^2$ in the result just obtained, we get:

$$S_{C_i}^{\Phi_{rad}} = \frac{1}{|K|^2} \left(A \, Im \, S_{C_i}^K - B \, Re \, S_{C_i}^K \right) . \qquad (3.50)$$

Also,

$$(A - jB) \left(S_{C_i}^K \right) = (A - jB) \left(Re \, S_{C_i}^K + j \, Im \, S_{C_i}^K \right)$$

$$= \left(A \, Re \, S_{C_i}^K + B \, Im \, S_{C_i}^K \right) + j \left(Im \, S_{C_i}^K - B \, Re \, S_{C_i}^K \right) .$$

It follows that

$$\begin{aligned} Re \, [\, (A - jB) \, S_{C_i}^K \,] &= A \, Re \, S_{C_i}^K + B \, Im \, S_{C_i}^K \\ Im \, [\, (A - jB) \, S_{C_i}^K \,] &= A \, Im \, S_{C_i}^K - B \, Re \, S_{C_i}^K \end{aligned} \qquad (3.51)$$

Substituting the relations of equation (3.51) into those of equations (3.49) and (3.50), we have:

$$S_{C_i}^{G_{nep}} = \frac{1}{|K|^2} \, Re \, [\, (A - jB) S_{C_i}^K \,] \qquad (3.52)$$

$$S_{C_i}^{\Phi_{rad}} = \frac{1}{|K|^2} Im\left[(A - jB)\, S_{C_i}^{K}\right] \tag{3.53}$$

Furthermore, since

$$\frac{1}{|K|^2} = \frac{1}{(A + jB)(A - jB)}$$

and

$$\frac{A - jB}{|K|^2} = \frac{1}{A + jB} = \frac{1}{K},$$

we have:

$$S_{C_i}^{G_{nep}} = Re\, \frac{S_{C_i}^{K}}{K}; \tag{3.54}$$

$$S_{C_i}^{\Phi_{rad}} = Im\, \frac{S_{C_i}^{K}}{K}. \tag{3.55}$$

Similarly, the sensitivity of G_{dB} can be shown to be

$$S_{C_i}^{G_{dB}} = \frac{20}{\ln 10} Re\, \frac{S_{C_i}^{K}}{K} \tag{3.56}$$

while that of Φ_{deg} becomes

$$S_{C_i}^{\Phi_{deg}} = \frac{180}{\pi} Im\, \frac{S_{C_i}^{K}}{K}. \tag{3.57}$$

The sensitivity of the magnitude of the complex gain can be determined by first noting the relationship

$$\frac{\partial \ln|K|}{\partial C_i} = \frac{1}{|K|} \frac{\partial |K|}{\partial C_i} \tag{3.58}$$

which implies that

$$\frac{\partial K}{\partial C_i} = \frac{|K|\, \partial \ln|K|}{\partial C_i}. \tag{3.59}$$

Using the definition of the sensitivity of the gain in nepers, equation (3.59) reduces to

$$S_{C_i}^{|K|} = |K|\, S_{C_i}^{G_{nep}}; \tag{3.60}$$

or, equivalently,

$$S_{C_i}^{|K|} = |K| \; Re \; \frac{S_{C_i}^{K}}{K} . \tag{3.61}$$

Thus, as per equation (3.61), the magnitude sensitivity of the complex gain is a real quantity. This is plausible since the sensitivity of a real quantity is often real. On the other hand, suppose that sensitivity of the complex gain is also required. Let this be expressed as a complex quantity of the form

$$S_{C_i}^{K} = Re \; S_{C_i}^{K} + j \; Im \; S_{C_i}^{K} . \tag{3.62}$$

Dividing each of the RHS and LHS of equation (3.62) by K itself results in

$$S_{C_i}^{K} = K \left(Re \; \frac{S_{C_i}^{K}}{K} + j \; Im \; \frac{S_{C_i}^{K}}{K} \right) . \tag{3.63}$$

Thus,

$$S_{C_i}^{K} = K \left(Re \; S_{C_i}^{G_{nep}} + j \; S_{C_i}^{\Phi_{rad}} \right) . \tag{3.64}$$

The individual contributions of the gain and phase sensitivities to the overall frequency response sensitivity can thus be seen from equation (3.64).

3.5 An Overview of Optimum Seeking Methods

Optimal network design, by its very nature, is a multivariable minimization process of a real valued function $E(c_1, c_2, \ldots, c_n)$, the objective function, of n components over a set S in an n-dimensional parameter space. When S defines the entire n-dimensional parameter space, the minimization process is said to be *unconstrained*. Otherwise, it is *constrained* by the conditions that define S. We deal in this thesis with unconstrained optimization which, in general, may be divided into the three categories of *function approximation*, *direct search*, and *gradient methods*.

Function approximation methods are used to find the constants in a network function, i.e., the network function itself is determined. The procedure, therefore, contrasts that used in direct search and gradient methods, where a general network function is available and the objective is to adjust the components so as to meet a specification. *Curve fitting* and *polynomial interpolation* [3], the methods of *point* and *coefficient matching* [2, 8, 27, 30], the *Remez* and

Zero shifting methods [3, 27, 46], and the *maximally flat approximation* method [3, 27] are some of the methods that fall into this category.

Direct search methods rely only on evaluating the objective function at a sequence of parameter vectors and comparing values to reach the minimum. No gradient information is sought and so there is no need for sensitivity analysis. These methods are intended only for problems in which E(c) is discontinuous, $\nabla E(c)$ is discontinuous at the solution, $\nabla E(c)$ has so many discontinuities that a method assuming continuity of $\nabla E(c)$ will fail, or the discontinuities are presumed to have no special structure [45: p. 93]. In any case, the basic problem in the design of a direct search method is to determine the component vector c^{r+1} given the component vectors $c^1, c^2, \ldots, c^r$ and the corresponding error values $E^1, E^2, \ldots, E^r$. The theory in this regard is well documented in the literature [46, 48, 49, 54]. Some of the extensively used direct search methods include the *Fibonacci search*, *Golden Section search*, *pattern search*, *the simplex technique* (not the same as the simplex method of linear programming), *Powell's Quadratic Interpolation method*, and *Davidon's Cubic Interpolation Method*.

Gradient methods, on the other hand, are based on the observation that the increase or decrease of the performance function in the direction of Δc depends on whether the directional derivative $\nabla E(c)^T \Delta c$, which is also called projected gradient [43: p.133], is positive or negative. The method may or may not use derivative information for arriving at the modified set of parameters and, when derivative information is used, this is done through sensitivity analysis. *The steepest-descent method*, *the Fletcher-Powell minimization procedure* (also called *the Davidon-Fletcher-Powell* or *DFP method*), *the Fletcher-Reeves method*, and *least-squares Taylor method* come under this category. The list of methods may further be expanded by mentioning *Broyden's Method* [2], *Newton's method* [27], *Generalized Newton-Raphson method* [29], *the least p-th Taylor method* [27], *Smith's method* [54], *Powell's method* [54], and *the Gauss-Newton method* [43].

Gradient methods that do not require the evaluation of derivatives depend on the properties of conjugate directions. The procedure followed is to make linear searches along mutually conjugate directions. As a result, the first derivatives are not computed or approximated by standard difference formulae. Instead, they are estimated and, as the minimization proceeds, the estimates are improved. Typical methods of this class are Smith's method and Powell's method.

Gradient methods that require the evaluation of derivatives are divided into *first order* and *second order methods* - based upon the multivariable Taylor series expansion of the next value of objective function in the neighborhood of a current value the parameter vector:

$$E^{j+1} = E^j(c^j) + \sum_{i=1}^{n} \frac{\partial E^j}{\partial c_i^j} \Delta c_i^j + \frac{1}{2} \sum_{k=1}^{n} \sum_{l=1}^{n} \frac{\partial^2 E^j}{\partial c_k^j \, \partial c_l^j} \Delta c_k^j \Delta c_l^j + \ldots \quad (3.65)$$

or, equivalently,

$$E^{j+1} = E^j(c^j) + \nabla E^j(c^j)^T \Delta c^j + \frac{1}{2} \Delta c^{jT} H^j(c^j) \Delta c^j + \ldots \quad (3.66)$$

Retaining terms up to and including the second order in equation (3.66),

$$E^{j+1} \approx E^j(c^j) + \nabla E^j(c^j)^T \Delta c^j + \frac{1}{2} \Delta c^{jT} H^j(c^j) \Delta c^j . \quad (3.67)$$

where

- the column vector

$$\nabla E^j(c^j) = \left[\frac{\partial E^j(c^j)}{\partial c_1^j} \quad \frac{\partial E^j(c^j)}{\partial c_2^j} \quad \ldots \quad \frac{\partial E^j(c^j)}{\partial c_n^j} \right]^T \quad (3.68)$$

is the *error gradient*, and

- the nxn symmetric matrix $H^j c^{(j)}$ of second partial derivatives is the *Hessian matrix*. The (k,l)-th entry of this matrix is equal to $\frac{\partial^2 E^j(c^j)}{\partial c_k^j \, \partial c_l^j}$, i.e.,

$$H^j(c^j) = \begin{bmatrix} \frac{\partial^2 E^j(c^j)}{\partial c_1^{j2}} & \frac{\partial^2 E^j(c^j)}{\partial c_1^j \, \partial c_2^j} & \ldots & \frac{\partial^2 E^j(c^j)}{\partial c_1^j \, \partial c_n^j} \\ \frac{\partial^2 E^j(c^j)}{\partial c_2^j \, c_1^j} & \frac{\partial^2 E^j(c^j)}{\partial c_2^{j2}} & \ldots & \frac{\partial^2 E^j(c^j)}{\partial C_2^j \, c_n^j} \\ \cdot & \cdot & \cdot & \cdot \\ \frac{\partial^2 E^j(c^j)}{\partial c_n^j \, \partial c_1^j} & \frac{\partial^2 E^j(c^j)}{\partial c_n^j \, \partial c_2^j} & \ldots & \frac{\partial^2 E^j(c^j)}{\partial c_n^{j2}} \end{bmatrix} \quad (3.69)$$

Equation (3.67) can be put as

$$E^{j+1} \approx E^j(c^j) + \Delta E^j , \quad (3.70)$$

where ΔE^j is the correction that must be added on E^j to obtain E^{j+1} and is given by

$$\Delta E^j = \nabla E^j (c^j)^T \Delta c^j + \frac{1}{2} \Delta c^{jT} H^j (c^j) \Delta c^j . \quad (3.71)$$

i.e., ΔE^j is expressed by a quadratic model of the objective function.

The three-term approximation of the objective function as given by equation (3.67) represents a quadratic function. This is pleasing since quadratic functions possess well determined minima and rapid rates of convergences [52: pp. 23 -24].

The classification of gradient methods into first and second order methods is related to the use of terms from equation (3.67). If the Hessian matrix is assumed small enough to be neglected, then a first order method results. Otherwise, a second order method is defined.

Now, at a minimizing set of design parameters $c = \hat{c}$, E(c) is increased for any small change Δc away from $\hat{c}$. As a result, the term depending linearly on Δc_i in equation (3.67) must be zero while the term with quadratic dependence on Δc_i must be positive. In the first case, $\nabla E = 0$ and in the second case H must be a p.d. matrix. These two conditions are the requirements for a minimum. If H is positive semidefinite, then higher order terms of the expansion must be considered.

The steepest-descent method is the fundamental first order method. It was first proposed by A. Cauchy in 1845 [53: p.188] and is based upon the idea that *a small motion in the direction opposite to the gradient always minimizes E*. This can be justified as follows:

For small ΔC, equation (3.67) can be approximated by

$$E^{j+1} \approx E^j(c^j) + \nabla E^j(c^j)^T \Delta c^j . \quad (3.72)$$

To achieve the largest reduction in E^{j+1}, the second term in the RHS of equation (3.72) should be as large as possible and negative. But, from vector calculus,

$$\nabla E^j(c^j)^T \Delta c^j = |\nabla E^j| \, |\Delta c^j| \cos\beta \quad (3.73)$$

where β is the angle between the two vectors.

For given $|\nabla E|$ and $|\Delta c|$, the largest negative value of E^{j+1} is obtained when $\beta = \pi$ and this means that the two vectors are collinear and oppositely directed. Hence, for small step size, the step should be taken in the negative gradient direction. This is the direction in which the error E decreases most rapidly.

In the method of steepest-descent, therefore, one starts from an initial trial point c^j and iteratively moves towards the minimum point according to the rule

$$c^{j+1} = c^j + \alpha^j s^j \quad (3.74)$$

where α^j, a positive valued scalar, is called *the step length*. The parameter vector c^{j+1} is known as *Cauchy point* while the product $\alpha^j s^j$ may be called *Cauchy step*. The quantity s^j is a unit

vector in the direction of the negative gradient, i.e.,

$$s^j = -\frac{\nabla E}{\|\nabla E\|} \tag{3.75}$$

The step length α^j is found from a one-dimensional search along the direction of s^j.

The steepest-descent method is attractive when the starting point is well removed from the minimum. In the vicinity of the solution, however, convergence is at most linear and very slow. Infact, the method as it stands is not quite effective in most problems and numerous modifications have been suggested in the literature.

Proceeding further and taking the gradient of equation (3.67), we have

$$\nabla E(c^{j+1}) = \nabla E^j(c^j) + H^j \Delta c^j + \ldots \tag{3.76}$$

At optimality, $\nabla E^{j+1} = 0$ and, neglecting higher order terms, equation (3.76) reduces to

$$\nabla E(\hat{c}) = -H^j \Delta c^j . \tag{3.77}$$

This yields

$$\Delta c^j = -H^{-1^j} \nabla E^j(\hat{c}) . \tag{3.78}$$

The value Δc^j in equation (3.78) is called *Newton's step* and $c^{j+1} = c^j + \Delta c^j$ is called *Newton's point*. The resulting minimization algorithm is known as *Newton's Method*.

Newton's method is the fundamental second order method and is often regarded as the standard against which other algorithm's are measured [45: p. 106]. It converges at a *quadratic rate* in the immediate vicinity of a local minimum, but without restrictions on its step size it is often unreliable elsewhere. Specifically, in regions outside the validity of the quadratic model, difficulties and even failure are faced.

Apart from divergence problems, a major disincentive to the use of Newton's method is the computation and inversion of the Hessian matrix. Alternatives to this process are provided by the *finite difference* and *quasi-Newton methods*. These are also called *discrete Newton method* and *variable metric methods*, respectively.

In the *finite difference Newton method*, the Hessian matrix is approximated by finite differences of the gradient vector. This is achieved by taking increments h_i in each coordinate direction e_i and forming a matrix G whose i-th column is equal to the forward-difference approximation $\frac{\nabla E(c^j + h_i e_i) - \nabla E(c^j)}{h_i}$. The symmetric matrix $\frac{1}{2}(G + G^T)$ is then constructed and used to replace the H in Newton's method.

The finite difference Newton method requires n gradient evaluations, where n is the

number of variables, and solutions to a set of linear equations in each iterations. The approximate Hessian matrix may also not be p.d. For these reasons, the method is recommended only for large systems where the amount of differencing can be reduced.

An effective solution to avoid the disadvantages of discrete Newton methods is furnished by quasi-Newton methods. The introduction of these methods, it is reported [52], has greatly increased the range of problems to be solved.

The basic theory of quasi-Newton methods is based upon the fact that an approximation of the curvature of a non-linear function can be computed without explicitly forming the Hessian matrix, i.e., instead of $\Delta c^j = - H^{-1^j} \nabla E^j$, the system $\Delta c^j = - B^j \nabla E^j$ is solved, where B is an approximation to H. The initial approximation B^0 can be any p.d. matrix but, in the absence of any better estimate, the choice of the identity matrix is often made. In the latter case the process is equivalent to an iteration of the steepest-descent method. The approximate Hessian matrix B^j available at the j-th iteration is intended to reflect the curvature information already accumulated. After c^{j+1} has been computed, a new Hessian approximation B^{j+1} is made using the relation $B^{j+1} = B^j + U^j$, where U is an update matrix.

So, in the above process, successive estimates of H are updated to retain a key Newton property - hence the name "quasi-Newton". Also, in the same process, the approximation to the inverse Hessian matrix (which Davidon noted as metric) varies from iteration to iteration - hence the description "variable metric".

The general availability of the digital computer has revived interest in update formulas. Some of the well known such formulas include the *Powell-Symmetric-Broyden (PSB)*, the *Davidon-Fletcher-Powell (DFP)*, and the *Broyden-Fletcher-Goldfard-Shanno (BFGS)* formulas [45: pp.116-123]. The DFP formula, in particular, and which is also known as the *Fletcher-Powell minimization procedure*, is regarded as the most powerful procedure now available for finding the minimum of a general function [50: p. 79]. The method uses the method of conjugate directions [48, 50] and updates at each iteration the symmetric p.d. matrix G by simulating the steepest-descent method without going through the tedium of computing and inverting H as indicated above.

One way to improve the efficiency and reliability of the steepest-descent method is to associate it with *conjugacy* properties. This is the aim of the *conjugate gradient method*. Conjugacy of a set of vectors $s_1, s_2, \ldots, s_n$ to a p.d. matrix H is the property that $s_i^T H s_j = 0$, for $i \neq j$. The vectors $s_1, s_2, \ldots, s_n$ are known as *conjugate vectors*. A *conjugate directions method*, of which the conjugate gradient method forms a part, is one which

generates these directions (vectors) when applied to a quadratic function with Hessian matrix H.

The *Fletcher-Reeves method* is an example of a conjugate gradient method. It minimizes the quadratic model of equation (3.67) by successive linear searches along mutually conjugate directions. The initial direction is taken in the direction of the steepest-descent, i.e., $s_1 = -\nabla E_1$, while subsequent mutually conjugate directions are chosen so that $s_{i+1} = -\nabla E_{i+1} + \Sigma \beta_n s_n$, $n=1,2,\ldots i$, $i=1,2,\ldots$, where the coefficients β_n are to be determined. For a quadratic function, all coefficients except β_i vanish [54: p.123] and so $s_{i+1} = -\nabla E_{i+1} + \beta_i s_i$, where β_i is found from $\beta_i = |E_{i+1}|^2 / |E_i|^2$.

One approach to rectify the convergence problems of the above methods is to use a *hybrid* method. This approach has been implemented into a successful computer program by Wing and Behar [36]. The authors used the steepest-descent method at the initial iteration phases and then utilized Newton's method once a refined estimate to the solution was available. The transition from the steepest-descent to the Newton process was made when the error changed by less than 10 percent over four consecutive iterations.

Among other approaches that avoid reference to the Hessian matrix is the *least p-th Taylor method*. This method, through the incorporations of suitable damping, linear search and Levenberg - Marquardt methods, can be made to combine the advantages of the steepest-descent and Newton methods and exhibit good convergence properties.

The choice of an appropriate minimization algorithm is problem dependent. In general, gradient methods are superior to direct search methods if the functions involved have continuous derivatives which can be evaluated analytically [46: p.57]. Also, another great advantage of gradient methods, is that they will inherently stay away from a saddle point [47: p.227]. Function approximation methods, on the other hand, are useful when the approximating function is required for interpolating or predicting of the values of a specified function given by a table of calculated values.

In this thesis, of the many gradient methods available, that of the *least p-th Taylor method* is used invariably for the optimization tasks since, given suitable linear search and damping supports, this is one of the methods best suited to minimizations involving network functions that can be generated as multilinear functions of the design parameters. Infact, it will be fair to call the *least p-th Taylor method* used in the thesis as the *modified least p-th Taylor method* as indicated in section 3.7 below.

3.6 The Least p-th Taylor Method

The least p-th Taylor method is a generalization of the familiar least-squares method [51]. It furnishes the advantages of smaller maximum in-band error and improved convergence over the latter [35] and holds better for higher values of p [2: p.215]. In addition to these, and as discussed in section 3.5 above, the least p-th Taylor method combines most of the advantages of Newton's method with those of the steepest-descent. In the sequel, we shall retain the subscript p on both of E and H for clarity purposes.

Let w, F, F^j be all real and the exponent p be restricted to be an even integer. Then, re-writing equation (3.8), we have:

$$E_p = \sum_{i=1}^{m} (\epsilon_i)^p \tag{3.79}$$

or, in iterative form,

$$E_p^j = \sum_{i=1}^{m} (\epsilon_i^j)^p . \tag{3.80}$$

The value of the vector c^j in the next iteration is

$$c^{j+1} = c^j + \Delta c^j . \tag{3.81}$$

Similarly, the value of E_p^j in the next iteration is

$$E_p^{j+1} = E_p^{j+1}(c^{j+1}) = E_p^{j+1}(c^j + \Delta c^j) . \tag{3.82}$$

Differentiating equation (3.80) with respect to the l-th parameter c_l gives:

$$\frac{\partial E_p^j}{\partial c_l^j} = p \sum_{i=1}^{m} \left(\epsilon_i^j \right)^{p-1} \frac{\partial \epsilon_i^j}{\partial c_l^j} \tag{3.83}$$

A second order partial differentiation of equation (3.80) with respect to c_k and c_l yields the (k,l)-th component of the Hessian matrix:

$$\frac{\partial^2 E_p^j}{\partial c_k^j \, c_l^j} = p \sum_{i=1}^{m} \left\{ (p-1) \left(\epsilon_i^j \right)^{p-2} \frac{\partial^j \epsilon_i^j}{\partial c_k^j} \frac{\partial \epsilon_i^j}{\partial c_l^j} + \left(\epsilon_i^j \right)^{p-1} \frac{\partial^2 \epsilon_i^j}{\partial c_k^j \, c_l^j} \right\} \tag{3.84}$$

The second term in the summation of equation (3.84) is usually much less than the first and may be neglected. Hence,

$$\frac{\partial^2 E_p^j}{\partial c_k^j \, c_l^j} = p \sum_{i=1}^{m} (p-1) \left(\epsilon_i^j \right)^{p-2} \frac{\partial \epsilon_i^j}{\partial c_k^j} \frac{\partial \epsilon_i^j}{\partial c_l^j} . \tag{3.85}$$

It is also necessary to work with the iterative forms of equations (3.76) - (3.78). So, taking the gradient of equation (3.67),

$$\nabla E_p^j \left(c^{j+1} \right) \approx \nabla E_p^j \left(c^j \right) + H_p^j \, \Delta c^j. \tag{3.86}$$

Optimally,

$$\nabla E_p^j \left(c^{j+1} \right) = \nabla E_p^j \left(\hat{c} \right) = 0 .$$

Thus,

$$\nabla E_p^j \left(c^j \right) + H_p^j \, \Delta c^j = 0 . \tag{3.87}$$

From equation (3.87), the parameter increment Δc^j can be determined as

$$\Delta c^j \approx - H_p^{j^{-1}} \nabla E_p^j \tag{3.88}$$

If E_p^j is quadratic, Δc^j is reached in exactly one step [7]. When this is not the case, equation (3.88) provides the basis of the iterative scheme called the generalized Newton-Raphson method, whereby equation (3.81) takes the form

$$c^{j+1} = c^j - H_p^{j^{-1}} \nabla E_p^j . \tag{3.89}$$

The generalized Newton-Raphson method, although quadratically convergent, may or may not converge at all depending on the initial guess of the parameter vector c_0. To counteract this tendency of divergence, equation (3.89) is usually modified to

$$c^{j+1} = c^j - \alpha^j \, H_p^{j\,-1} \nabla E_p^j \tag{3.90}$$

where α^j is chosen to minimize E_p^{j+1} in the negative direction indicated by $H_p^{j\,-1} \nabla E_p^j$. Even that may not be effective [7]. Also, the computations of H and its inverse are generally considered time-consuming operations, adding to the disadvantages of the method.

Gradient methods that use the exact H have been reported in [36] and [42]. These are based on the elements of the inverse NAM, i.e., nodal method of sensitivity analysis. In this study, on the other hand, we use the direct method of sensitivity analysis and intend to avoid the computation of H. To that end, the optimality equation (3.87) can be put as

$$\frac{\partial E_p^j}{\partial c_l^j} + \sum_{k=1}^{n} \frac{\partial^2 E_p^j}{\partial c_k^j \, \partial c_l^j} \, \Delta c^j = 0 , \quad l = 1, 2, \ldots, n. \tag{3.91}$$

Inserting equations (3.83) and (3.85) into equation (3.91),

$$p \sum_{i=1}^{m} \left(\epsilon_i^j \right)^{p-1} \frac{\partial \epsilon_i^j}{\partial c_l^j} + \sum_{k=1}^{n} p \left\{ \sum_{i=1}^{m} (p-1) \left(\epsilon_i^j \right)^{p-2} \frac{\partial \epsilon_i^j}{\partial c_k^j} \frac{\partial \epsilon_i^j}{\partial c_l^j} \right\} \Delta c_k^j = 0 . \qquad (3.92)$$

which can be put in the more suitable form

$$\sum_{i=1}^{m} \left(\epsilon_i^j \right)^{p-1} \frac{\partial \epsilon_i^j}{\partial c_l^j} + (p-1) \sum_{k=1}^{n} \left\{ \sum_{i=1}^{m} \left(\epsilon_i^j \right)^{p-2} \frac{\partial \epsilon_i^j}{\partial c_k^j} \frac{\partial \epsilon_i^j}{\partial c_l^j} \right\} \Delta c_k^j = 0 . \qquad (3.93)$$

Equation (3.93), can be written compactly by introducing matrices A and B and vector e. A contains the first partial derivatives of the calculated response while B and e are formed of the errors at the m sample points. Thus,

$$A^j = \left(a_{kl}^j \right) : \quad \text{an m x n matrix;}$$

$$a_{kl}^j = w \left(\omega_k \right) \frac{\partial F^j \left(\omega_k, c^j \right)}{\partial c_l^j} = - \frac{\partial \epsilon_k^j}{\partial c_l^j} , \qquad (3.94)$$

$$B^j = diag \left(\epsilon_i^j \right)^{p-2} , \quad \text{an m x m diagonal matrix;}$$

$$e^j = \left[\left(\epsilon_1^j \right)^{p-1} \left(\epsilon_2^j \right)^{p-1} \ldots \left(\epsilon_m^j \right)^{p-1} \right]^T , \quad \text{an m element vector}$$

where k = 1, 2,...,m and l = 1, 2, ..., n.

∇E and H can now be expressed in terms of A, B, and e. Thus, from equation (3.83),

$$\nabla E_p^j \left(c^j \right) = -p \, A^j \, e^j , \qquad (3.95)$$

and, from equation (3.85),

$$H_p^j \left(c^j \right) \approx p \left(p - 1 \right) A^{jT} B^j A^j . \qquad (3.96)$$

Hence, equation (3.87) becomes,

$$-p \, A^{jT} e^j + p \, (p-1) \, A^{jT} B^j A^j \, \Delta c^j = 0 ; \qquad (3.97)$$

from which the increment Δc^j can be calculated as

$$\Delta c^j = (p-1)^{-1} \left(A^{jT} B^j A^j \right)^{-1} A^{jT} e^j . \qquad (3.98)$$

The quantity Δc^j in equation (3.98) shall be called *least p-th Taylor point*. For p = 2, equation (3.98) reduces to

$$\Delta c^j = \left(A^{jT} A^j \right)^{-1} A^{jT} e^j . \qquad (3.99)$$

Equation (3.99) forms the basis of a non-linear least squares (NLLS) iteration scheme

called the *Gauss-Newton method* and the resulting Δc^j is called *Gauss-Newton step*. The parameter update c^{j+1} is known as *Gauss-Newton point*.

Thus, the algorithm for the least p-th Taylor method can be put in the following way:

STEP 1: calculate Δc^j from equation (3.98);

STEP 2: update c^{j+1} using equation (3.81);

STEP 3: update A^{j+1}, B^{j+1}, and e^{j+1};

STEP 4: if $e^{j+1} \approx 0$, then stop; else repeat steps [1] - [3].

An apparent disadvantage of the least p-th Taylor method is that it does not tolerate poor initial conditions. The roots of this problem lie in the approximations effected when obtaining equations (3.67), (3.85) and (3.88) - because of which the Hessian matrix becomes non-p.d.. Also, very large and/or small numbers may occur in the calculations (specially when p is large, i.e., $p \geq 10$) giving rise to *floating-point overflow*.

The above problems have extensively been studied for the least squares (Gauss-Newton) method and solutions have been developed using a technique called the Levenberg-Marquardt method. This is even more powerful when supported by component damping techniques. In the next two sections, we shall examine similar procedures that make the least p-th Taylor method *immune* to the problems of divergence and floating-point overflow.

3.7 Trust Neighborhoods and Levenberg - Marquardt Methods

If, in the least p-th Taylor method, divergence is to be precluded, then the value of the objective function must be decreased with each iteration. This requirement is known as *the downhill property* of the error along the error surface. Indeed, for an infinitesimally small positive scalar λ, equation (3.66) takes the form

$$E_p(c + \lambda \Delta c) \rightarrow E_p(c) + \lambda \nabla E_p(c)^T \Delta c . \qquad (3.100)$$

Substituting the expressions for the Gauss-Newton step and the gradient vector from equations (3.98) and (3.95), respectively, into the relation of equation (3.100), we get:

$$E_p(c + \lambda \Delta c) \rightarrow E_p(c) - \frac{\lambda}{p\,(p-1)} \nabla E_p(c)^T (A^T B A)^{-1} \nabla E_p(c) . \qquad (3.101)$$

Since $(A^TBA)^{-1}$ is p.d. [35], then, by the property of quadratic forms, the quadratic form in equation (3.101) is also p.d. Furthermore, since λ and p are also positive, one can write

$$E(c + \lambda \Delta c) - E(c) \leq 0 , \qquad (3.102)$$

i.e.,

$$E(c + \lambda \Delta c) \leq E(c) . \tag{3.103}$$

Thus, although equation (3.98) does not, in general, minimize E in the least p-th Taylor sense, equation (3.103) suggests that the convergence of the least p-th Taylor method can be realized by restricting the size of Δc. This can be made more profitable using a linear search in the direction of Δc in each iteration. The practical problem, however, is that the resulting convergence is not as anticipated and often very slow as too small step sizes will be taken almost parallel to the constant E-contours.

A remedy to the above problem is furnished by *the Levenberg-Marquardt method*. This was first suggested (independently) by K. Levenberg (1944) and D. W. Marquardt (1963) in the context of non-linear least squares, i.e., to circumvent the difficulty caused by non-positive definite Hessian matrices in Newton's method [52]. For this reason, we shall first consider the theory as applied to Newton's method and then study it with reference to the least p-th Taylor method.

When applied to the Newton's method, the Levenberg-Marquardt method combines the advantages of steepest-descent and Newton's methods and serves as a switching policy between a full Newton step and an infinitesimal Cauchy step and, instead of solving equation (3.78), one is led to solve for the parameter increment vector Δc from the equation:

$$\Delta c = - [H(c) + \lambda I]^{-1} \nabla E(c) . \tag{3.104}$$

where the scalar parameter $\lambda \geq 0$, which takes on the form of a Lagrange multiplier, is called Levenberg-Marquardt (LM) parameter and I is the identity matrix.

The question of when to use and then to step from steepest-descent to Newton directions, and when to limit the Cauchy and Newton steps to a reasonable maximum is answered by the concept of *trust neighborhoods*. A *trust neighborhood* is also known as a *trust region* and refers to the region in the n-dimensional parameter space where the quadratic model of the objective function is valid.

Depending on the manner in which the search direction is computed, non-linear optimization methods can be divided into *step-length-based methods* and *trust-region methods*.

A *step-length-based method* is one for which c^{j+1} is computed from $c^{j+1} = c^j + \alpha \Delta c^j$, where α is called a *step-length* and determined by linear search along Δc^j. This idea was introduced in the discussion of the steepest-descent method in section 3.5 above. Unlike trust-region methods, neither the Hessian matrix nor its approximation is modified in step-length based methods.

The basic structure of the j-th iteration in a step-length-based method is:

STEP 0: begin with an initial estimate c^0;

STEP 1: determine the direction of search Δc^j;

STEP 2: find α^j that will minimize $E(c^j + \alpha \Delta c^j)$ with respect to α;

STEP 3: set $c^{j+1} = c^j + \alpha^j \Delta c^j$.

A *trust region* or *trust neighborhood* consideration leads to what Fletcher[52] calls *restricted-step methods*. In *restricted-step* or *trust-region methods*, some neighborhood Ω^j, *the trust neighborhood*, of c^j in which the quadratic model approximation of equation (3.67) agrees with the value of $E(c^j + \Delta c^j)$ is first defined. Then c^{j+1} is computed from $c^{j+1} = c^j + \Delta c^j$, where the correction Δc^j minimizes equation (3.67) for all $c^j + \Delta c^j$ in Ω^j. In order to ensure a descent (downhill) property, several trial vectors Δc are explored before finding a satisfactory Δ^j.

The idea of the trust-region approach, therefore, is to accept the minimum of the quadratic model only as long as the quadratic model adequately reflects the behavior of E. This is usually effected by defining a *trust radius* R^j in the j-th iteration as the ratio of the actual error reduction to the predicted error reduction, i.e.,

$$R^j = \frac{E^j - E(c^j + \Delta c^j)}{E^j - q^j} \tag{3.105}$$

where the term q^j in the denominator of equation (3.105) represents the approximation indicated as equation (3.67) above. Substituting the relation in equation (3.67) into that of equation (3.105), we get:

$$R^j = \frac{E(c^j + \Delta c^j) - E^j(c)}{\nabla E^j(c^j)^T \Delta c^j + \Delta C^{j^T} H^j(c^j) \Delta c^j} \tag{3.106}$$

or, equivalently,

$$R^j = \frac{E^{j+1} - E^j}{\nabla E^{j^T} \Delta c^j + \Delta C^{j^T} H^j \Delta c^j} \; . \tag{3.107}$$

The *trust radius* R provides a measure of how good equation (3.67) approximates the actual error function. The closer R is to unity, the better the approximation. If $R \gg 1$, then this is considered to be good news; $R \ll 1$ indicates bad news.

Next, consider the solution of the non-linear relation of equation (3.104).

There are various approaches to the computation of the optimum λ that must be used at each iteration in equation (3.104). Levenberg proposed a linear search along the *Levenberg-Marquardt trajectory* but this is considered to be a time-consuming operation [46: p.104]. Marquardt devised a simple geometric scheme in which an arbitrary value was used for the

initial λ that was multiplied or divided by powers of a factor υ (10 was suggested) until a decrease in the value of the error function was achieved. Marquardt's approach has received much use although there are practical difficulties associated with it. Fletcher has pointed out [46: p.104] four of these difficulties by considering the cases of poor initial choice of λ, too small increment Δc, the need for too many iterations to increase λ in case of a failure to reduce the value of the error function, and the inability of the algorithm to assure quadratic convergence - all of which contribute to inefficient number of iterations and objective function evaluations.

To overcome the difficulties in Marquardt's algorithm, Fletcher has proposed the next algorithm [52], for which we reserve the name *Fletcher's modification of the Levenberg-Marquardt algorithm*:

STEP 1: given c^j and λ^j, calculate $\nabla E(c)^j$ and $H(c)^j$;

STEP 2: factorize $H(c)^j + \lambda^j I$; If not p.d., reset $\lambda^j = 4\,\lambda^j$ and repeat;

STEP 3: solve equation (3.104) to give Δc^j;

STEP 4: evaluate $E(c^j + \Delta c^j)$ and hence R^j;

STEP 5: if $R^j < 0.25$ set $\lambda^{j+1} = 4\lambda^j$

if $R^j > 0.75$ set $\lambda^{j+1} = \lambda^j / 2$;

otherwise set $\lambda^{j+1} = \lambda^j$;

STEP 6: if $R^j \leq 0$ set $c^{j+1} = c^j$ else $c^{j+1} = c^j + \Delta c^j$.

In the above *Fletcher's modification of the Levenberg-Marquardt algorithm*, $\lambda^0 > 0$ is chosen arbitrarily for the initial iteration. It can therefore be noted that the algorithm is in the spirit of Marquardt.

Fletcher has also considered further modifications on the issue of increasing or decreasing λ in step 5 [43: pp. 206-207; 46: pp. 103-106]. Thus, by defining a multiplying factor q [43: p.206; 46: p.106] for the case when $R^j < 0.25$ which may be put as

$$q = \frac{2[\nabla E^{j^T} \Delta c + E^j - E^{j+1}]}{\nabla E^{j^T} \Delta c}\,, \tag{3.108}$$

Fletcher computes q, sets it equal to 10 if $q > 10$ and to 2 if $q < 2$, and then uses $q\lambda$ in place of λ. For the case when $R^j > 0.75$, the resulting λ obtained after taking $\lambda/2$ was compared against the trace of $(A^TBA)^{-1}$ [46: p.105] and, when the result was less than the trace, the new λ was set to zero for quadratic convergence.

Now, when applied to the modified least p-th Taylor method, every thing in Fletcher's modification of the Levenberg-Marquardt method is retained in the least p-th Taylor method except for the replacement of H by A^TBA instead of A^TA as in the least-squares case. Hence,

equation (3.104) will take the form of equation (3.109) next:

$$\Delta c^j = (p-1)^{-1} (A^{jT} B^j A^j + \lambda I)^{-1} A^{jT} e^j . \qquad (3.109)$$

3.8 Further Damping Techniques

The technique of *component damping* must be implemented to counteract the tendency of floating point overflow that arises because of too large and/or small component values. Component damping may take various forms. One application is to introduce a scalar α ($0 < \alpha < 1$) and then penalize large values of $| \Delta c^j |$. If α is sufficiently small, oscillation is prevented. In practical applications, $\alpha = | c_i^j / 2\Delta c_i^j |$ and an upper bound of $c_i^j / 2$ for Δc_i^j are recommended [27]. The process can therefore be put in an algorithm form as follows:

STEP 1: calculate Δc^j using equation (3.98);

STEP 2: test all Δc_i^js against $c_i^j / 2$ in magnitude;

STEP 3: find Δc^{j+1} if the Δc_i^js are within their limits;

STEP 4: if the Δc_i^js are out of limits, use $\alpha \Delta c_i^j$ in step [3];

An alternative simplification to the above damping technique is to perform the damping by cutting the size of any Δc_i greater in magnitude than $| c_i / 2 |$ back to $| c_i / 2 |$. In this case only the limit-exceeding Δc_i^j's are replaced by $(\Delta c_i^j / | \Delta c_i^j |)(c_i^j / 2)$ and step [4] in the above algorithm is modified accordingly.

3.9 Summary

In this chapter, the theory of electrical network design through computer-aided optimization techniques has further been extended through expositions on network optimization. Thus, at the outset, the design problem was formulated as being one of minimizing a scalar objective function. Sampling was then shown to be necessary to extract this function.

Survey of error criteria commonly used in electrical network design has also been made.

The stages of network analysis and design were linked via sensitivity analysis.

Of the optimum seeking methods studied, gradient methods were shown to be superior and, of these, the modified least p-th Taylor method was selected. This used Fletcher's modification of the Levenberg-Marquardt algorithm and can be made even more effective when used in conjunction with component damping and linear search techniques.

4. COMPUTER PROGRAM DEVELOPMENT

4.1 Basic Considerations

The objective in this chapter is to develop a *Pascal Program for Optimal Network Design*. This shall be called *P-POND* for short. Before proceeding, however, it is customary to consider first some basic software guidelines.

First of all, electrical network design generally requires large amounts of physical memory on a desktop computer or a PC. Programming languages such as BASIC, in which the memory allocation is static, are not suitable for the task. Dynamic memory allocation must be used since larger networks can then be handled using the same physical memory.

Although programs like SPICE [22] handle the memory management problem via a separately written subroutine and continue to use *FORTRAN*, the choice of that language has been abandoned here in favor of *Pascal*. It is also generally believed [37] that Pascal, like the *C language*, gives the best performance for network simulation.

Apart from language selection, other software guidelines for the development of *P-POND* were the notions of *simplicity*, *efficiency* and *generality*. These has been considered as parts of the overall objective of the study.

Since the primary user is the designer, and not a programmer, the program had to be made simple and require no externally made programs for its running. The schematic, *the natural language of network design*, was used at the input stage. Subsequent stages had to be driven by an interactive menu system that was furnished with systematic mechanisms for error control. It was also necessary to document the program, both externally and internally, to make it readable and easy to modify at a later time. *External documentation* refers to the information outside the body of the executable code while *internal documentation* includes comments, program formatting, and self-documenting code. The use of the latter, in particular, emphasizes the functions and relationships of various constants and variables and reduces the need for too many comment lines. Also, comments were not felt necessary on each and every line of statement where the relevant inference could be taken directly from the corresponding Pascal pseudo-codes. The basic program format used in *P-POND* was derived from the convention in [41] which is reproduced as **Appendix D** at the end of this thesis work.

Undue usage of computer memory was avoided through dynamic allocation of memory. As much as possible, all variables resided in memory when needed only. Temporary variables did not claim constant memory area and care was used in the selection of global variables. The algorithms in the program avoided time and space consuming operations. In short, *P-POND* was made as efficient as possible.

The requirement of generality in the design of *P-POND* was considered in light of the overall simplicity and efficiency of the program. Infact, these requirements of *simplicity*, *efficiency*, and *generality* were bound to be contradictory and certain degrees of compromises between them were necessary. The final program, while entertaining a wide range of electrical networks, achieved these objectives at the same time.

4.2 Program Organization

The design and general organization of *P-POND* were dictated by the functions the program had to accomplish. Accordingly, four stages were first identified. These were the *input*, *analysis*, *optimization*, and *output stages*. At *the input stage*, it was found useful to feed all network data and code the result for subsequent uses. In the *analysis phase*, the network whose information had already been available, had to be analyzed using the usual electrical network analyses laws, the theory of graphs, and equation formulation techniques. Then, the network had to be compared against a set of specifications and repeated analyses were necessary at the *optimization stage*. Finally, at the *output stage*, it was necessary to code all results and display them on the screen and, at its best, this had to be done in both graphic and numeric forms. A nice feature here was that all results could be made available for printing as files were used to link the various stages.

Based upon the above functional requirements, *P-POND* was organized into a total of six executable Turbo Pascal programs and four Turbo Pascal units. All files were eventually integrated by the single integrator file $\pi.PAS$.[1,2]

Thus, a total of ten source files [3] made up *P-POND*. These were the integrator and executable program $\pi.PAS$, the five executable programs *INITIAL.PAS*, *WIRING.PAS*, *ANALOPT.PAS*, *GRAPHER.PAS*, *FINAL.PAS*, and the four Turbo Pascal units *ROOT.PAS*, *COMPNNTS.PAS*, *ANAL0.PAS*, and *ANAL1.PAS*. For brevity, we shall henceforth refer to all

---

[1] The designation π (= *PI*) is used here to mean *Program Integrator*.

[2] "π" may be entered from the keyboard by first pressing the shift and Alt keys at the same time and then entering the number 227 from the numeric keypad.

[3] The usage "source file" shall refer to any file containing actual programming codes. This is satisfied by all files ending in the three letter extensions PAS while TPU and EXE files are not considered as source files in this context.

Some network optimizing programs [38], [40] use pre-prepared modules for the optimization part. Here, however, such routines are self-contained as parts of *P-POND* itself in the program *ANALOPT*, which uses units *ANAL1* and *ANAL0*. Unit *ANAL1* also uses unit *ANAL0*. while unit *ANAL0* consists of global variables together with other general purpose utilities for the analysis-optimization stages. Unit ANAL0 is also furnished with network input data. The unit *ANAL1* and the program *ANALOPT* share whatever remains for complete network analysis as set out in the study, with *ANALOPT* further including relevant optimization routines.

In its current form, *P-POND* requires at least 600 Kilo bytes of disk space. The use of self-documenting codes in the actual writing of the program decreased appreciably the total number of comment lines necessary and the complete program could now extend to a little more than 7000 lines of source code. The program has also been run on the IBM PS/2 model 30 286 and model 70 386 machines which have, respectively, 1 and 4 Mega bytes of memory capacities. The disk space requirements of *P-POND* are summarized in table 3.1 below.

Table 4.1: Further Descriptions on the Files of *P-POND*.

No.	Source File	Disk Space	Source File Type	Remarks
1	*INITIAL*	29 546	executable file	PAS + EXE
2	π	7 570	executable file	PAS + EXE
3	*WIRING*	69 881	executable file	PAS + EXE
4	*ANALOPT*	146 427	executable file	PAS + EXE
5	*GRAPHER*	98 848	executable file	PAS + EXE
6	*ROOT*	35 698	Turbo Pascal unit	PAS + TPU
7	*COMPNNTS*	23 360	Turbo Pascal unit	PAS + TPU
8	*ANAL1*	79 871	Turbo Pascal unit	PAS + TPU
9	*ANAL0*	44 939	Turbo Pascal unit	PAS + TPU
10	*FINAL*	20 913	executable file	PAS + EXE
Total Space		***557 053***		

The remarks column in Table 4.1 above indicates the files needed for a complete run. Also, a disk space value for a source file was obtained by summing the disk space requirements of the files under the remarks column for that file.

A complete listing of the ten source files that together made up *P-POND* is given in ***Appendix E*** at the end of this thesis work.

4.3 Program Features

P-POND has the following basic features.

1. USER INTERFACE. The program is furnished with separate menus for its input, analysis, optimization, and plotting stages. Prompts that interact with the user are also provided. A Ctrl-Break action at any one stage breaks that stage and a series of such actions will eventually take to the DOS prompt.

2. NETWORK INPUT. Three input options are available: the schematic, an external ASCII file, and an alphanumeric input from the keyboard. The first and last options are interactive. Networks consisting of passive and/or active components, with the latter represented as VCCSes, must be entered in equivalent network diagram form.

Except for the ASCII file form of input which contains pre-prepared network information, the other two input options are provided with sets of menus with the help of which component types, values, and node connections can be entered. Node information, in particular, must be fed by pressing "N" twice.

For best results, normalization of network components is necessary. But, this purpose is not provided by *P-POND*. Also, the current program has no options for saving, retrieving, and modifying a graphic input. Modification of network information, however, is possible and this is achieved following the steps indicated under item no. 7 below.

3. TRANSFER FUNCTION GENERATION. The voltage gain transfer function is computed in both numeric and symbolic forms. In the former case, this function can be computed to a required number of decimal places, preferably less than twelve. The symbolic transfer function is shown as a general function of the components. Its complete listing, however, may prove awkward in view of the too many terms even for small circuits.

4. SENSITIVITY ANALYSIS. Sensitivity of the voltage gain transfer function to the network components is computed as part of the analysis phase. Frequency response sensitivity is also available but this must be done after performing frequency response analysis.

5. NETWORK INFORMATION. A network is available as .LOD or .CIR extensions of the input file name. The latter is available from the analysis phase and is useful for periodic check up of the connectivity information. Modification of the .CIR file is possible and is done following the procedure indicated under item 7 below.

6. FREQUENCY DOMAIN ANALYSIS. The commands to achieve this are FR and RE. When typing FR, prompts for initial frequency, frequency increment, frequency scale, and total number of frequencies appear. Once these are entered, the program is then ready to make frequency domain analysis through the command RE.

7. COMPONENT MODIFICATIONS. It is possible to modify both the connectivity information and component values as often as wanted. The command to achieve this is EL. As a result of this action, the .LOD file containing coded network information will be updated.

8. NODE MODIFICATIONS. Typing of the command NO from the second of the analysis menus leads to the possibility of changing the input-output nodes.

9. OPTIMIZATION. The optimizer routines of the program depend on the procedures that generate voltage gain transfer function and perform frequency domain and sensitivity analyses. The design specifications may be stored in a .SPC file or entered interactively together with the frequency-weight informations. It is also necessary to supply the desired error level and the p-index for the least p-th Taylor method. The program will then proceed to find a new and optimal set of component values for the selected initial choices. If the computed error level is less than the desired value, the program must be re-run with a higher value of p unless the computed error level is considered satisfactory for all practical purposes. Otherwise, a success message will appear at the bottom of the screen and one more strike of the ENTER key will yield the optimal design values. As a matter of fact, different runs of the optimization process with different sets of initial component values may yield different sets of optimal values, showing the local nature of the solution. The global solution is decided finally by comparing the results of different optimization runs.

10. PLOTTING UTILITIES. The plotting routine searches for the maximum and minimum plotting coordinates out of the available data. There are also default values for the number of x-y divisions. Prompts to accept/change both the maximum-minimum coordinate values and the number of divisions will then appear. It is therefore possible to magnify a restricted portion of any curve. A hard copy of the plots is obtainable by first typing the command *GRAPHICS* from the DOS prompt, i.e., before running *P-POND*, and then using DOS's *Print Screen* facility.

The plotting utilities supported by *P-POND* are the frequency domain plotting utilities

of specified gain, initial gain, final gain, imaginary part of initial phase, real part of initial phase, and initial phase itself. The optimum error is shown in three forms: error versus frequency, error distribution at discrete frequency points, and the error as a continuous function of frequency. Combinations of initial and final gains, optimum and specified gains, as well as initial-final and specified gains may be seen on a single graph. One can also consult helps on both available plotting commands and list of files that contain results of current design session.

Further options for LOOKING the contents of any file from within *P-POND* itself are contained as parts of the plotting routine. It is thus possible to read the contents of the following nineteen files that contain input, analysis and optimization results:

Spc_Gain	*Ini_Gain*	*Opt_Gain*	*Img_Part*
Rea_Part	*Phas_Ini*	*Err_Iter*	*Err_Dist*
Err_Freq	*Ini_SpcG*	*Opt_SpcG*	*FIS_Gain*
InFile.TRL	*InFile.LOD*	*InFile.SYM*	*InFile.BCL*
InFile.CIR	*InFile.SPC*	*InFile.OPT*	

where *InFile* refers to the parent file name holding network information.

11. ERROR CONTROL. The program is furnished with various built-in error control mechanisms. For instance, an accidental string input or entry at a prompt for an integer value will not HALT the program altogether as is normally the case. Instead, the message "-- Entry Error" will be displayed with a caret(^) symbol pointing to the position where an error has first occurred. This approach can thus lend the convenience of retyping the correct data type. There are also options to control correct command usage, undefined answers for a prompt, an attempt to use non-existing file, correct sequence of operations, and a lot more other purposes. The various options are usually accompanied with a warning sound. At the input stage, however, a further option of letting the sound off when not desired is provided.

12. PRINTING UTILITIES. Several files are generated at the end of each compete run of *P-POND*. Of these, those bearing the network name and differing only in their three-letter extensions are important. Thus, there are the .LOD, .CIR, .BCL, .TRL, .SYM, .SPC, and .OPT files. These files contain, respectively, coded network information, uncoded network connection information, branch-component list, trees generated while evaluating the voltage gain transfer function, symbolic voltage gain transfer function information, the design specifications, and the result of each optimization run. It is possible to have a print out of one or more of these files using any word processing facility. Printing options from within *P-POND* itself, however, are not provided in the current program.

5. APPLICATIONS

5.1 Design of an Active Amplifier Equalizer

This chapter contains demonstration on the capabilities of *P-POND*. The basic steps involved in the computer-aided design of electrical networks shall also be uncovered.

Thus, as a first design example, suppose that an active amplifier equalizer is to be designed to meet the response of Fig. 5.1.

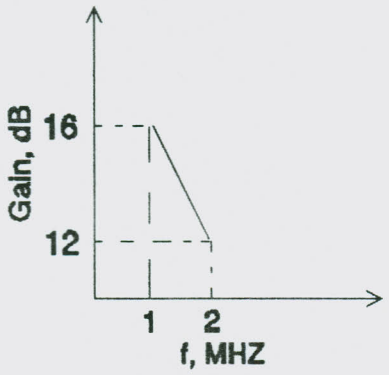


Fig. 5.1: Specified Response.

Then, in equation form, the specified gain is given by:

$$G_{dB} = 20 - 4f_{MHZ} \tag{5.1}$$

The initial step in the design is usually to select a suitable network. One popular op-amp network, the inverter, for instance, gives a flat response as shown in Fig. 5.2.*

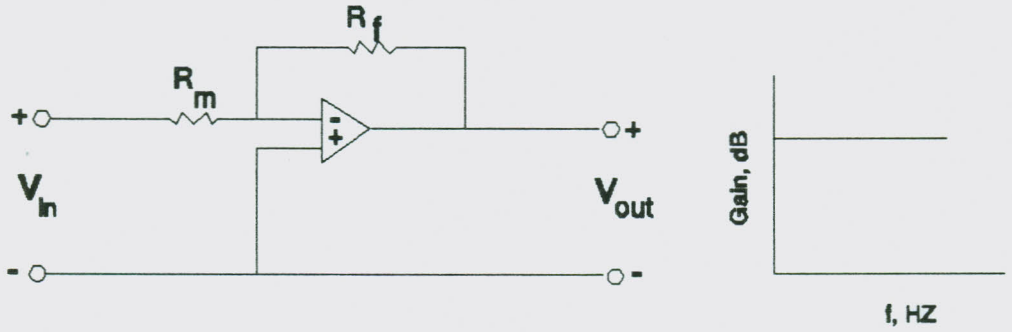


Fig. 5.2: The Inverter as a Constant Gain Network.

The output voltage is given by:

---

* In the literature, R_m is known as *metering* resistor while R_f is called *feedback* resistor.

$$V_{out} = - V_{in} \frac{R_f}{R_m} \tag{5.2}$$

Another popular op-amp network is the integrator and has a frequency response curve whose slope is too much for our purpose:

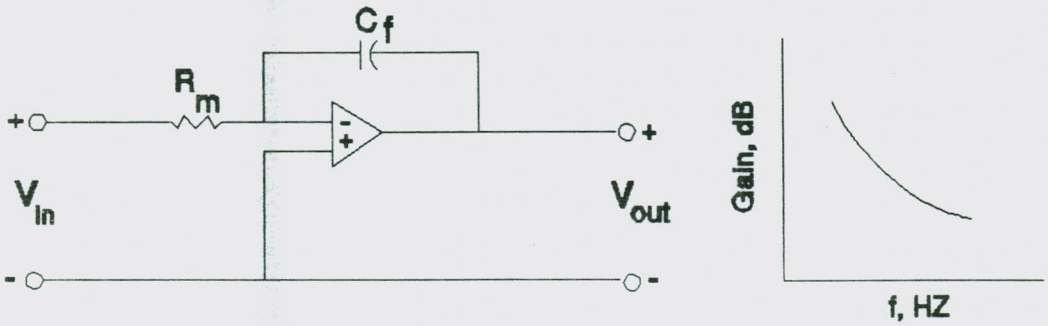


Fig. 5.3: The Integrator Network.

One way to improve the slope of the integrator network is to add a series resistor to C_f. An output proportional to the input and its time integral will then be obtained as in Fig. 5.4:

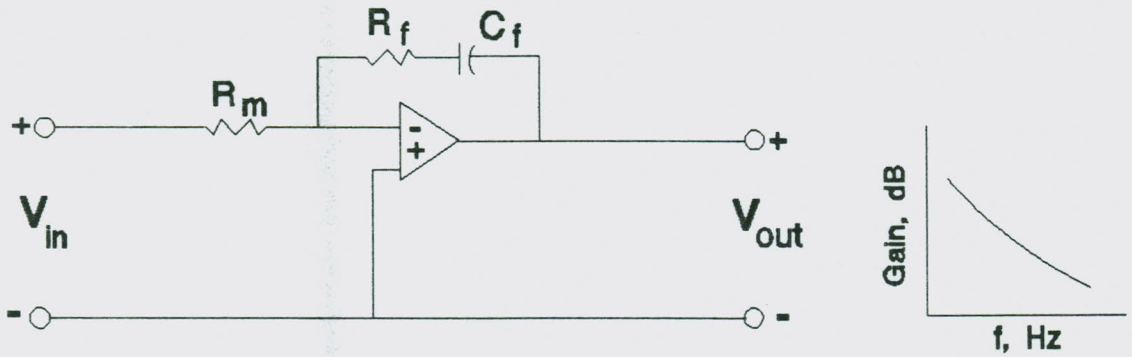


Fig. 5.4: The Augmenting Integrator Network.

Combining inverter and integrator networks as in Fig. 5.4 into a single network results in what is known as the augmenting integrator [55: p.143]. The output voltage in this case is:

$$V_{out} = -\frac{R_f}{R_m} V_{in} - \frac{1}{R_m C_f} \int V_{in} dt \tag{5.3}$$

The augmenting integrator can thus be used to design our active amplifier equalizer. The op-amp and the best value for R_m must also be selected. R_m is usually chosen to minimize the effects of the op-amp's input and output impedances and this is met by taking R_m to be much

smaller than the input impedance and much larger than the output impedance. When selecting the op-amp, on the other hand, such key specifications as input offset voltage, offset voltage drift, input bias current, input offset current, common mode voltage, common mode gain, common mode error, common mode rejection ratio, slew rate, input impedance, output impedance, and gain (frequency response) must usually be considered [56]. But here, suppose that technical and economical conditions have dictated that focus be made on only the latter three, i.e., *output impedance* = 100 Ω; *input impedance* = 10 KΩ; and *voltage gain* = 40 dB. A good choice for R_m will then be a value of 1000Ω. This is ten times the value of the output impedance but only 10% of that of the input impedance.

It remains now to compute for R_f and C_f. This will be done through the use of *P-POND*. But first, basics such as prior analysis, normalization, and conversion of the network to its equivalent form must be ready. Thus, let Fig. 5.4 be redrawn as in Fig. 5.5, where the input and output impedances of the op-amp are shown along with a node numbering scheme:

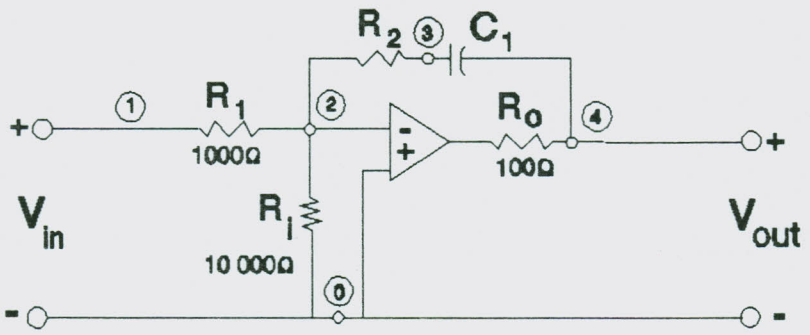


Fig. 5.5: Complete Network for the Active Amplifier Equalizer.

Fig. 5.5 can be simplified by neglecting the input and output impedances. Also, let w_0, z_0, c_0, and g_0 define the frequency, impedance, capacitance, and VCCS scale factors. Then,

$$c_0 = \frac{1}{w_0 z_0} \tag{5.4}$$

$$g_0 = \frac{1}{z_0} \tag{5.5}$$

Let $w_0 = 10^6$ and $z_0 = 10^3$. Then, $c_0 = 10^{-9}$ and $g_0 = 10^{-3}$. These factors are useful for

the purpose of dividing the respective actual component values.*

The equivalent network of Fig. 5.5 is shown in Fig. 5.6 below.

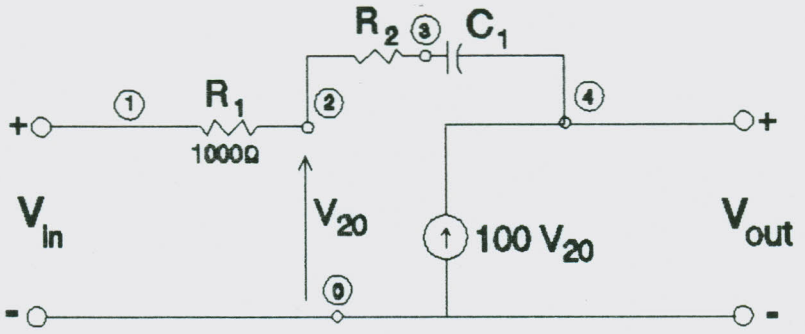


Fig. 5.6: Equivalent Network of the Active Amplifier Network.

Fig. 5.6 is redrawn in Fig. 5.7 with normalized values and with the assumptions of 100Ω and 100F for R_2 and C_1.**

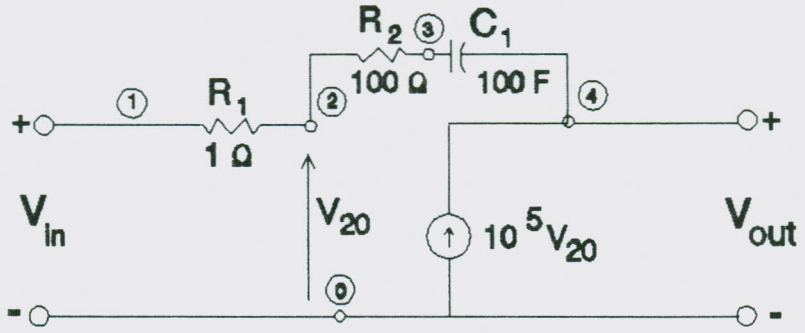


Fig. 5.7: Gain Equalizer in Normalized Values.

The use of π from the DOS prompt will now yield a set of message screens that eventually lead to the *SCHEMATIC CAPTURE* stage. Here, network information is fed to the computer as a set of nodes, components, values and connections. The result is stored in a .LOD extension of the input file. There is also an option to bypass the *SCHEMATIC CAPTURE* stage

---

* At the end of the optimization process, the reverses of these steps shall be carried out. Also, a VCCS value has been used where the actual variable to consider was the voltage gain (40 dB or a gain of 100 in the example). Since the other contributing term to the VCCS value is only a multiplying factor, this approach does not upset the result in any way.

** These are initial guesses for the indicated component values.

and this takes to the further options of data input either from the keyboard or a file. On the other hand, all input forms are equivalent.

Suppose next that the *SCHEMATIC CAPTURE* stage was not bypassed and an input file name of SOULG has been utilized. Then, the contents of file SOULG.LOD would read:

```
4 2 0 1 1
      1   1   1   2   1.00000E+0000
      1   2   2   3   1.00000E+0002
      3   1   3   4   1.00000E+0002
     -1   1   0   4   1.00000E+0005   2   0,
```

where

- the first line shows the total numbers of nodes, Rs, Ls, and Cs, in that order;
- lines 2 - 4 refer to passive components with columns 1 - 5 indicating, respectively, component type, number, plus node, minus node, and value. Passive components are identified by the numbers 1, 2, and 3. The number 1 represents a resistor, 2 an inductor, and 3 a capacitor;
- line five in the above .LOD file refers to the active component or VCCS with the -1 indicating the type of the component as Gm; the second column refers to the VCCS number; the third and fourth columns to the VCCS current receiving - + nodes; column five to the VCCS value; and columns six and seven to the VCCS voltage sending + - nodes.

An alternative to the .LOD file was found in the .CIR file. The two files are basically identical except that line 1 of the .LOD file did not appear in the .CIR file. The approach resulted in easily understood codes for the network connection.

The contents of the .CIR file were:

```
R1   1   2    1.00000E+0000
R2   2   3    1.00000E+0002
C1   3   4    1.00000E+0002
G01  2   0   4   0    1.00000E+0005.
```

In the .CIR file for passive components, columns two and three contained the + - nodes while the last column in each row was reserved for component values. For active components, on the other hand, columns two to four were filled with the node numbers for voltage sending

+ - nodes and current receiving + - nodes. Here again, the last column for each VCCS contained the VCCS value for that row while column one in each VCCS description was made up of three things: the letter G which showed that the component being considered was active, one of 0,B, or S, and an integer indicating component number. A code of 0 was used to represent a VCCS type of Gm, B for Gm/s, and S for sGm.

Proceeding further, the transfer function to three decimal places was obtained as:

$$\frac{1000000100.000S\uparrow 1 + 100000.000}{-9999900.000S\uparrow 1} \qquad (5.6)$$

The up-arrow in equation (5.6) signifies exponentiation and s is the usual complex frequency variable. When computed in numeric form, the numerator and denominator symbolic terms were found as follows:

Numerator Symbolic Terms
S^0 -G01
S^1 +R2C1G01 +C1
Denominator Symbolic Terms
S^0
S^1 -R1C1G01 +C1

i.e., the symbolic transfer function was:

$$\frac{(R_2C_1G_{01} + C_1)s - G_{01}}{(-R_1C_1G_{01} + C_1)s} \qquad (5.7)$$

The use of five frequency points in steps of 0.25 from 1 to 2 gave the next frequency response information:

FREQ., HZ	GAIN, DB	PHASE, DEG.	REAL PART	IMAG. PART
1.000E+0000	40.0001	-0.0009	-1.000E+0002	1.592E-0003
1.250E+0000	40.0001	-0.0009	-1.000E+0002	1.273E-0003
1.500E+0000	40.0001	-0.0009	-1.000E+0002	1.061E-0003
1.750E+0000	40.0001	-0.0009	-1.000E+0002	9.095E-0004
2.000E+0000	40.0001	-0.0009	-1.000E+0002	7.958E-0004

Sensitivity analyses were done on the magnitude and phase of the voltage gain transfer function as well as the real and imaginary parts of the gain. Each component was considered separately at each of the five frequency sample points:

CPT NO.	DB SEN	DEG SEN	RP SEN	IP SEN
R1	-8.686E+0000	0.000E+0000	1.000E+0002	-1.592E-0003
R2	8.686E-0002	9.119E-0006	-1.000E+0000	0.000E+0000
C1	-2.200E-0011	9.119E-0006	1.158E-0019	-1.592E-0005
GM1	-8.773E-0010	-9.119E-0016	1.010E-0008	-1.592E-0013
R1	-8.686E+0000	0.000E+0000	1.000E+0002	-1.273E-0003
R2	8.686E-0002	7.295E-0006	-1.000E+0000	0.000E+0000
C1	-1.408E-0011	7.295E-0006	-0.000E+0000	-1.273E-0005
GM1	-8.773E-0010	-7.295E-0016	1.010E-0008	-1.273E-0013
R1	-8.686E+0000	-3.882E-0023	1.000E+0002	-1.061E-0003
R2	8.686E-0002	6.079E-0006	-1.000E+0000	0.000E+0000
C1	-9.778E-0012	6.079E-0006	-7.720E-0020	-1.061E-0005
GM1	-8.773E-0010	-6.079E-0016	1.010E-0008	-1.061E-0013
R1	-8.686E+0000	0.000E+0000	1.000E+0002	-9.095E-0004
R2	8.686E-0002	5.211E-0006	-1.000E+0000	0.000E+0000
C1	-7.184E-0012	5.211E-0006	-0.000E+0000	-9.095E-0006
GM1	-8.773E-0010	-5.211E-0016	1.010E-0008	-9.095E-0014
R1	-8.686E+0000	0.000E+0000	1.000E+0002	-7.958E-0004
R2	8.686E-0002	4.559E-0006	-1.000E+0000	0.000E+0000
C1	-5.500E-0012	4.559E-0006	1.158E-0019	-7.958E-0006
GM1	-8.773E-0010	-4.559E-0016	1.010E-0008	-7.958E-0014

As mentioned earlier, optimization must be preceded by sampling using a pre-created ASCII file or by entering values directly from the keyboard. Equation (5.1) was used and both the sampled desired response and the corresponding weights were stored in the file SOULG.SPC as follows:

16.00 1
15.00 1
14.00 1
13.00 1
12.00 1

And, for convenience, a plot of the above data is also reproduced next:

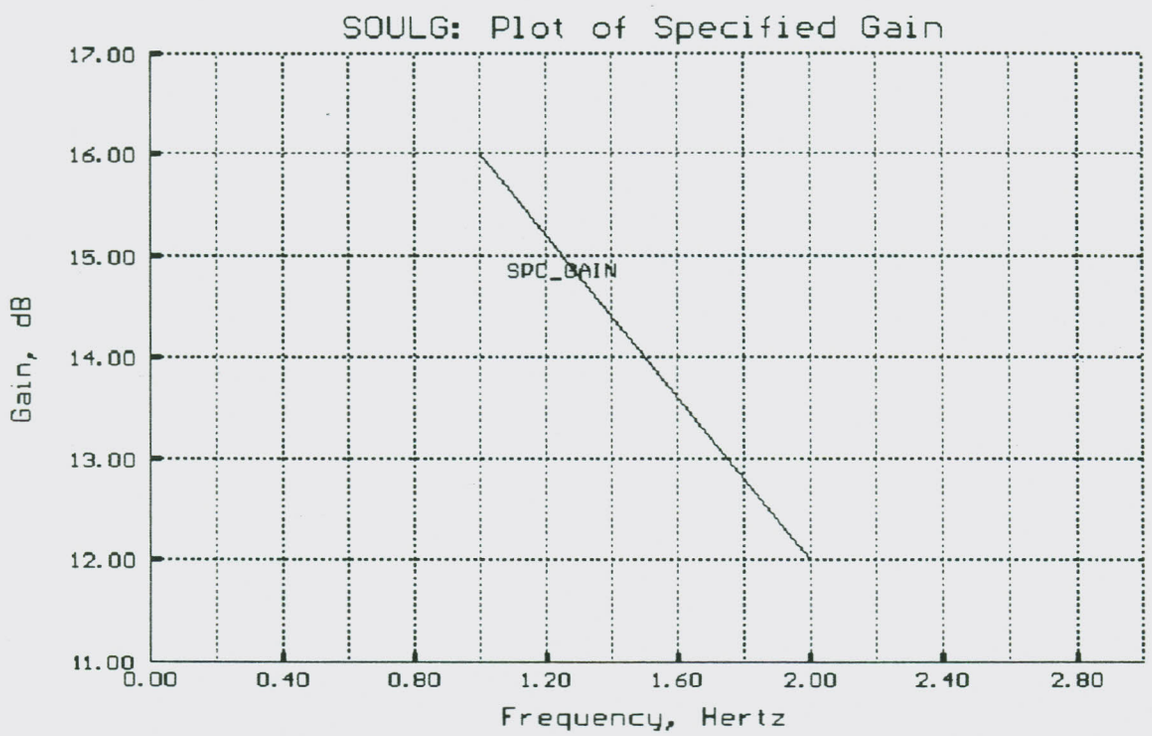


Fig. 5.8: Plot of Specified Response.

Optimization using *P-POND* also required that the network transfer function and frequency response be first computed. When these stages are passed with success, an *OPTIMIZATION MENU* will appear on the screen. This contains prompts for the number and type of components to be optimized, the frequency information, sampling information, least-p level to be used and the error level. The least-p values to be used are 2, 4, 6, etc. and it is a good practice to start with the lowest value and progress to higher values at the succeeding stages using the Y/N prompt at the bottom of the screen.

In the present case, an error level of zero and the least 4-th error criterion were used to optimize the active gain equalizer network. The results obtained are summarized in the file SOULG.OPT and reproduced next:

```
**************************************************
        Optimization results ...

   I    Name of input file  :  SOULG
        Number of nodes     :  4
        Number of branches  :  4
        Total R, L, C, Gm   :  2,0,1,1

   II   NUMBER OF VARIABLES :  2
        LEAST p LEVEL USED  :  4
        TOTAL SAMPLE POINTS :  5
        DESIRED ERROR LEVEL :  0.00000000E+0000
        ATTAINED ERROR LEVEL:  2.01254299E-0002
        TOTAL ITERATIONS    :  21

   III  Optimized component values:
         R1   1.00000000000000E+0000
         R2   2.88418953474650E+0000
         C1   2.72749800412434E-0002
         GM1  1.00000000000000E+0005
**************************************************
```

As shown in the listing above, the network has been optimized with an error of about 0.02. A total of 21 iterations were necessary. The results obtained were, to four decimal places and in normalized units, R1 = 1.0000, R2 = 2.8842, C1 = 0.0273, and Gm1 = 100000.000. Hence, the optimum values are R1 = 1000Ω, R2 = 288.42Ω, C1 = 27.3μF, and Gain = 100.* On the other hand, the desired error level of zero could not be attained using the least 4-th Taylor method and increasing the p-level was necessary to arrive at a better set of results.

---

* These latter values could be obtained through *denormalization*.

We now turn to graphical interpretations of the various results so far.

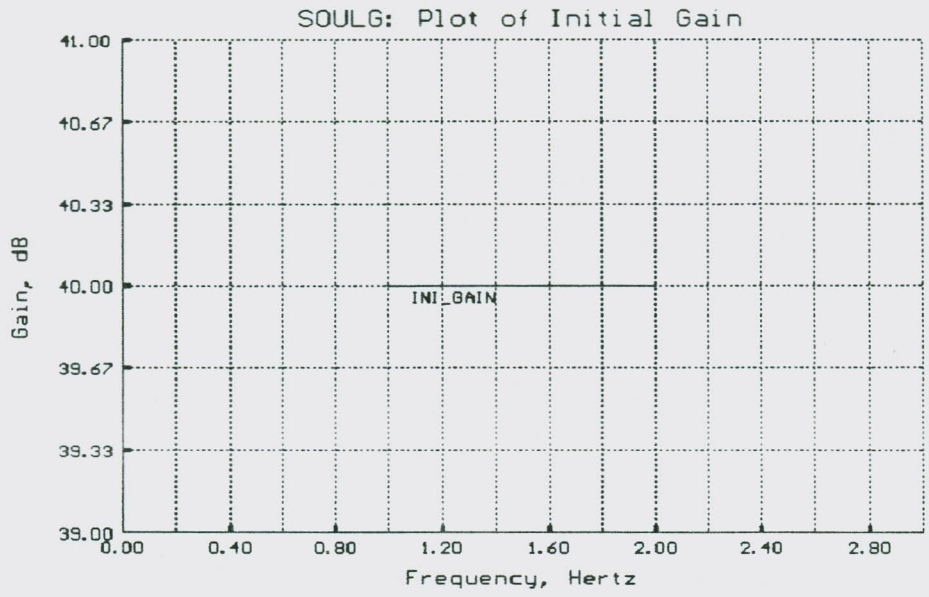


Fig. 5.9: Initial Gain versus Frequency.

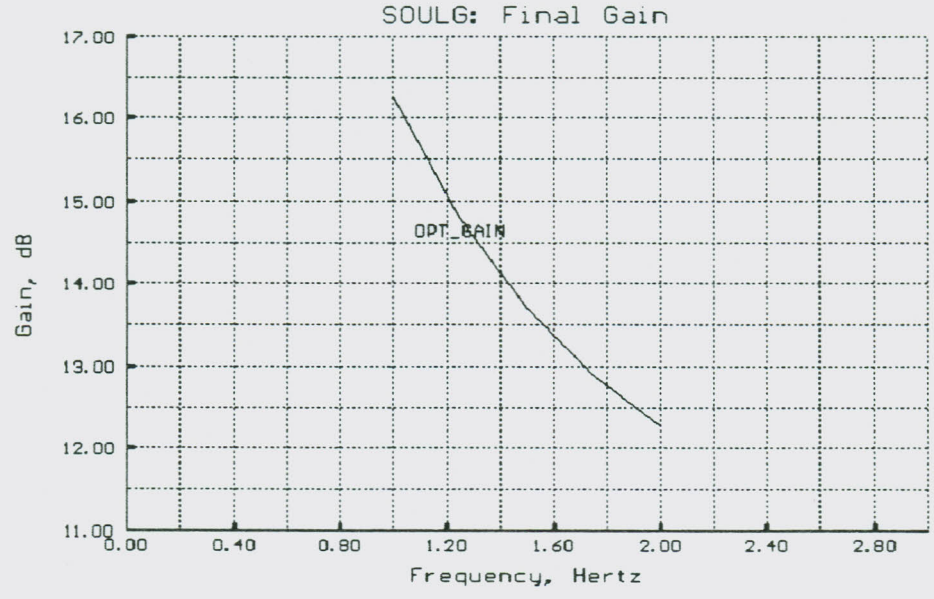


Fig. 5.10: Final Gain Versus Frequency.

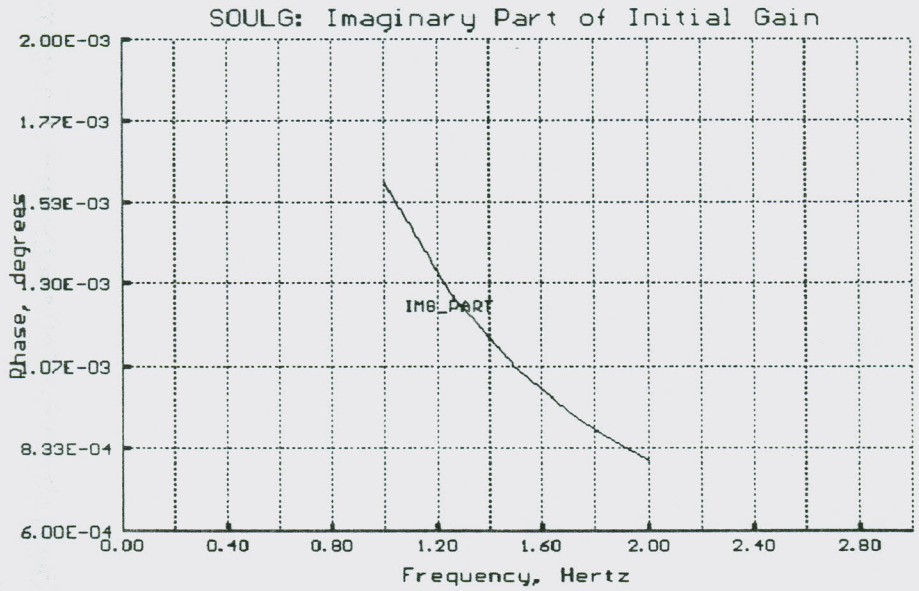


Fig. 5.11: Imaginary Part of Initial Gain.

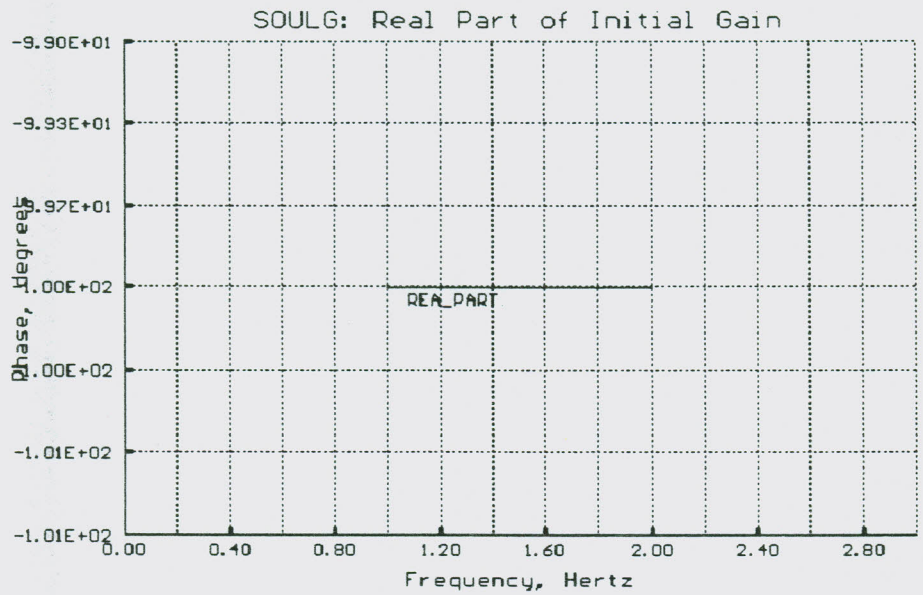


Fig. 5.12: Real Part of Initial Gain.

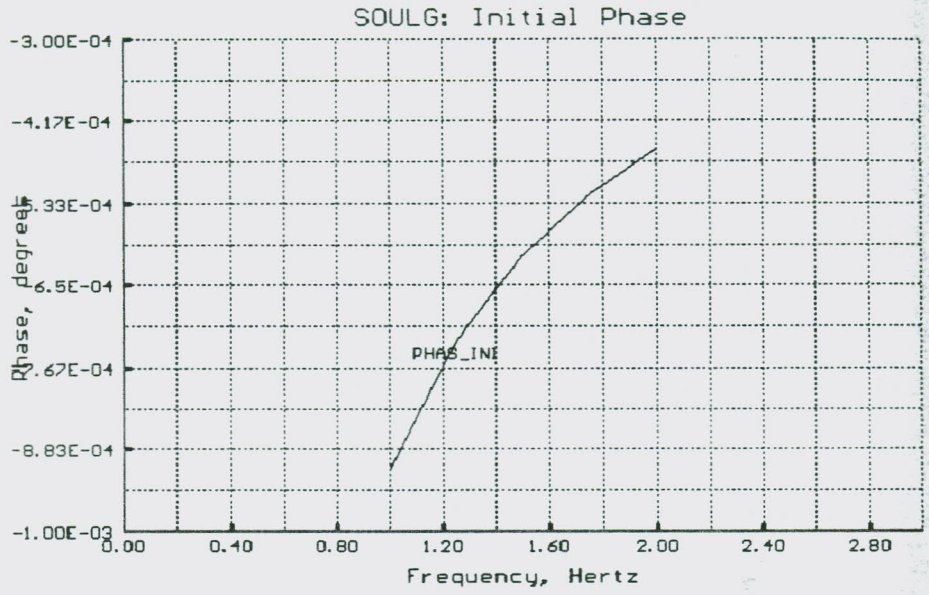


Fig. 5.13: Initial Phase.

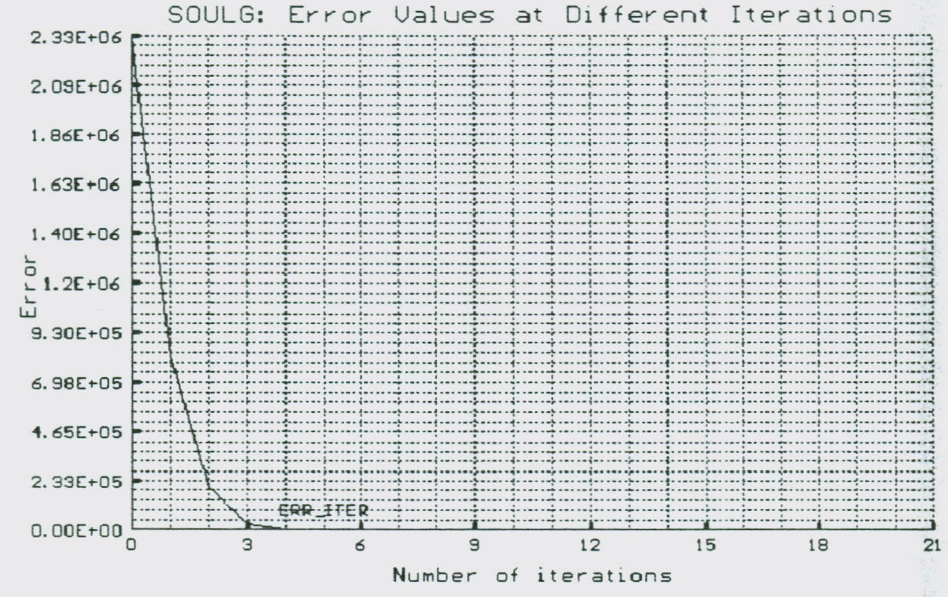


Fig. 5.14: The Error Function at Different Iterations.

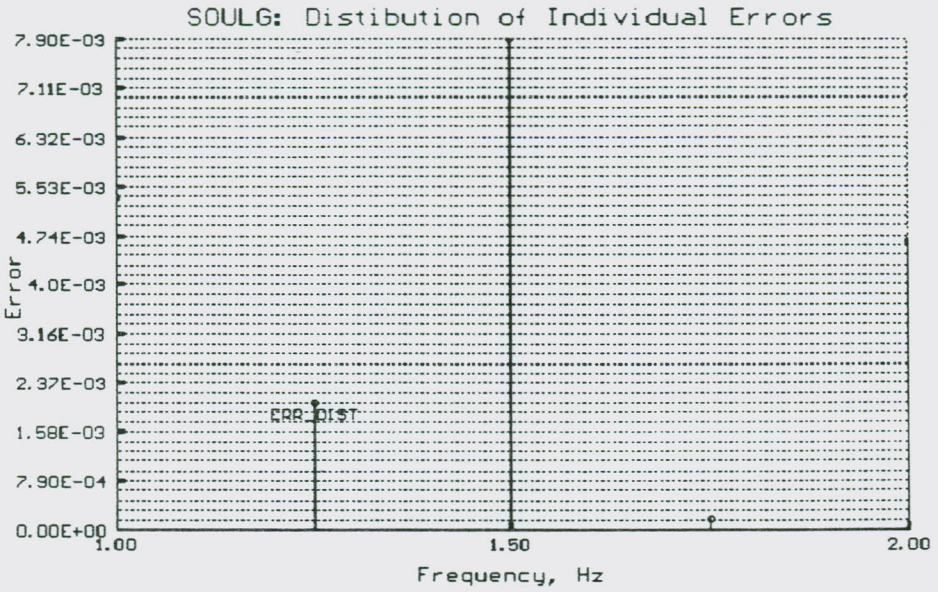


Fig. 5.15: Distribution of Error Values at the Final Iteration.

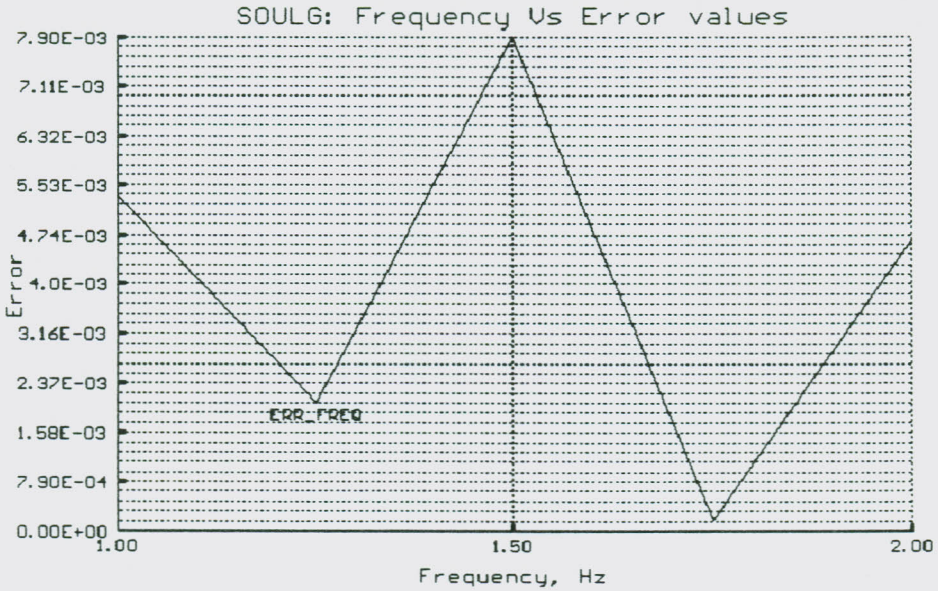


Fig. 5.16: Final Error Values as Functions of Frequency.

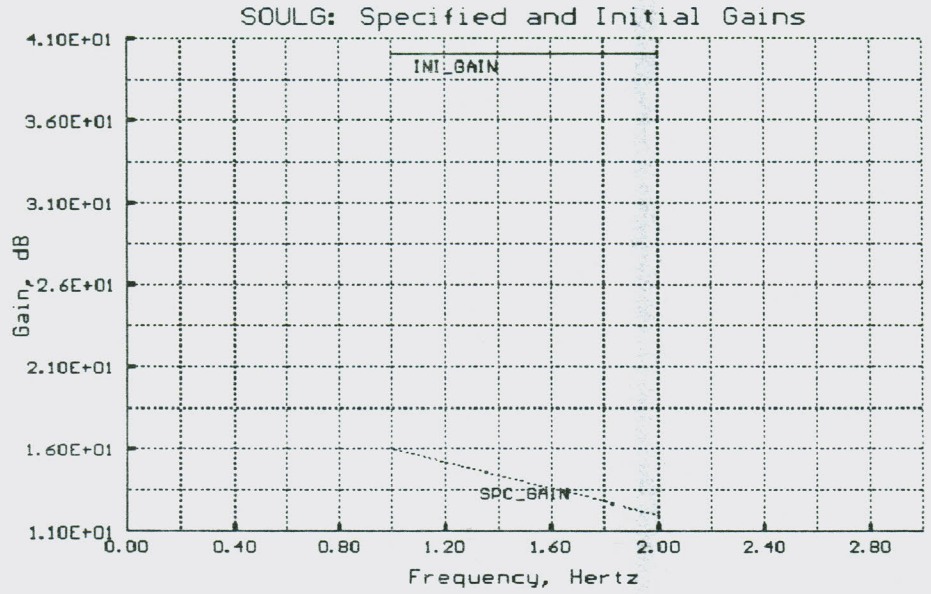


Fig. 5.17: Initial and Specified Gains.

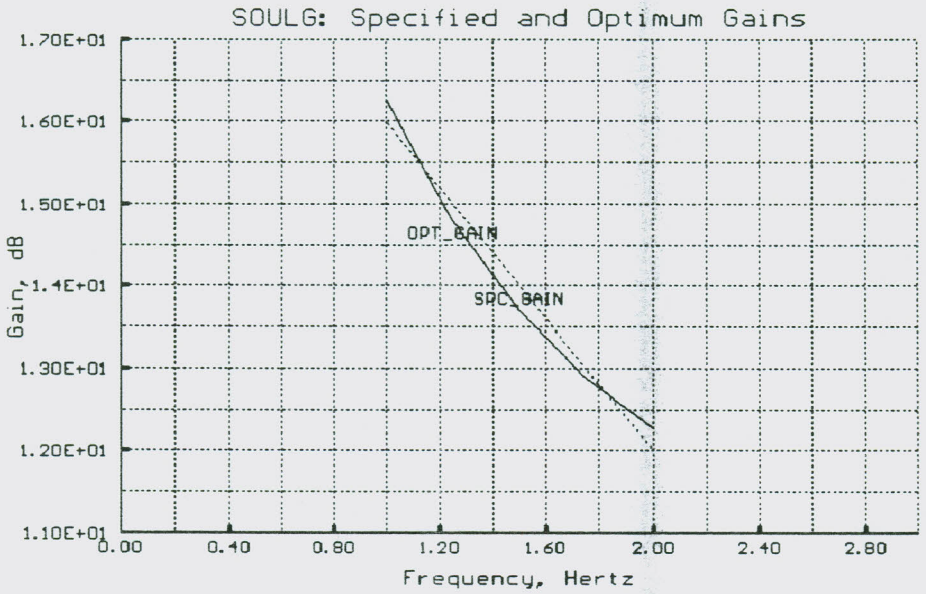


Fig. 5.18: Optimum and Specified Gains.

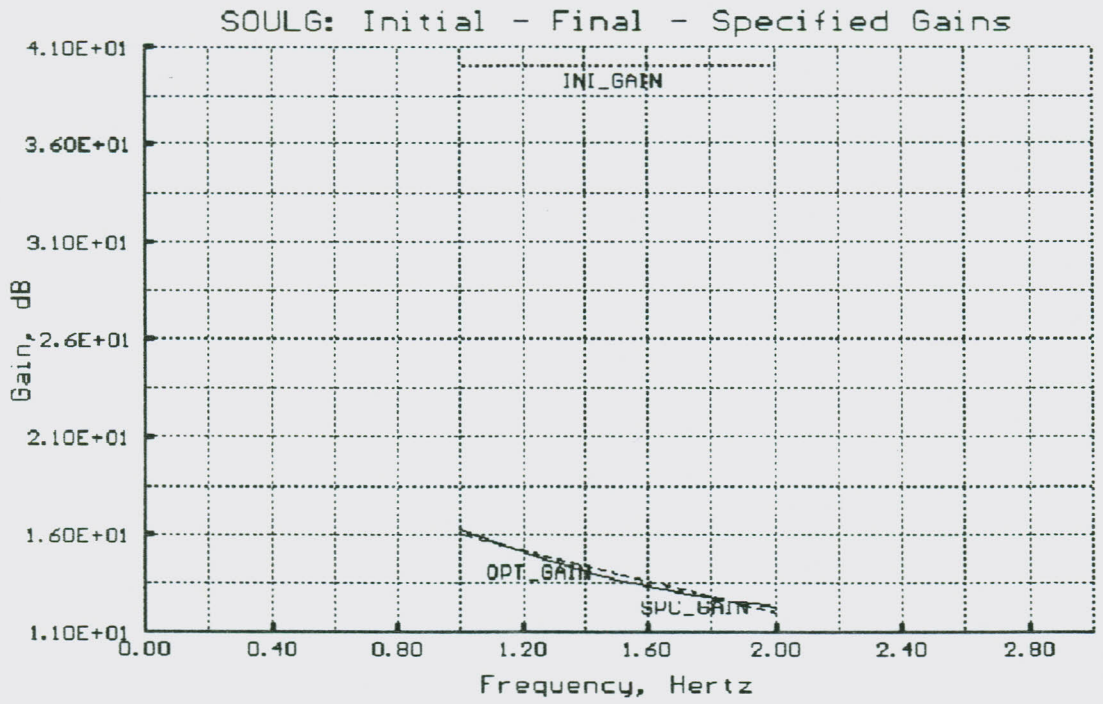


Fig. 5.19: Simultaneous plots of Final, Initial, and Specified Gains.

Some notes concerning the plots of Fig. 5.9 - 5.19 are now in order. First of all, the plots in Figs. 5.17 - 5.19 contain comparisons of initial, final and specified gains. In particular, the combined plots in Fig. 5.19 indicate that the final gain was a reasonable approximation to the specified gain. The *error analyses plots* of Figs. 5.14 - 5.16, where the downhill property of the error is illustrated, also justify this fact. These latter plots additionally show both the diminished magnitudes and a more or less even distribution of the errors at the final iteration.

The separate plot of Fig. 5.10 for the final or optimum gain was included to observe the function more closely by a change of scale at will. This approach could once more show that this function was a good approximation of the specified gain.

The four plots of Figs. 5.9, 5.11, 5.12 and 5.13 summarize the results obtained for the initial optimization stage. The initial phase has also been included although *P-POND*, in its current form, and does not perform network design based on phase specifications.

5.2 Design of a Passive Parallel Ladder Network

Suppose next that we want to design an electrical network that meets the voltage gain transfer function

$$F(s) = \frac{1}{15}\left(\frac{s^2 + s + 1}{s^2 + 4s + 3}\right) . \tag{5.8}$$

in the normalized frequency range of 0 to 1 Hertz.

Then, *Guillemin's parallel ladder network* [17: p.328; 57: p.561] may be used. This is an example of a passive network and shall be our second design example. The result of drawing the network using *P-POND* is indicated in Fig. 5.20, where the nodes have been numbered to conform to the scheme indicated in [17]:

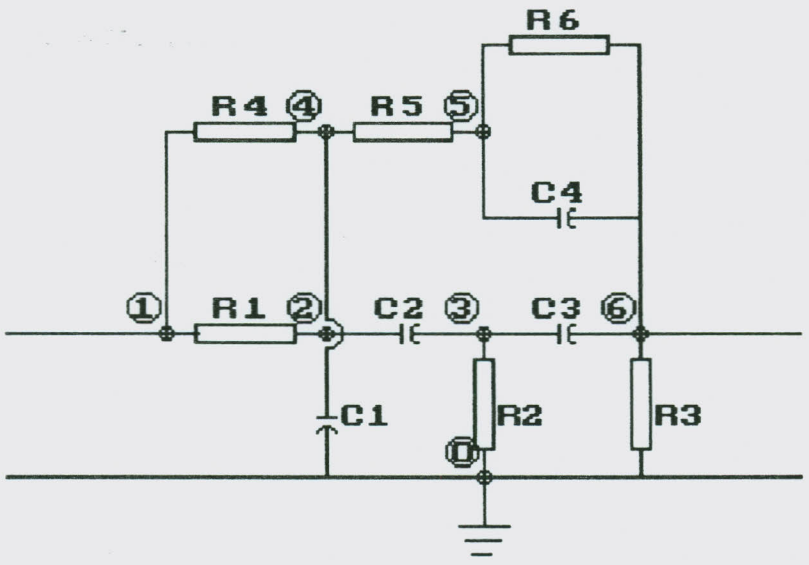


Fig. 5.20: Guillemin's Parallel Ladder Network.

Evaluating equation (5.8) at s = jf* and then taking the magnitude of the resulting relation, we have the magnitude response for exact component values as

$$G_{dB} = \frac{20}{\ln(10)} \ln\left(\frac{\sqrt{f^8 + 9f^6 + f^2 + 9}}{f^4 + 10f^2 + 9}\right) \tag{5.9}$$

---

* In the actual sampling process, f shall be multiplied by 2π and so the substitution here shall be equivalent to letting s = jω.

where f is frequency in units of Hertz.

Now, as usual, initial component values, input-output nodes, voltage gain transfer function, number of components to be optimized, frequencies of interest, information concerning the sampled and prescribed magnitude response, level of p to be used for the least p-th Taylor method, and error level to be maintained must all be supplied to carry out the design through the method of this study. P.R. Adby [17] has analysed the network using the next component values which were rounded to three significant figures, with the resistors in Ωs and the capacitors in Farads:

$$R1 = 15;$$
$$R2 = 1.84;$$
$$R3 = 2.73;$$
$$R4 = 7.5;$$
$$R5 = 45;$$
$$R6 = 67.5;$$
$$C1 = 0.0444;$$
$$C2 = 0.0212;$$
$$C3 = 0.214; \text{ and,}$$
$$C4 = 0.0148.$$

The magnitude response for the above component values is shown in Fig. 5.21:

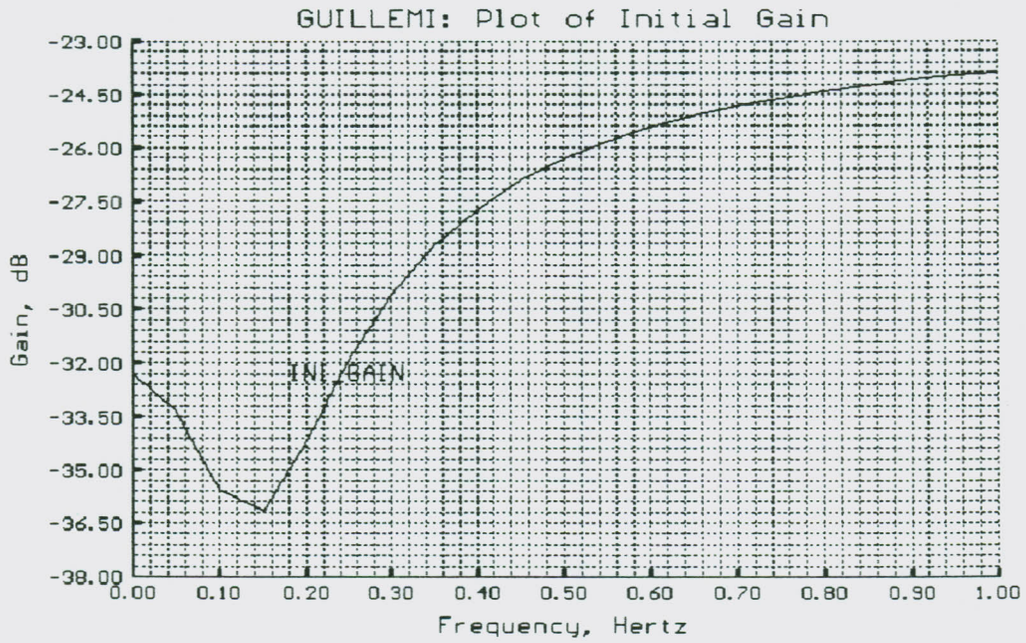


Fig. 5.21: Response for the Guillemin Parallel Ladder Network.

The design in this case shall be exemplified by using *P-POND* to refine Adby's approximate values. For this purpose, let equation (5.9) be sampled at 21 frequency points from 0 to 1 Hertz in steps of 0.05 with R2, R3, R4, R5, and R6 chosen as variables. The contents of GUILLEMI.LOD were:

```
6 6 0 4 0
1 1 1 2 15
1 2 3 0 2
1 3 6 0 3
1 4 1 4 8
1 5 4 5 45
1 6 5 6 68
3 1 4 0 0.0444
3 2 2 3 0.0212
3 3 3 6 0.214
3 4 5 6 0.0148;
```

while the corresponding .CIR file read:

```
R1   1   2    1.50000E+0001
R2   3   0    2.00000E+0000
R3   6   0    3.00000E+0000
R4   1   4    8.00000E+0000
R5   4   5    4.50000E+0001
R6   5   6    6.80000E+0001
C1   4   0    4.44000E-0002
C2   2   3    2.12000E-0002
C3   3   6    2.14000E-0001
C4   5   6    1.48000E-0002.
```

The transfer function using the above values was, to three significant decimal places,

$$TF_{NUMERIC} = \frac{0.438S^4 + 2.955S^3 + 6.082S^2 + 5.384S + 3.000}{6.057S^4 + 58.762S^3 + 198.962S^2 + 273.005S + 124.000}$$

The voltage transfer function is thus of fourth order in both of the numerator and denominator polynomials. Comparing this with the exact transfer function in equation (5.8), equation (5.9) must contain a common factor which numerically cancels when component values are exact.

Next, before proceeding, we also produce the symbolic transfer function terms that are contained in the file GUILLEMI.SYM as in the next columns:

denom 2 +R4R6C1C2
denom 2 +R3R4C1C2
denom 3 +R3R4R6C1C2C3
denom 3 +R3R4R6C1C2C4
denom 2 +R4R5C1C2
denom 3 +R3R4R5C1C2C3
denom 3 +R4R5R6C1C2C4
denom 4 +R3R4R5R6C1C2C3C4
denom 1 +R4C2
denom 2 +R3R4C2C3
denom 2 +R4R6C2C4
denom 3 +R3R4R6C2C3C4
denom 1 +R6C2
denom 1 +R3C2
numer 1 +R3C2
denom 2 +R3R6C2C3
denom 2 +R3R6C2C4
numer 2 +R3R6C2C4
denom 1 +R5C2
denom 2 +R3R5C2C3
denom 2 +R5R6C2C4
denom 3 +R3R5R6C2C3C4
denom 3 +R2R4R6C1C2C3
denom 3 +R2R3R4C1C2C3
denom 4 +R2R3R4R6C1C2C3C4
denom 3 +R2R4R5C1C2C3
denom 4 +R2R4R5R6C1C2C3C4
denom 2 +R2R4C2C3
denom 3 +R2R4R6C2C3C4
denom 2 +R2R6C2C3
denom 2 +R2R3C2C3
numer 2 +R2R3C2C3
denom 3 +R2R3R6C2C3C4
numer 3 +R2R3R6C2C3C4

denom 2 +R2R5C2C3
denom 3 +R2R5R6C2C3C4
denom 1 +R4R6C1
denom 1 +R3R4C1
denom 2 +R3R4R6C1C3
denom 2 +R3R4R6C1C4
denom 1 +R4R5C1
denom 2 +R3R4R5C1C3
denom 2 +R4R5R6C1C4
denom 3 +R3R4R5R6C1C3C4
denom 0 +R4
denom 1 +R3R4C3
denom 1 +R4R6C4
denom 2 +R3R4R6C3C4
denom 0 +R6
denom 0 +R3
numer 0 +R3
denom 1 +R3R6C3
denom 1 +R3R6C4
numer 1 +R3R6C4
denom 0 +R5
denom 1 +R3R5C3
denom 1 +R5R6C4
denom 2 +R3R5R6C3C4
denom 2 +R2R4R6C1C2
denom 2 +R2R3R4C1C2
denom 3 +R2R3R4R6C1C2C3
numer 3 +R2R3R4R6C1C2C3
denom 3 +R2R3R4R6C1C2C4
denom 2 +R2R4R5C1C2
denom 3 +R2R3R4R5C1C2C3
numer 3 +R2R3R4R5C1C2C3
denom 3 +R2R4R5R6C1C2C4
denom 4 +R2R3R4R5R6C1C2C3C4

numer 4 +R2R3R4R5R6C1C2C3C4
denom 1 +R2R4C2
denom 2 +R2R3R4C2C3
numer 2 +R2R3R4C2C3
denom 2 +R2R4R6C2C4
denom 3 +R2R3R4R6C2C3C4
numer 3 +R2R3R4R6C2C3C4
denom 1 +R2R6C2
denom 1 +R2R3C2
numer 1 +R2R3C2
denom 2 +R2R3R6C2C3
numer 2 +R2R3R6C2C3
denom 2 +R2R3R6C2C4
numer 2 +R2R3R6C2C4
denom 1 +R2R5C2
denom 2 +R2R3R5C2C3
numer 2 +R2R3R5C2C3
denom 2 +R2R5R6C2C4
denom 3 +R2R3R5R6C2C3C4
numer 3 +R2R3R5R6C2C3C4
denom 2 +R2R4R6C1C3
denom 2 +R2R3R4C1C3
denom 3 +R2R3R4R6C1C3C4
denom 2 +R2R4R5C1C3
denom 3 +R2R4R5R6C1C3C4
denom 1 +R2R4C3
denom 2 +R2R4R6C3C4
denom 1 +R2R6C3
denom 1 +R2R3C3
numer 1 +R2R3C3
denom 2 +R2R3R6C3C4
numer 2 +R2R3R6C3C4
denom 1 +R2R5C3
denom 2 +R2R5R6C3C4

In the listing just had, each line represented a coding scheme for a denominator or numerator transfer function term. As an example, take the description "denom 2 +R4R6C1C2". This was used to indicate that term $R_4R_6C_1C_2$ shall add to the coefficient of S^2 in the denominator. All other terms were similarly assembled by *P-POND* itself and the resulting completely assembled symbolic transfer function [and not just the listing above] was displayed on the screen. Then, after minor modifications on the *P-POND* screen output, the symbolic transfer function became:

$$TF_{SYMBOLIC} = \frac{\begin{array}{c}(R_2R_3R_4R_5R_6C_1C_2C_3C_4)S^4 + (R_2R_3R_6C_2C_3C_4 + \\ R_2R_3R_4R_6C_1C_2C_3 + R_2R_3R_4R_5C_1C_2C_3 + \\ R_2R_3R_4R_6C_2C_3C_4 + R_2R_3R_5R_6C_2C_3C_4)S^3 + \\ (R_3R_6C_2C_4 + R_2R_3C_2C_3 + R_2R_3R_4C_2C_3 + R_2R_3R_6C_2C_3 + R_2R_3R_6C_2C_4 + R_2R_3 \\ R_2R_3R_6C_3C_4)S^2 + (R_3C_2 + R_3R_6C_4 + R_2R_3C_2 + R_2R_3C_3)S + R_3\end{array}}{\begin{array}{c}(R_3R_4R_5R_6C_1C_2C_3C_4 + R_2R_3R_4R_6C_1C_2C_3C_4 + \\ R_2R_4R_5R_6C_1C_2C_3C_4 + R_2R_3R_4R_5R_6C_1C_2C_3C_4)S^4 + \\ (R_3R_4R_6C_1C_2C_3 + R_3R_4R_6C_1C_2C_4 + R_3R_4R_5C_1C_2C_3 + \\ R_4R_5R_6C_1C_2C_4 + R_3R_4R_6C_2C_3C_4 + R_3R_5R_6C_2C_3C_4 + \\ R_2R_4R_6C_1C_2C_3 + R_2R_3R_4C_1C_2C_3 + R_2R_4R_5C_1C_2C_3 + \\ R_2R_4R_6C_2C_3C_4 + R_2R_3R_6C_2C_3C_4 + R_2R_5R_6C_2C_3C_4 + \\ R_3R_4R_5R_6C_1C_3C_4 + R_2R_3R_4R_6C_1C_2C_3 + R_2R_3R_4R_6C_1C_2C_4 + \\ R_2R_3R_4R_5C_1C_2C_3 + R_2R_4R_5R_6C_1C_2C_4 + R_2R_3R_4R_6C_2C_3C_4 + \\ R_2R_3R_5R_6C_2C_3C_4 + R_2R_3R_4R_6C_1C_3C_4 + R_2R_4R_5R_6C_1C_3C_4)S^3 + \\ (R_4R_6C_1C_2 + R_3R_4C_1C_2 + R_4R_5C_1C_2 + R_3R_4C_2C_3 + R_4R_6C_2C_4 + \\ R_3R_6C_2C_3 + R_3R_6C_2C_4 + R_3R_5C_2C_3 + R_5R_6C_2C_4 + R_2R_4C_2C_3 + \\ R_2R_6C_2C_3 + R_2R_3C_2C_3 + R_2R_5C_2C_3 + R_3R_4R_6C_1C_3 + R_3R_4R_6C_1C_4 + \\ R_3R_4R_5C_1C_3 + R_4R_5R_6C_1C_4 + R_3R_4R_6C_3C_4 + R_3R_5R_6C_3C_4 + \\ R_2R_4R_6C_1C_2 + R_2R_3R_4C_1C_2 + R_2R_4R_5C_1C_2 + \\ R_2R_3R_4C_2C_3 + R_2R_4R_6C_2C_4 + R_2R_3R_6C_2C_3 + R_2R_3R_6C_2C_4 + \\ R_2R_3R_5C_2C_3 + R_2R_5R_6C_2C_4 + R_2R_4R_6C_1C_3 + R_2R_3R_4C_1C_3 + \\ R_2R_4R_5C_1C_3 + R_2R_4R_6C_3C_4 + R_2R_3R_6C_3C_4 + R_2R_5R_6C_3C_4)S^2 + \\ (R_4C_2 + R_6C_2 + R_3C_2 + R_5C_2 + R_4R_6C_1 + R_3R_4C_1 + R_4R_5C_1 + \\ R_3R_4C_3 + R_4R_6C_4 + R_3R_6C_3 + R_3R_6C_4 + R_3R_5C_3 + R_5R_6C_4 + \\ R_2R_4C_2 + R_2R_6C_2 + R_2R_3C_2 + R_2R_5C_2 + R_2R_4C_3 + R_2R_6C_3 + \\ R_2R_3C_3 + R_2R_5C_3)S + (R_4+R_6+R_3+R_5)\end{array}}$$

Thus, there were a total of 102 terms involved in transfer function generation. The full symbolic listing of the coefficients of the complex frequency variable S, therefore, is not advisable. On the other hand, if exact component values are available, the full symbolic listing may be used to advantage by calculating the numeric coefficients of S and thus check the earlier

result for the numeric transfer function.

Next, sampling of equation (5.9) gave the result of Fig. 5.22:

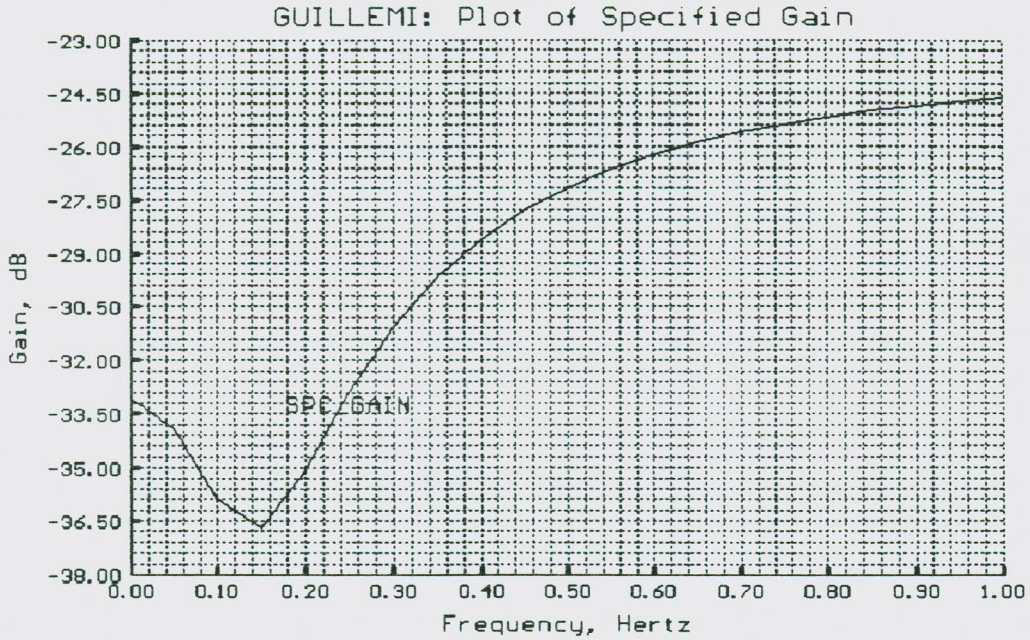


Fig. 5.22: Result of Sampling Equation (5.9).

The result of optimizing the parallel ladder network using the least 10-th Taylor method and a desired error level of 0.00000000E+0000 was summarized in the file GUILLEMI.OPT as follows:

****

Optimization results ...

I Name of input file : GUILLEMI
Number of nodes : 6
Number of branches : 10
Total R, L, C, Gm : 6,0,4,0

II NUMBER OF VARIABLES : 4
LEAST p LEVEL USED : 10
TOTAL SAMPLE POINTS : 21
DESIRED ERROR LEVEL : 0.00000000E+0000
ATTAINED ERROR LEVEL: 1.59806941E-0013
TOTAL ITERATIONS : 26

III Optimized component values:

R1 1.50000000000000E+0001
R2 1.84228242879356E+0000
R3 2.74049741077046E+0000
R4 7.52688601506700E+0000
R5 4.50000000000000E+0001
R6 6.76214724720658E+0001
C1 4.44000000000000E-0002
C2 2.12000000000000E-0002
C3 2.14000000000000E-0001
C4 1.48000000000000E-0002

****

Further results for Guillemin's parallel ladder passive network are shown in the plots of Figs. 5.23 - 5.29 below. Other outputs from the program include plots of the real and imaginary parts of the initial gain as well as the initial phase and listings of results that are available by selecting LOOK from the output menu.

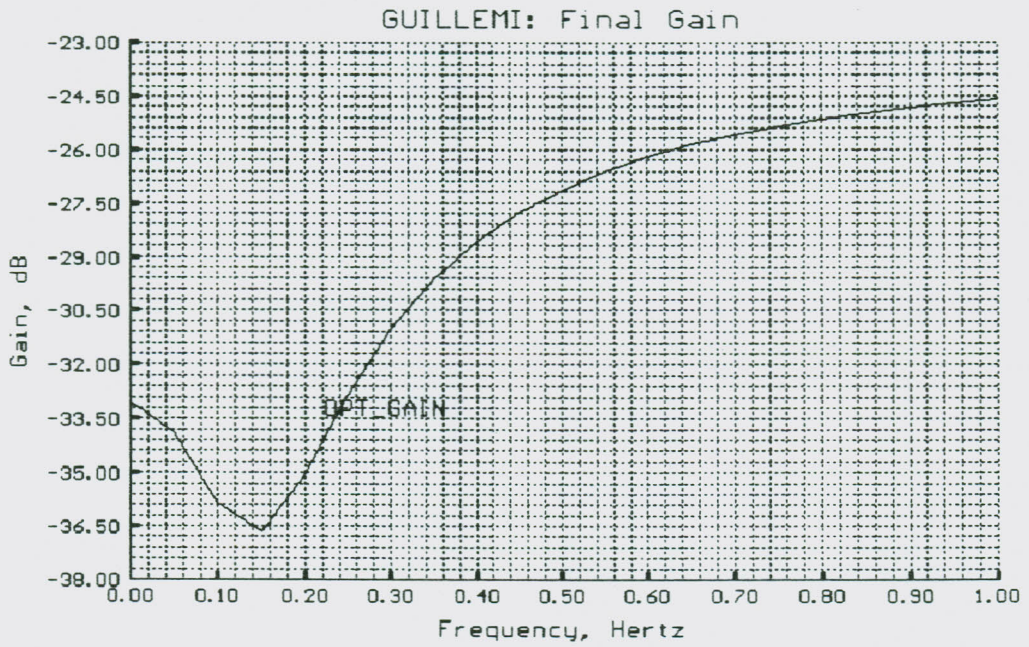


Fig. 5.23: The Gain Magnitude Response at the Final Iteration.

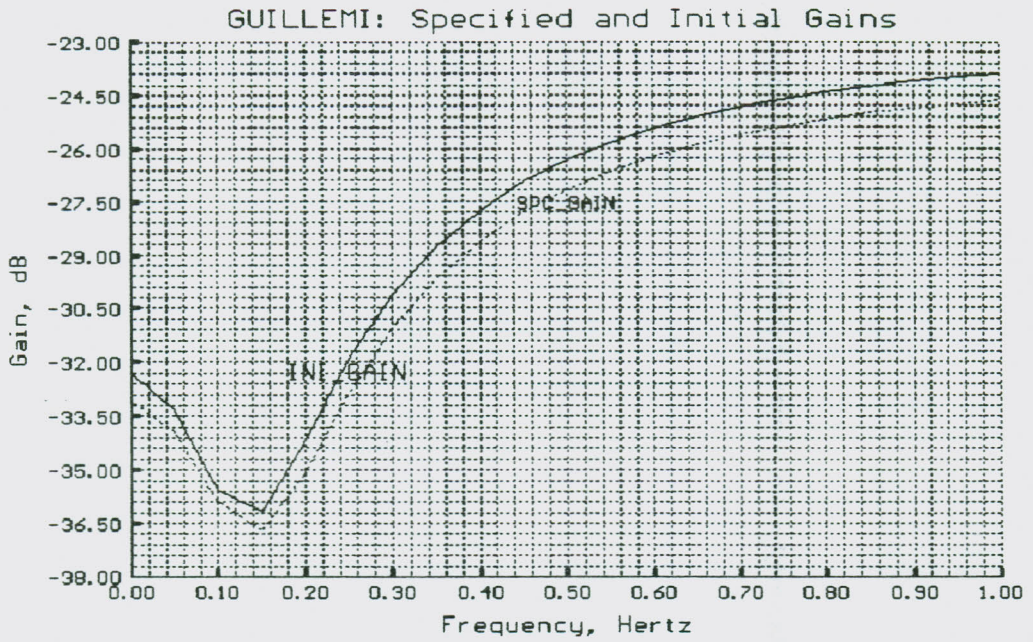


Fig. 5.24: Specified and Initial Gains Compared.

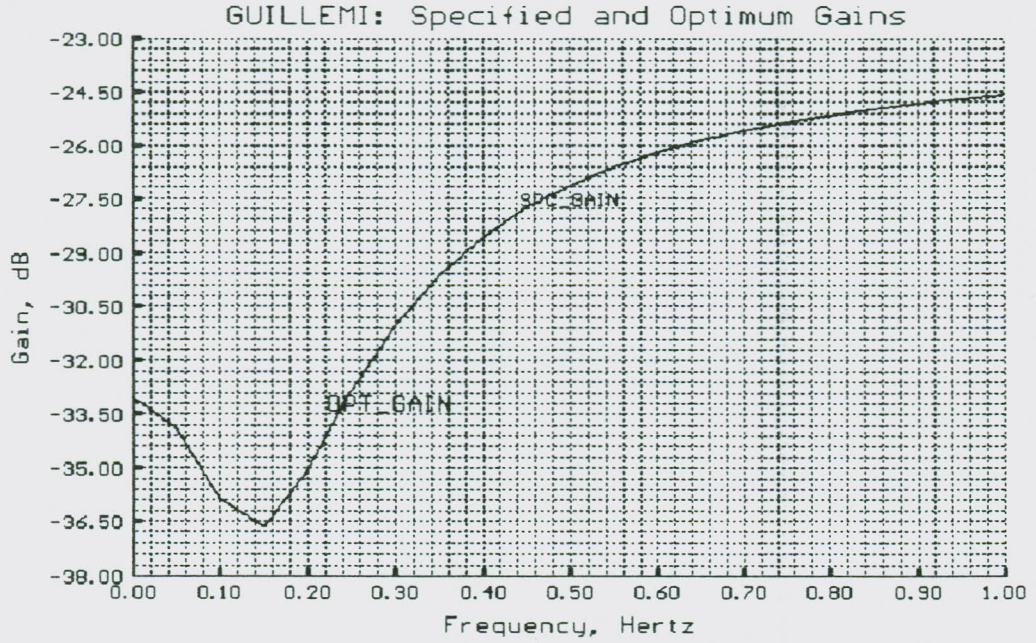


Fig. 5.25: Specified and Optimum Gains.

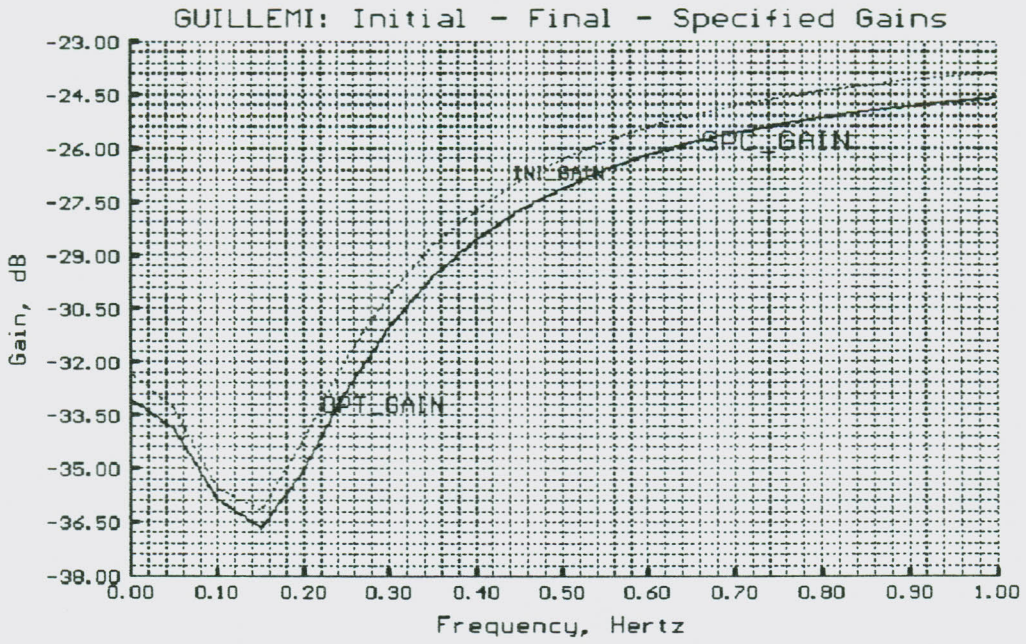


Fig. 5.26: Simultaneous Plots of Initial, Optimum, and Specified Gains.

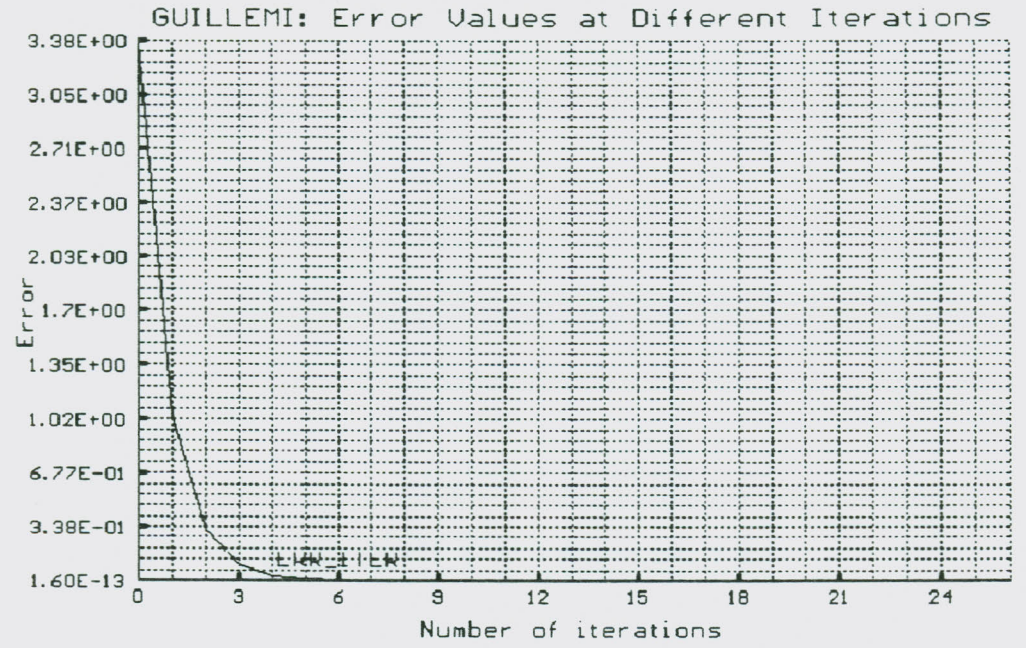


Fig. 5.27: Error Versus Iteration for the Optimum Gain.

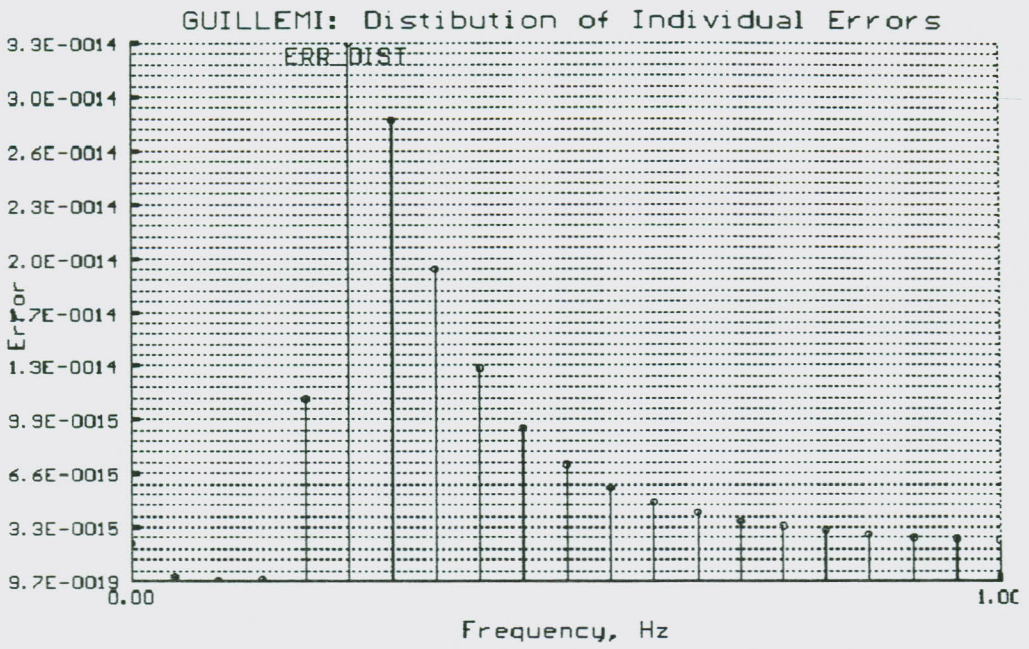


Fig. 5.28: The Error Distribution for the Optimum Gain.

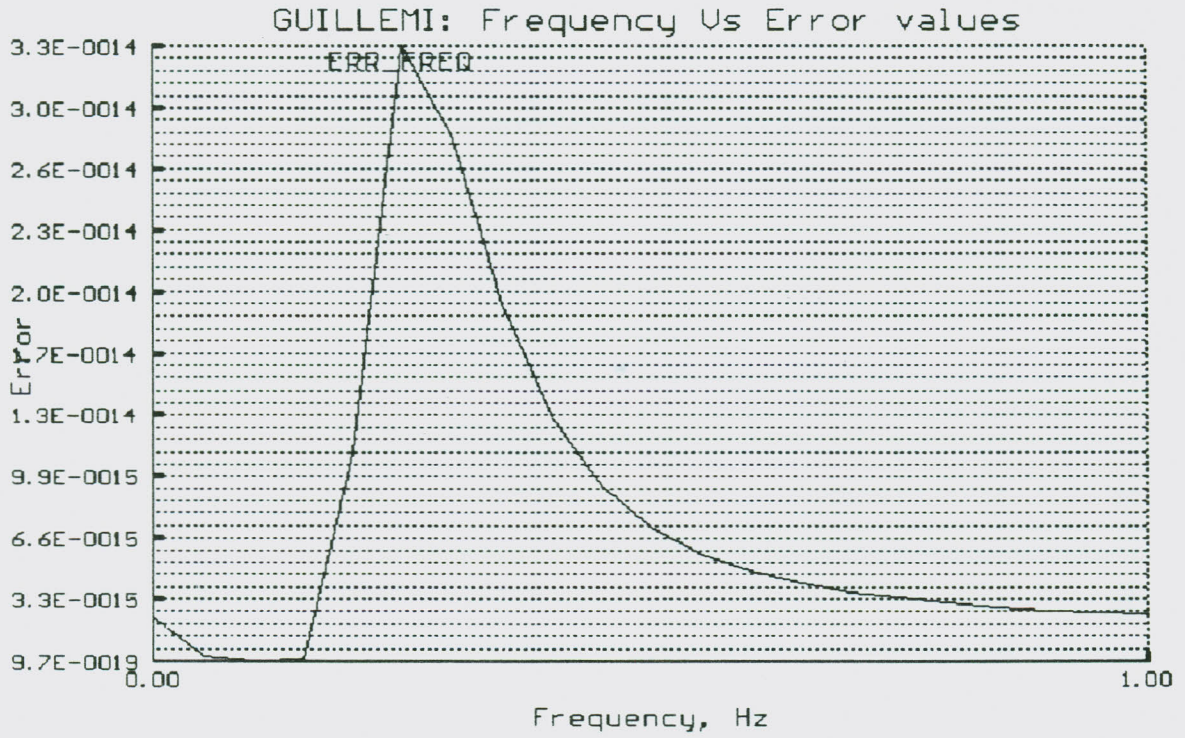


Fig. 5.29: Frequency Versus Error for the Final Gain.

As shown on pp. 95 - 97, there were similarities between the plots of Figs. 5.21, 5.22, and 5.23 for the initial, specified, and final gains. In the actual program, colors and line styles differentiated one plot from another when a single graph was used as in Figs. 5.24 - 5.26. Both of Figs. 5.24 and 5.25 on page 97 showed the specified gain in broken lines but, in the latter case, the specified gain fell on top of the optimum gain until there seemed no way of differentiating between the two. The same could be said about the plots in Fig. 5.26 on page 99. In any case, the situations were good indications of the accuracy achievable using *P-POND* -- a fact that may also be discerned by considering the very small error value of $\approx 1.6 \times 10^{-3}$ obtained for the last iteration.

On the other hand, although 26 iterations were required for convergence, the error has drastically decreased from a maximum value of 3.38 to about 0.3 in a matter of three iterations. The downhill property of the error has also been ascertained. These situations always hold when the initial approximation for the component vector is not displaced too much from the solution vector.

Further error analyses as in Fig. 5.28 showed that the maximum error for the final or optimum gain occurred at the sixth frequency sample point and the value achieved was 3.3×10^{-14} while the minimum error obtained was 3.7×10^{-19}. Hence, it may be said that the error was evenly distributed between the various samples. Infact, and this is shown in Fig. 5.29, the error values at the final iteration assumed a more or less Gaussian distribution over the entire range of sample points indicating once more the normal nature of the results obtained.

6. CONCLUSIONS

Design of electrical networks through computer-aided optimization techniques is called for when traditional methods of network design are inapplicable. The basic steps involved are obtaining an initial network, supplying initial values for the components, deriving a transfer function, establishing the specifications, and adjusting of the component values in an iterative manner until the specifications are satisfied. The Pascal Program for Optimal Network Design or *P-POND* designed and implemented in this study was based upon these general themes and had its specific bases on suitable blends of the nodal equation formulation method, graph theoretic concepts, the theory of sensitivity analysis, sampling strategies, the least p-th Taylor method, Fletcher's modification of the Levenberg-Marquardt algorithm, component damping techniques, and related programming concepts.

In carrying out the optimization, different initial component values yielded different optima, showing the local nature of the solution. It was therefore necessary to have a series of such runs before deciding on the global optimum.

Although primarily geared towards optimization, *P-POND* can also be used to analyze wide ranges of electrical networks when these are provided in their equivalent forms. The components accepted are resistors, inductors, capacitors, and voltage controlled current sources. Due provisions are necessary when networks lack such representations and these requirements could arise as a result of the use of the nodal method for formulating the network equations. Although the sparse-tableau, modified tableau, MNA (modified nodal approach), and RMNA (reduced modified nodal approach) possess sufficient generality to accommodate virtually all circuit configurations, the results are usually achieved at the costs of larger matrix sizes and computational efforts. In addition, much more elaborate considerations such as numerical conditioning of the system of equations are also required when employing these methods. The nodal method, on the other hand, presented well-conditioned system of equations that were solved with a minimal amount of computational effort. There were thus good reasons to retain the simplicity of this equation formulation method throughout the study.

In an attempt to use the schematic, the natural language of network design, the network was fed to the computer directly in its equivalent form at a stage called the *SCHEMATIC CAPTURE phase*. This was supported by the further options of keyboard and file inputs. Graphic input has some peculiar advantages over the other two ways specially when granted the options of modifying, saving, and retrieving the network schematic.

As the findings of the study revealed, there was much attraction in topological methods when used as vehicles for transfer function generation. The prime source of attraction lied in

their generality and to this could be added the attributes of generating multi-linear transfer function components, suitability for symbolic network analyses, and ease of modification.

The generation of transfer functions using topological methods was based upon finding of one-trees and two-trees of a linear graph and the resulting network functions could be put in symbolic forms. When coupled with the nodal equation formulation method, this approach, apart from avoiding matrix inversion, which is usually considered as an inefficient procedure, in the solution of the basic nodal system of equations, could lend an effective solution to the problem of computing symbolic network functions, i.e., functions of the network elements in symbolic rather than in numeric form. A symbolic network function was found quite useful in network optimization.

Also, and apart from theoretical significances, the topological approach was found to be a lot more handy for programming on the digital computer. The short-cut methods of writing, by inspection from the network, the determinants and cofactors of the node systems of equations, *without even writing out the equations*, were suited for implementation on a digital computer and, what is more, this could be done for both passive and active networks. The added advantage of avoiding the usual cancellations inherent in evaluations of network determinants also merits attention.

While appreciating the advantages of the topological approach, mention must also be made of the tree-finding problem as being the basic limitation of the approach. This limitation is primarily in the amount of computer time required to solve a particular design problem. While this may be accepted as a general truth, it must also be seen in lights of the results obtained in this study, the possibility of developing even more efficient algorithms for the task, the advent of high speed computing machines, and the general demand for accuracy.

On the other hand, the direct method of sensitivity analysis, which linked the network analysis and design stages, could easily be implemented into a computer program in this study as the use of topological methods reduced all partial differentiations to simple algebraic relations. The trend in this respect has almost often been to use the adjoint method of sensitivity analysis, although this requires consideration of two networks at any one time.

The least p-th Taylor method, which furnished the luxury of avoiding evaluation of the Hessian matrix, needed modifications before being used as a general optimizing algorithm. The classical Levenberg-Marquardt solution to the problem did not result in good convergence properties and so a resort to Fletcher's modification of the LM algorithm was only natural. The latter approach, although it performed good for many networks, demanded further refinements. The component damping techniques implemented in the study were attempts towards those ends

and gave good results. However, this does not mean that every thing done here was a success! Infact, for certain networks, the results did not converge to any sort of optimum. Oscillation about a value were the usual occurrences in such cases and there were also instances of divergence. These things all call for further research and study in the theory of numerical methods and optimization algorithms used in the study.

The current program may also be extended by allowing it for interfacing with a mouse. A more important issue, however, is the inclusion of a software library to model as many components and devices as possible. This will mark *P-POND* more versatile and relieve the user of working out the equivalent network himself. But then, and to say the least, the key question of memory management has to be thought of carefully and anew.

APPENDIX A: FLOW CHART FOR AN OPTIMIZATION PROGRAM.

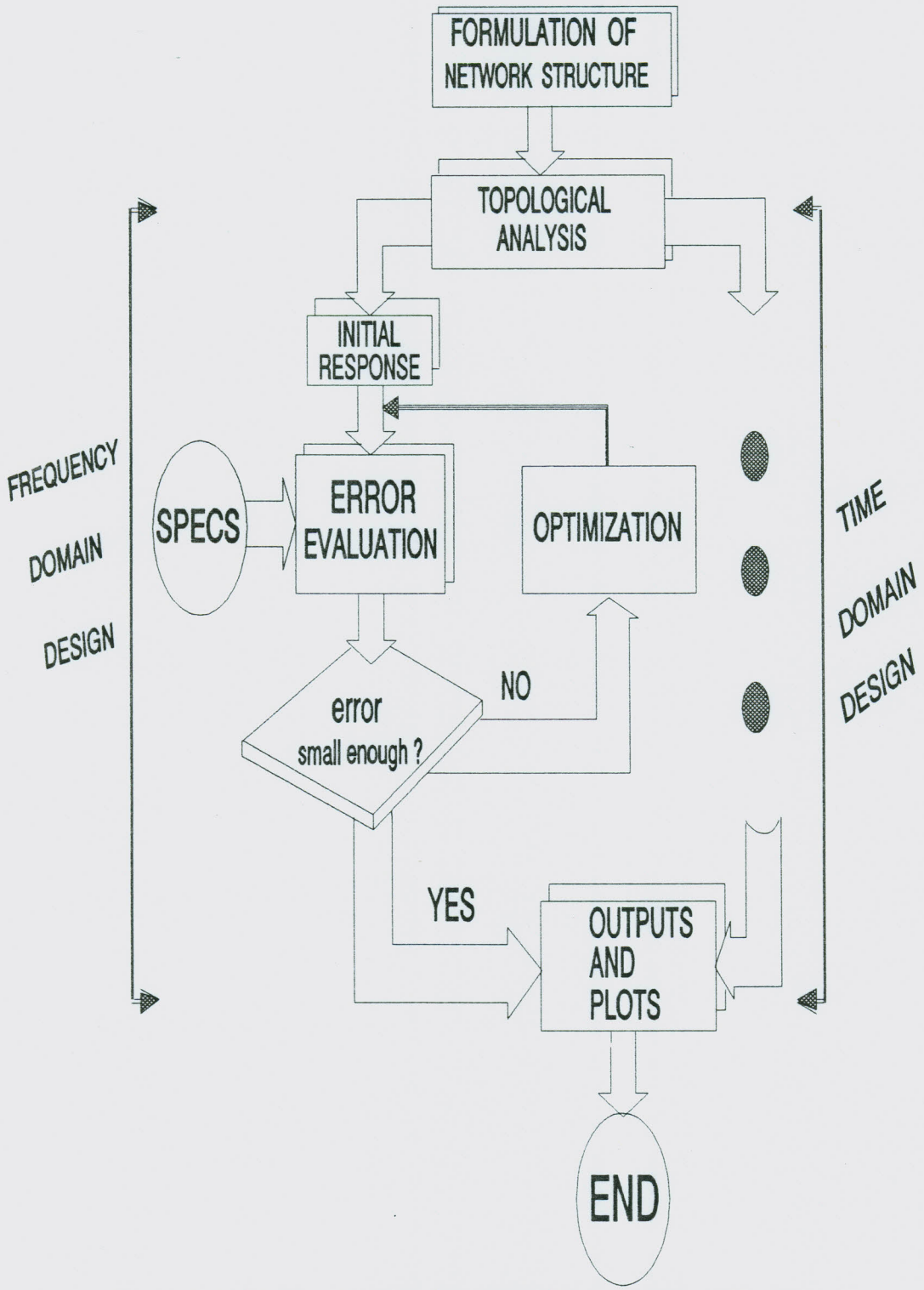


Fig. A1: COMPUTER-AIDED NETWORK OPTIMIZATION.

APPENDIX B: SOME THEOREMS ON NETWORK TREES.

Theorem B1:

Every finite connected graph contains a tree [11, p.25].

Theorem B2:

A finite graph is a tree if and only if there exists exactly one path between any two vertices of the graph [11, p.24].

Theorem B3:

The number of edges on a tree of a graph is one less than the number of nodes of the graph [20: p.72].

Theorem B4:

Any subgraph G of n-1 edges having no circuits must be a tree[8: p.18].

APPENDIX C: SOME PROPERTIES OF NETWORK TREES.

Property C1.

The determinant of any incidence matrix of a tree is equal to $\pm$ 1. [13: pp.125-126].

Property C2.

If the determinant of a major of A equals 0, then the edges do not form a tree. [17: p. 288].

Property C3.

A square submatrix of A of order n-1 is non-singular iff its columns correspond to the branches of a tree. [2: p.36].

Property C4.

The number of trees of a graph is equal to the determinant of AA^T. [20: p.75].

Property C5.

Given a graph, its incidence matrix is unique. The converse is not true, however.

APPENDIX D: PROGRAM FORMATTING

These suggestions will lead to more readable programs.

1. Each statement must begin on a separate line.

2. Each line shall be less than or equal to 72 characters.

3. At least one space must appear before and after ':=' and'='. At least one space must appear after ':'.

4. At least one blank line (or other recognizable dividing line) must appear before PROCEDURE and FUNCTION declarations.

5. PROGRAM, PROCEDURE, and FUNCTION headings must begin at the left margin. However, nested procedures and functions should be indented according to the level of nesting.

6. The keywords REPEAT, BEGIN, and END must stand on a line by themselves.

7. The main BEGIN-END block for programs, procedures and functions shall be lined up with the corresponding heading.

8. Each statement within a BEGIN-END, REPEAT-UNTIL or CASE statement must be aligned. The bodies of CONST, TYPE and VAR declarations and BEGIN-END, FOR, REPEAT, WHILE, CASE and WITH statements must be indented from the corresponding header keywords. Be consistent with indenting.

9. An IF-THEN statement must be displayed as:

```
IF expression
        THEN statement
        ELSE statement
```

Of course <statement> can be a compound statement:

```
IF expression
        THEN
                BEGIN
                        statements
                END
        ELSE
                BEGIN
                        statements
                END
```

An exception is allowed for multiple nesting (generalized case statement):

```
IF expression THEN
        statement
ELSE IF expression THEN
        statement
          .
          .
          .
ELSE
        statement
```

APPENDIX E: COMPLETE PROGRAM LISTINGS OF *P-POND*

```
{$M 1024,0,0}
program PROGRM_INTEGRATOR;

uses CRT, DOS;
var path: STRING;
var ch: CHAR;

begin
  clrscr;
  write('*************************');
  writeln('************************************');
  write('***                      ');
  writeln('                              ***');
  write('*** P-POND:- A PASCAL PROGRAM FOR ');
  writeln('OPTIMAL NETWORK DESIGN  ***');
  write('***          by Solomon Zewde, ');
  writeln('Graduate, EE Dept., AAU    ***');
  write('*************************');
  writeln('************************************');
  writeln;
  writeln;
  exec('\command.com', '/c c:');
  exec('\command.com', '/c cd\');
  exec('\command.com', '/c md ppond');
  exec('\command.com', '/c cd ppond');
  gotoxy(1, whereY - 1);
  writeln('Directory in use shall be c:\PPOND.
  Overwrite, Y/N?');
  repeat
    ch := UpCase(readkey)
  until ch in ['Y','N'];
  if ch = 'Y' then
  repeat
    write('Source containing ppond files? ');
    readln(path);
    if path <> '' then
    begin
      path := '/c copy ' + path + '\*.* c:\ppond';
      exec('\command.com', path)
    end
  until path <> '';
  clrscr;
  gotoxy(1,hi(windmax));
  write(' *** Please wait. ***');
  exec('\command.com', '/c attrib c:\tp5\*.* -r');
  exec('\command.com', '/c initial');
  exec('\command.com', '/c wiring');
  exec('\command.com', '/c analopt');
  exec('\command.com', '/c grapher');
  exec('\command.com', '/c final');
  exec('\command.com', '/c CD\USERS');
  clrscr
end.
```

```
PROGRAM INITIAL;
USES CRT, GRAPH, DOS;
Const
   const_1: STRING[24] = 'FOR ELECTRICAL NETWORKS';
   const0 : STRING[24] = 'by SOLOMON ZEWDE      ';
   const1 : STRING[15] = 'written for the';
   const2 : STRING[29] = 'DESIGN OF ELECTRICAL
   NETWORKS';
   const3 : STRING[13] = 'T H R O U G H';
   const4 : STRING[38] = 'COMPUTER-AIDED OPTIMIZATION
   TECHNIQUES';
   const5 : STRING[15] = 'AN M.Sc.THESIS';

var
   d,m,GrafMode,k: INTEGER;
            s1: STRING;
            ch: CHAR;

procedure wait;
begin
  write(#7);
  ch := readkey
end;

procedure erase;
begin
  for k := 1 to 17 do
  begin
    sound(k*100);
    delay(100)
  end;
  NoSound;
  delay(500)
end;

BEGIN
  DetectGraph(d,m);
  Initgraph(d,m,'c:\TP5');
  Cleardevice;
  GrafMode := GetGraphmode;

  for m := 1 to 2 do
  begin
    if m = 2 then
      cleardevice
    else
    begin
      SetColor(Yellow);
      setFILLstyle(solidfill, Blue);
      bar(90, 100, GetmaxX-150, GetmaxY);
      Rectangle(92, 102, GetmaxX-152, GetmaxY-2);
      Rectangle(90, 100, GetmaxX-150, GetmaxY);
      setColor(LightRed);
      setFILLstyle(linefill,Blue);
      bar(107,48,513,402);
      Rectangle(110, 50, 510, 400);
      Rectangle(107, 48, 513, 402);
      setFILLstyle(0, 0);
      SetTextStyle(1,0,3);
```

```
  SetColor(LightRed);
  for k := 1 to 3 do
  begin
    OutTEXTxy(259+k, 200, 'P');
    OutTEXTxy(259+k, 240, 'P');
    OutTEXTxy(259+k, 280, 'O');
    OutTEXTxy(259+k, 320, 'N');
    OutTEXTxy(259+k, 360, 'D')
  end;

  setColor(LightGray);
  OutTEXTxy(230, 200, 'A');
  OutTEXTxy(276, 200, 'ascal');
  OutTEXTxy(276, 240, 'rogram for');
  OutTEXTxy(276, 280, 'ptimal');
  OutTEXTxy(276, 320, 'etwork');
  OutTEXTxy(276, 360, 'esign')
end;

SetTexTStyle(3, 0, 10);
if m = 2 then
begin
  SetColor(LightRed);
  for m := 1 to 2 do
  begin
    if m = 1 then
      k := 0
    else
      k := 263;
    circle(117+k,300,2);
    line(120+k,300,125+k,300);
    rectangle(125+k,298,145+k,302);
    line(145+k, 300, 155+k, 300);
    line(155+k, 300, 155+k, 340);
    line(150+k, 340, 160+k, 340);
    arc(155+k, 350, 0, 180, 5);
    line(155+k, 345, 155+k, 380);
    line(120+k, 380, 237+k, 380);
    circle(117+k, 380, 2);
    circle(239+k, 380, 2);
    line(185+k, 380, 185+k, 363);
    circle(185+k, 350, 12);
    line(185+k,356,185+k,344);
    line(183+k,346,185+k,344);
    line(187+k,346,185+k,344);
    line(185+k, 338, 185+k, 300);
    line(185+k, 300, 195+k, 300);
     arc(200+k, 300, 0, 180, 5);
     arc(210+k, 300, 0, 180, 5);
     arc(220+k, 300, 0, 180, 5);
    line(225+k, 300, 237+k, 300);
    circle(239+k, 300, 2)
  end
end;
for k := 40 to 60 do
begin
  if GrafMode = VGAHi then
    SetColor(Red);
  OutTextXY(145, k+40, 'P');
  OutTextXY(210, k+9, 'P');
  OutTextXY(275, k-5, 'O');
  OutTextXY(350, k+11, 'N');
  OutTextXY(420, k+40, 'D')
end;

for k := 1 to 7 do
begin
  if k <= 3 then
    line(399+k,158,399+k,194+k)
  else if k < 7 then
    line(399+k,158,399+k,193+k)
  else
    line(399+k,158,399+k,192+k);
end;

k := 40;
if GrafMode = VGAHi then
  SetColor(LightRed);
OutTextXY(145,k+40,'P');
OutTextXY(210,k+9,'P');
OutTextXY(275,k-5,'O');
OutTextXY(350,k+11,'N');
OutTextXY(420,k+40,'D');

setTEXTstyle(1,0,2);
SetColor(LightGray);
if m = 2 then
begin
  const0 := const_1;
  SetColor(LightCyan)
end;
for k := 1 to length(const0) do
begin
  s1 := Copy(const0,k,1);
  if m = 1 then
    OutTextXY((GetmaxX div 2 - 150) + k*16,
    GetmaxY-50, s1)
  else
    OutTextXY((GetmaxX div 2 - 200) + k*16,
    GetmaxY-50, s1);
  delay(30)
end;
SetTextStyle(1,0,3);
wait;
cleardevice;
end;

SetColor(LightRed);
SetFillStyle(solidfill,blue);
bar(0, 0, GetmaxX, GetmaxY);
setFILLstyle(solidfill, black);
bar(70, 122, GetmaxX-30, GetmaxY-148);

SetColor(LightRed);
setFILLstyle(Solidfill, Magenta);
bar(60, 140, GetmaxX-52, GetmaxY-140);
rectangle(60,140,GetmaxX-52,GetmaxY-140);
rectangle(58,138,GetmaxX-50,GetmaxY-138);
if GrafMode = VGAHi then
  SetCOLOR(yellow);
outTEXTxy(70,150, const1);
if GrafMode = VGAHi then
  SetColor(LightCyan);
for k := 1 to 30 do
```

```
begin
  s1 := Copy(const2,31-k,1);
  OutTextXY(70 + 16*(30-k), 195, s1);
  delay(50)
end;
OutTextXY(70, 225, const3);
OutTextXY(70, 255, const4);

SetTextStyle(3,0,2);
if GrafMode = VGAHi then
  SetColor(yellow);
for k := 1 to length(const5) do
begin
  s1 := Copy(const5,k,1);
  if (k = length(const5)) and (const5[k] = 'S') then
    OutTextXY(47 + k*12, GetmaxY-170, s1)
  else if const5[k] = ' ' then
    OutTextXY(55 + k*8, GetmaxY-170, s1)
  else
    OutTextXY(55 + k*12, GetmaxY-170, s1);
  delay(20)
end;
delay(500);
wait;
erase;

cleardevice;
setColor(Yellow);
setFILLstyle(linefill, Blue);
bar(98, 128, 560, 360);
rectangle(100,130,GetmaxX-80,GetmaxY-120);
rectangle(98,128,GetmaxX-78,GetmaxY-118);
s1 := 'Advisor';
outTEXTxy(GetmaxX div 2 - (length(s1)*8) div 2 - 10,
200, s1);
s1 := 'Dr. Wolde Ghiorghis W.';
outTEXTxy(GetmaxX div 2 - (length(s1)*8) div 2, 240,
s1);
ch := readkey;
CloseGraph
END.
```

```
PROGRAM WIRING;
(*-------------------------------------------------------------

  This program performs schematic capture of an electrical network.
  Once the network schematic is captured, it is also coded here for
  analysis and optimization purposes ...

-------------------------------------------------------------*)

(*********************************)
USES CRT, GRAPH, ROOT, COMPNNTS;
(*********************************)

(*-------------------------------------------------------------*)
Const
  const0 = 'PPOND: A PASCAL PROGRAM FOR OPTIMAL
  NETWORK DESIGN.';
Type
  name = Procedure(scll,dirr: INTEGER); { define procedural type}
Var
  InFileName: STRING[12];
     myproc: name;
      zis: TEXT;
(*-------------------------------------------------------------*)

Procedure NetworkName;
var ii,jj: INTEGER;

  procedure GetFile;
  begin
    repeat
      InFileName := '';
      s:= 'File name of NETWORK INFORMATION = ';
      OutTEXTxy(i*8-120,j*(16+jj),s);
      repeat
        ch := readkey;
        if ch < > #13 then
        begin
          InFileName := InFileName + ch;
            OutTEXTxy(i*8+150+8*length(InFileName),
            j*(16+jj),ch)
        end
      until ch = #13;

      if InFileName = '' then
      begin
        write(^G);
        write(^G)
      end
    until InFileName < > '';
    jj := jj + 2
  end; { of GetFile }

begin { NetworkName }
  setTEXTstyle(2,0,7);
  bar(GetmaxX div 6, GetmaxY div 3, GetmaxX-100, GetmaxY div 2);
  SetColor(Yellow);
  rectangle(GetmaxX div 6+8, GetmaxY div 3+8,
        GetmaxX - 108, GetmaxY div 2-8);
  rectangle(GetmaxX div 6 + 11,GetmaxY div 3 + 10,
        GetmaxX - 111, GetmaxY div 2 - 10);
  SetColor(LightCyan);
```

```
  s := 'Enter file name for network ... ';
  OutTEXTxy(GetmaxX div 6 + 40, GetmaxY div 3 + 30, s);

  setTEXTstyle(0,0,0);
  GotoXY((GetmaxX div 6+180)div 8,(GetmaxY div 3+51) div 13);

  jj := 0;
  i := whereX;
  j := whereY;
  GetFile;

  InFileName := InFileName + '.LOD';
  repeat
    Assign(zis, InFileName);
    {$I-}
      Reset(zis);
      ii := IOresult
    {$I+};
    if ii = 0 then
    begin
      ii := 0;
      repeat
        s := #7'File already exists. Replace, Y/N?';
        OutTEXTxy(i*8-120,j*(17+ii),s);
        ch := UpCase(ReadKey);
        OutTEXTxy(i*8+170, j*(17+ii), ch);
        if ch = 'N' then
          GetFile;
        if not (ch in ['Y','N',#13]) then
          write(^G);
        ii := ii + 2
      until ch in ['Y','N'];
    end
  until (ch = 'Y') or (ii < > 0)
end; { of NetworkName }

procedure xReset;
begin
  ClrBrd;
  NetworkName;
  ClrBrd
end;

procedure eraseMODE(xx,yy: INTEGER);
begin { eraseMODE }
  if Eraze = 0 then
  begin
    MenuBar;
    outTEXTxy(8,14,'Select erase mode:');
    outTEXTxy(168,14,'1=Normal
    2=Extended.');
    EscBar('              ');
    ClrPart
  end;
  repeat
    if Eraze = 0 then
      chE := readkey;
    if chE = #0 then
      chE := readkey;
    case chE of
      '1': GetBlank(xx, yy);
      '2': GetBlankE(xx, yy)
    end
  until chE in ['1', '2'];
  Eraze := 1
end; { of eraseMODE }

procedure SELECTwires;

  procedure cmn(xx, yy, direction: INTEGER);
  { var done: INTEGER; has been made global! }
  begin { cmn }
    case direction of
      1: if xx < = GetmaxX-10 then { Right }
           done := 1;
      2: if xx > = 10 then { Left }
           done := 2;
      3: if yy < = GetmaxY-15 then { Down }
           done := 3;
      4: if yy > = 25 then { Up }
           done := 4;
      5: if (xx < = GetmaxX-10) and (yy < = GetmaxY-15) then
           done := 5; { DiaLD, F5 }
      6: if (xx > = 10) and (yy < = GetmaxY-15) then
           done := 6; { DiaRD, F6 }
      7: if (xx < = GetmaxX-15) and (yy > = 25) then
           done := 7; { DiaLU, F7 }
      8: if (xx > = 10) and (yy > = 25) then
           done := 8; { DiaRU, F8 }
      9: if xx < = GetmaxX-10 then
           done := 9; { F3, Horizontal Jumper }
     10: if yy < = GetmaxY-15 then
           done := 10; { F4, Vertical Jumper }
    end;
    if erasing then
    begin
      case chE of
        '1': begin
               for i := 1 to Grid do
               case done of
                 1: GetBlank(x0+i,y0);
                 2: GetBlank(x0-i,y0);
                 3: GetBlank(x0,y0+i);
                 4: GetBlank(x0,y0-i);
                 5: GetBlank(x0+i,y0+i);
                 6: GetBlank(x0-i,y0+i);
                 7: GetBlank(x0+i,y0-i);
                 8: GetBlank(x0-i,y0-i)
               end
             end;
        '2': GetBlankE(xx,yy)
      end
    end;
    x1 := xx;
    y1 := yy;
    if drawing then
      if (direction < = 8) and (done = direction) then
        Lineto(x1,y1);
    if (direction = 9) and (done = direction) then
    begin
      GetBar(black,x1-15,y1,x1-11,y1,black);
      SetColor(white);
      arc(x1-10, y1, 0, 180, 5);
      x1 := x1 - 5
```

```
    end;
    if (direction = 10) and (done = direction) then
    begin
      GetBar(black,x1,y1-15,x1,y1-11,black);
      SetColor(white);
      arc(x1, y1-10, 270, 90, 5);
      y1 := y1 - 5
    end;
    moveto(x1,y1)
  end; { of cmn }
begin { SELECTwires }
  if badCOMMAND then
  begin
    com := '                ';
    Command;
    badCOMMAND := false;
  end;
  chh := Upcase(readkey);
  case chh of
    #77: cmn(x0+Grid, y0, 1); { Right}
    #75: cmn(x0-Grid, y0, 2); { Left }
    #80: cmn(x0, y0+Grid, 3); { Down }
    #72: cmn(x0, y0-Grid, 4); { Up   }
    #63: cmn(x0+Grid, y0+Grid, 5); {DiaLD, F5}
    #64: cmn(x0-Grid, y0+Grid, 6); {DiaRD, F6}
    #65: cmn(x0+Grid, y0-Grid, 7); {DiaLU, F7}
    #66: cmn(x0-Grid, y0-Grid, 8)  {DiaRU, F8}
  end;
  if drawing then
  case chh of
    #61: cmn(x0+Grid,y0,9);  {F3, Horizontal jumper wire}
    #62: cmn(x0,y0+Grid,10)  {F4, Vertical jumper wire}
  end;
  if jumping then
  case chh of
    #71: begin { Home = 10,25 }
           x0 := 10;
           y0 := 25
         end;
    #73: begin { PgUp = x1,25 }
           x0 := x1;
           y0 := 25
         end;
    #79: begin { End = maxX-20, maxY-15 }
           x0 := GetmaxX-20;
           y0 := GetmaxY-15
         end;
    #81: begin { PgDn = x1,maxY-15 }
           x0 := x1;
           y0 := GetmaxY-15
         end
  end;
  if jumping and (chh in [#71,#73,#79,#81]) then
  begin
    case chh of
      #71: Com := '   Jump Home';
      #73: Com := '   Jump Top';
      #79: Com := '   Jump End';
      #81: Com := '   Jump Bottom'
    end;
    Command;
    moveto(x0,y0);
    x1 := x0;
    y1 := y0
  end;
  if (ch=#0) and (chh in [#28..#62,'C'..'G','I','J',
    'L','N','O','Q','R'..'Z']) then
      badCOMMAND := true;
  if jumping and (chh in [#71,#73,#79,#81]) then
    badCOMMAND := false;
  if drawing and (chh in[#61,#62]) then
    badCOMMAND := false;
  ch := chh
end; { of SELECTwires }

procedure SELECTprocedure;
begin { SELECTprocedure }
  case ch of
    'R': myproc := RR;
    'I': myproc := LL;
    'C': myproc := CC;
    'V': myproc := VCCS;
    'N': NODE;
    'E': EARTH
  end
end; { of SELECTprocedure }

procedure SELECTcomponents;
var right, left, up, down: BOOLEAN;

  procedure PASSprocedure(element: STRING);
  begin { PASSprocedure }
    SELECTprocedure;
    clrPART;
    s := '... arrow keys to position the ' + 'element';
    outTEXTxy(8,14,s);
    chh := ch;
    ch := readkey;
    case ch of
      #0:begin
           ch := readkey;
           if ch in[#77,#75,#80,#72] then
             COMs;
           case ch of
             #77: begin
                    myproc(Scl,1);  {Right}
                    right := true
                  end;
             #75: begin
                    myproc(Scl,10); {left }
                    left := true
                  end;
             #80: begin
                    myproc(Scl,110);{down }
                    down := true
                  end;
             #72: begin
                    myproc(Scl,11); {up   }
                    up := true
                  end
           end;
           if ch in[#77,#75,#80,#72] then
             ch2 := #2; { #2 = ^B }
         end
```

```
      end;
      ch := chh
  end; { of PASSprocedure }

begin { SELECTcomponents }
  right := false;
  left := false;
  up := false;
  down := false;
  if not (ch in ['E','N'] ) then
    MenuBar;
  repeat
    if ch in['R','I','C','V'] then
      chNode := '-';
    case ch of
      'R': PASSprocedure('resistor');
      'I': PASSprocedure('inductor');
      'C': PASSprocedure('capacitor');
      'V': PASSprocedure('VCCS');
      'N': NODE;
      'E': EARTH
    end;
    if (ch in ['R','I','C','V']) and (done = 9) then
    begin
      if right then
      begin
        GetBar(black,x1-5,y1,x1,y1,black);
        x1 := x1 - 5
      end
    end
    else if (ch in ['R','I','C','V']) and
       (done = 10) then
    begin
      if down then
      begin
        GetBar(black,x1,y1-5,x1,y1,black);
        y1 := y1 - 5
      end
    end;
    right := false;
    left := false;
    up := false;
    down := false;
    done := 0;
    SetColor(white);
    moveto(x1,y1)
  until ch2=#2;
  if not (ch in['E','N']) then
    drawMENU
end; { of SELECTcomponents }

Procedure GetNetwork;
begin { GetNetwork }
  Getpen(x0,y0);
  if erasing then
    eraseMODE(x0,y0);
  MENUs;
  if erasing and (chE='2') then
  begin
    MenuBar;
    OutTEXTxy(8,14,'Use F5 - F8 or the arrow keys ...');
    EscBar('Esc ends ERASE')
  end;
  repeat
    ch := Upcase(readkey);
    chh := #1;
    COMs;
    GetPen(x0,y0);
    if erasing then
      eraseMODE(x0,y0);
    if not drawing then
    case ch of
      #4 : xReset;
      #45 : ClrMenu; {#45 = - character}
      #0 : SELECTwires;
      else
        if ch<>#27 then
          Warnning
    end;

    if drawing then
    case ch of
      #4 : xReset;
      #45 : ClrMenu; { #45 = - character }
      #0 : SELECTwires;
      'R', 'I', 'C', 'V', 'N', 'E'
        : SELECTcomponents;
      else
        if ch <> #27 then
          Warnning
    end;

    ch2 := '';
    if ch=#45 then { #45 = - character }
    repeat
      ch2 := readkey;
      if ch=#45 then
        ch := #27;
      if ch2=#43 then
        defaults;
    until ch2 = #43; { #43 = + character }

    if badCOMMAND and (ch<>#27) then
      Warnning
    else
      if erasing or not (ch in[#71,#73,#79,#81]) then
        if not (ch in['R','I','C','V']) then
          COMs;
      if ch<>#27 then
        GetPen(x1,y1);
      if erasing then
        eraseMODE(x1,y1);
      x0 := x1;
      y0 := y1
  until ch=#27;
  rootMENU;
  drawing := false;
  jumping := false;
  erasing := false
end; { of GetNetwork }

procedure optionsWARNNING;
begin { optionsWARNNING }
  if ch<>#27 then
```

```
  begin
    if ch=#0 then
      ch := readkey;
    ch := #1;
    Warnning
  end
end; { of optionsWARNNING }

procedure change(msg1,msg2: STRING;
                 var SclGrid: INTEGER; xx: INTEGER);
var SG: STRING;
begin { change }
  MenuBar;
  s := '... Enter ' + msg1 + ' then SPACE bar';
  OutTextXY(8,14,s);
  s := 'Esc ends ' + msg1;
  EscBar(s);
  AcceptSG(SclGrid);
  ch := #27;
  setcolor(14);
  str(SclGrid,SG);
  OutTEXTxy(xx,GetmaxY-9,msg2+SG);
  setcolor(15);
  Com := Com + 'NEW';
  Command
end; { of change }

procedure changeScale;
begin { changeScale }
  GetBar(1,GetmaxX-236,GetmaxY-11,GetmaxX-147,GetmaxY,14);
  change('SCALE',' Scale=',Scl,GetmaxX-236);
end; { of changeScale }

procedure changeGrid;
begin { changeGrid }
  GetBar(1,GetmaxX-146,GetmaxY-11,GetmaxX-76,GetmaxY,14);
  change('GRID','Grid=',Grid,GetmaxX-146);
end; { of changeGrid }

procedure changeBell;
begin { changeBell }
  MenuBar;
  s := 'Let warnning sound on, Y/N?';
  OutTEXTxy(8,14,s);
  repeat
    ch := UpCase(readkey);
    case ch of
      'Y': bell := true;
      'N': bell := false;
      else
        optionsWARNNING
    end
  until ch in['Y','N',#27];
  MenuBar;
  OutTEXTxy(8,14,' ... Esc to continue');
  GetBar(1,GetmaxX-66,GetmaxY-11,GetmaxX,GetmaxY,14);
  if not bell then
    OutTEXTxy(GetmaxX-65,GetmaxY-9,'NO-SOUND')
  else
    OutTEXTxy(GetmaxX-50,GetmaxY-9,'SOUND')
end; { of changeBell }

procedure SELECToptions;
begin { SELECToptions }
  optionsMENU;
  repeat
    ch := UpCase(readkey);
    COMs;
    case ch of
      'U': changeBell;
      'G': changeGrid;
      'S': changeScale;
      else
        optionsWARNNING
    end
  until ch=#27;
  options := false;
  rootMENU
end; { of SELECToptions }

procedure SELECTmode(comSTR: STRING; var mode: boolean);
begin { SELECTmode }
  Com := comSTR;
  GetPen(x0,y0);
  GetPen(x0,y0);
  mode := true
end; { of SELECTmode }

procedure STOREinTEMPO;
  procedure storeRLC(xx: INTEGER);
  begin { storeRLC }
    with UGX^ do
    begin
      for i := 1 to nT do
      if U[i,0] = xx then
      begin
        for j := 0 to 3 do
        write(zis,U[i,j],' ');
        writeln(zis,X[i])
      end
    end
  end; { of storeRLC }

  procedure storeALL;
  begin { storeALL }
    with UGX^ do
    begin
      writeln(zis,nN,' ',nR,' ',nL,' ',nC,' ',nGm);
      storeRLC(1);
      storeRLC(2);
      storeRLC(3);
      x0 := 0;
      for i := 1 to nT do
      begin
        if U[i,0] < 0 then
        begin
          for j := 0 to 3 do
          write(zis,U[i,j],' ');
          x0 := x0 + 1;
          writeln(zis, X[i],' ',G[x0,2],' ',G[x0,3])
        end
      end
    end
  end; { of storeALL }
```

```
begin { STOREinTEMPO }
  write(' Extracting network information ...');
  assign(zis,'Tempo');
  rewrite(zis);
  storeALL;
  close(zis)
end; { of STOREinTEMPO }

procedure SAVEinInFileName;
var xx: INTEGER;
begin { SAVEinInFileName }
  with UGX^ do
  begin
    assign(zis,'tempo');
    reset(zis);
    readln(zis,nN,nR,nL,nC,nGm);

    for i := 1 to nT do
    begin
      for j := 0 to 3 do
      read(zis,U[i,j]);
      read(zis,X[i]);
      if i > nR + nL + nC then
        read(zis,G[i-nR-nL-nC,2],G[i-nR-nL-nc,3]);
      readln(zis)
    end;
    close(zis);

    Assign(zis,InFileName);
    Rewrite(zis);

    writeln(zis,nN,' ',nR,' ',nL,' ',nC,' ',nGm);
    for i := 1 to nT do
    begin
      if U[i,0] < 0 then
      begin
        xx := U[i,2];
        U[i,2] := U[i,3];
        U[i,3] := xx
      end;
      for j := 0 to 3 do
      write(zis,U[i,j]:5,' ');
      write(zis,' ',X[i]:14);
      if i > nR + nL + nC then
        write(zis,G[i-nR-nL-nC,2]:5,' ',G[i-nR-nL-nC,3]:5);
      writeln(zis)
    end;
    Close(zis)
  end
end; { of SAVEinInFileName }

procedure beginJOB;
begin { beginJOB }
  ClrScr;
  GetGraph;
  Pen;
  MenuBar;

  setfillstyle(1,Black);
  bar(0,0,GetmaxX,GetmaxY);

  setCOLOR(LightRed);

for i := 1 to length(const0) do
begin
  if i < length(const0) div 3 then
    sound(i*50);
  s := Copy(const0,length(const0) - i + 1,1);
  OutTextXY(100 + 8*(length(const0)-i), GetmaxY div 45, s);
  if s < > ' ' then
    delay(10);
  Nosound
end;

setTEXTstyle(1,0,2);
s := 'Welcom e to the SCHEMATIC CAPTURE phase';
for i := 1 to length(s) do
begin
  if (i > = 17) and (i < = 33) then
    setColor(Green)
  else
    setColor(Yellow);
  OutTEXTxy(GetmaxX div 10 - 10 + (i*13), GetmaxY div 6,s[i]);
end;

setfillstyle(1,Brown);
bar(GetmaxX div 6+2, GetmaxY div 3+2,
   GetmaxX - 102, GetmaxY div 2-2);
SetColor(Yellow);
rectangle(GetmaxX div 6+8, GetmaxY div 3+8, GetmaxX - 108,
  GetmaxY div 2-8);
rectangle(GetmaxX div 6 + 11, GetmaxY div 3 + 10, GetmaxX -
  111, GetmaxY div 2 - 10);
s := '      Bypass, Y/N ?';

setcolor(LightCyan);
setTEXTstyle(2,0,7);
OutTEXTxy(GetmaxX div 6 + 40, GetmaxY div 3 + 30, s);
write(^G);

repeat
  bar(GetmaxX div 6 + 290, GetmaxY div 3 + 15, GetmaxX div 6
    + 320, GetmaxY div 3 + 60 );
  ch := UpCase(readkey);
  if ch = #0 then
  begin
    ch := readkey;
    ch := '$'
  end;
  if not (ch in ['Y', 'N']) then
  begin
    OutTEXTxy(GetmaxX div 6 + 300, GetmaxY div 3 + 30, ch);
    write(getX div 8, GetY div 13, ^G);
    write(getX div 8, GetY div 13, ^G)
  end
until ch in ['Y', 'N'];

if ch = 'Y' then
  HALT;

NetworkName;
help;
setting;
Defaults;
rootMENU;
```

```
  for i := 0 to nodes do
  for j := 1 to 2 do
  where[i,j] := 0
end; { of beginJOB }

procedure codeNETWORK;
begin { codeNETWORK }
  repeat
    ch := UpCase(readkey);
    if ch=#0 then
    begin
      ch := readkey;
      ch := #1;
    end;
    if ch< >#27 then
      GetPen(x0,y0);
    case ch of
       'H': GetHelp;
       'D': SELECTmode('  DRAW',drawing);
       'J': SELECTmode('  JUMP',jumping);
       'E': SELECTmode('  ERASE',erasing);
       'O': SELECTmode('  OPTIONS',options);
       #4 : xReset;
      #45 : ClrMenu;
       else
         if ch< >'X' then
           Warnning
    end;

    ch2 := '';
    if ch=#45 then { #45 = - character }
    repeat
      ch2 := readkey;
      if ch2=#43 then { #43 = + character }
      begin
        defaults;
        rootMENU;
        ch := #27
      end
    until ch2=#43;

    if ch in ['D','E','J'] then
      GetNetwork;
    if ch in ['O'] then
      SELECToptions;
    if ch < > #27 then
      GetPen(x1,y1);
    x0 := x1;
    y0 := y1
  until ch = 'X';
  closeGraph;
  clrscr;
  if sameNode then
       nN := nN-1
end; { of codeNETWORK }

procedure eraseTEMPO;
begin { eraseTEMPO }
  Assign(zis,'Tempo');
  Erase(zis)
end; { of eraseTEMPO }
```

```
begin { PROGRAM WIRING }

  clrscr;
  Mark(heap);
  New(UGX);
  beginJOB;
  codeNETWORK;
  STOREinTEMPO;
  SAVEinInFileNAME;
  eraseTEMPO;
  Release(heap)

end. { OF MAIN PROGRAM WIRING }


{File name: ROOT.PAS.}

UNIT ROOT;

                              interface
(*******************_____________________***************)

(**************)
USES GRAPH, CRT;
(**************)

Const
  Polygon  : Array[1..5] of pointTYPE
           =
((x:2;y:6),(x:8;y:10),(x:8;y:3),(x:14;y:18),(x:8;y:20));
  branches = 200;
    nodes = 100;

Type
  dataOUT = ^Arrays;
  Arrays  = record
              U: Array [0..40,0..5] of -3..branches;
              G: Array [0..10,2..3] of  1..branches;
              X: Array [0..40     ] of REAL;
            end;

var
  where: array [ 0..nodes, 1..2 ] of INTEGER;
  chE  : '1'..'2';
  UGX  :  dataOUT;
  dir  :    BYTE;
  Eraze:    0..1;
  drawing,
  jumping,
  erasing,
  options,
  bell,
  badCOMMAND,
  sameNode,
  earthh:  BOOLEAN;
  s,com:  STRING;
  ch2  :STRING[1];
  ch,chh,chNODE: CHAR;
  x0,y0,x1,y1,Scl,Grid,i,j: INTEGER;
  SyzP,Penna,blnk,blnkE,hlp,heap: POINTER;
  nR,nL,nC,nGm,nN,nP,vR,vL,vC,vGm,nT,done:INTEGER;
```

```
procedure GetGraph;

procedure GetBar(fillclr,x,y,xx,yy,drwclr: INTEGER);

procedure setting;

procedure Posn;

procedure Diagonals;

procedure Command;

procedure warnning;

procedure EscBar(message: STRING);

procedure MenuBar;

procedure ClrBrd;

Procedure ClrMenu;

procedure clrPART;

procedure restorePART;

procedure Save;

procedure Rtrv;

procedure Pen;

procedure Help;

procedure GetHelp;

procedure GetPen( x,y: INTEGER ) ;

procedure GetBlank(x,y: INTEGER);

procedure GetBlankE(x,y: INTEGER);

procedure AcceptSG(var data: INTEGER);

procedure rootMENU;

procedure drawMENU;

procedure jumpMENU;

procedure eraseMENU;

procedure optionsMENU;

procedure COMs;

procedure MENUs;

procedure Value(ss: STRING; jj: INTEGER; var ij: REAL);

procedure Defaults;
```

```
                    implementation
(********_____********)
Procedure GetGraph;
var d,m: INTEGER;
begin
  DetectGraph(d,m);
  Initgraph(d,m,'c:\TP5');
  Cleardevice
end;

procedure GetBar(fillclr,x,y,xx,yy,drwclr: INTEGER);
begin
  setfillstyle(1,fillclr);
  bar(x,y,xx,yy);
  setcolor(drwclr)
end;

procedure setting;
begin
  GetBar(1,0,25,GetmaxX,GetmaxY,11)
end;

Procedure Posn;
begin
  GetBar(3,GetMaxX-65,12,GetmaxX-31,23,15);
  GetBar(3,GetMaxX-30,12,GetmaxX-3,23,15);
  str(x1,s);
  OutTEXTxy(GetmaxX-62,14,s);
  str(y1,s);
  OutTEXTxy(GetmaxX-30,14,s);
  moveto(x0,y0)
end;

procedure Diagonals;
begin
  if not badCOMMAND then
  case ch of
     #61: arc(GetmaxX-135,20,0,180,5);
     #62: arc(GetmaxX-128,17,270,90,5);
     #63: begin
            line(GetmaxX-135,14,GetmaxX-129,20);
            line(GetmaxX-134,14,GetmaxX-128,20);
            line(GetmaxX-133,20,GetmaxX-128,20);
            line(GetmaxX-128,16,GetmaxX-128,20)
          end;
     #64: begin
            line(GetmaxX-135,14,GetmaxX-141,20);
            line(GetmaxX-134,14,GetmaxX-140,20);
            line(GetmaxX-135,20,GetmaxX-140,20);
            line(GetmaxX-141,16,GetmaxX-141,20)
          end;
     #65: begin
            line(GetmaxX-135,20,GetmaxX-129,14);
            line(GetmaxX-134,20,GetmaxX-128,14);
            line(GetmaxX-133,14,GetmaxX-128,14);
            line(GetmaxX-128,18,GetmaxX-128,16)
          end;
     #66: begin
            line(GetmaxX-135,14,GetmaxX-129,20);
            line(GetmaxX-134,14,GetmaxX-128,20);
            line(GetmaxX-135,14,GetmaxX-130,14);
            line(GetmaxX-135,16,GetmaxX-135,18)
```

```
        end
    end
end;

Procedure Command;
begin
  GetBar(3,GetMaxX-222,12,GetmaxX-70,23,15);
  OutTEXTxy(GetmaxX-220,14,Com);
  Diagonals;
  moveto(x0,y0)
end;

procedure Warnning;
begin
  com := '    Bad Command';
  badCOMMAND := true;
  Command;
  gotoXY(1,1);
  if bell then
    write(chr(7))
end;

procedure EscBar(message:STRING);
begin
  GetBar(1,1,GetmaxY-13,126,GetmaxY,13);
  OutTEXTxy(1,GetmaxY-9,message)
end;

procedure MenuBar;
begin
  GetBar(6,0,0,GetmaxX-230,23,15)
end;

Procedure ClrBrd;
begin
  setting;
  EscBar('                ');
  OutTEXTxy(1,GetmaxY-9,'X = GOTO DESIGN');
  restorePART;
  setcolor(10);
  line(0,GetmaxY-14,GetmaxX,GetmaxY-14);
  Defaults
end;

Procedure ClrMenu;
begin
  GetBar(0,0,0,GetmaxX,24,1);
  GetBar(0,0,GetmaxY-14,GetmaxX,GetmaxY,1)
end;

Procedure clrPART;
begin
  GetBar(1,0,GetmaxY-13,GetmaxX-237,GetmaxY,15)
end;

Procedure restorePART;
begin
  if drawing then
    EscBar('Esc ends DRAW');
  if jumping then
    EscBar('Esc ends JUMP');
  if erasing then
    EscBar('Esc ends ERASE');
  if options then
    EscBar('Esc ends OPTIONS');
  GetBar(6,126,GetmaxY-13,GetmaxX-237,GetmaxY,11);
  OutTEXTxy(126,GetmaxY-9,'-/+ = MENU  OFF/ON   ^D =
  Restart');
  setcolor(15)
end;

procedure Save;
begin
  Getmem(syzP,ImageSize(0,0,GetmaxX,GetmaxY));
  Getimage(0,0,GetmaxX,GetmaxY,syzP^);
  Putimage(0,0,syzP^,XORput)
end;

procedure Rtrv;
begin
  Putimage(0,0,syzP^,XORput) ;
  Setbkcolor(4)
end;

Procedure Pen;

  procedure move_line(m1,m2,l1,l2: INTEGER);
  begin
    moveto(m1,m2);
    lineto(l1,l2)
  end;

begin
   setcolor(10);
   move_line(2,1,2,6);
   move_line(8,3,2,1);
   move_line(2,6,8,20);
   move_line(14,18,8,3);
   move_line(8,10,2,6);
   move_line(6,7,12,18);
   setfillstyle(1,10);
   Fillpoly(5,polygon);
   GetMem ( Penna , ImageSize( 1,1,14,20));
   GetImage( 1,1,14,20,Penna^) ;  { memorize Pen  }
   PutImage( 1,1 ,Penna^,XORput); { put pen at 1,1}

   GetMem( blnk,ImageSize(
   GetmaxX-3,3,GetmaxX-3,3)) ;
   GetImage( GetmaxX-3,3,GetmaxX-3,3,blnk^) ;
   PutImage( GetmaxX-3,3,blnk^,XORput);

   GetMem( blnkE,ImageSize( 10,10,1,1) ) ;
   GetImage( 10,10,1,1,blnkE^) ;
   PutImage( 10,10,blnkE^,XORput);

   setcolor(15);
   moveto(x0,y0)
end;

procedure Help;
begin
  GetBar(1,100,4,420,98,13);
  OutTextxy(100,28,'* Use one of D, J, E, O, X, -, +, or ^D;');
  OutTextxy(100,40,'* OPTIONS  controls  SCALE,  GRID,  &
```

```
SOUND;');
  OutTextxy(100,52,'* COMMAND WINDOW displays
  commands used;');
  OutTextxy(100,64,'* Lower-left-corner shows exit command;');
  OutTextxy(100,76,'* Pen(Upper-right) shows xy coordinates.');
  OutTextxy(100,91,'          Press Esc to continue ...');
  GetMem(hlp,ImageSize(100,4,420,98));
  GetImage(100,4,420,98,hlp^)
end;

procedure GetHelp;
var ch3: CHAR;

  procedure clean;
  begin
    putImage(45,7,hlp^,XORput);
    EscBar('X = GOTO DESIGN');
    Com := '    ROOT MENU';
    Command
  end;

begin
  ch := #3;
  Com := ' HELP ON ROOT MENU';
  Command;
  EscBar('Esc ends HELP');
  setBkcolor(14);
  putImage(45,7,hlp^,XORput);

  ch := Upcase(readkey);
  if ch = #27 then
    clean
  else
    warnning;
  if ch = #0 then
  begin
    ch  :=  readkey;
    ch := #1
  end;
  if ch < > #27 then
  repeat
    ch := Upcase(readkey);
    ch3 := #3;
    if ch < > #27 then
      warnning;
  until ch = #27;
  if ch3 = #3 then
    clean;
  ch := #1
end;

procedure GetPen( x,y: INTEGER ) ;
begin
   if x1 < 10 then
     x0 := GetMaxX-10; { Left Limit  }
   if y1 < 25 then
     y0 := GetmaxY-15;  { Top Limit   }
   if x1 > GetmaxX-10 then
     x0 := 10;  { Right Limit }
   if y1 > GetmaxY-15 then
     y0 := 25;  { Bottom Limit}
   PutImage(x,y,Penna^,XORPut); { move pen to x,y }
   Posn; { Shows the current x-y coordinates of pen }
   Command { Displays information in Command Window }
end;

procedure GetBlank(x,y: INTEGER);
begin
  PutImage (x,y,blnk^,COPYput)
end;

procedure GetBlankE(x,y: INTEGER);
begin
  if y > 35 then
    if y < GetmaxY-15 then
      PutImage (x-2,y-10,blnkE^,COPYput)
end;

procedure AcceptSG(var data: INTEGER);
var
  k   : BYTE;
  Okay: BOOLEAN;
  StrD  : STRING[6];
  StrData: array [1..6] of CHAR;

  begin
    k := 1;
    repeat
      Okay := false;
      repeat
        ch := readkey;
        case ch of
          '0'..'9',' ': Okay := true;
                #27: Exit;
          else
            warnning
        end
      until Okay;
      StrData[k] := ch;
      k := k+1
    until ch = ' ';  { Space bar }
    for i := 1 to k-1 do
    begin
      if i = 1 then
        StrD := StrData[i];
      if i > 1 then
        StrD := StrD + StrData[i]
    end;
    Delete(StrD,k-1,6);
    val(StrD,Data,i);
    if ch = 'S' then
      Com := '  OPTIONS  SCALE';
    if ch = 'G' then
      Com := '  OPTIONS  GRID';
    Command
end;

procedure rootMENU;
begin
   drawing := false;
   jumping := false;
   erasing := false;
   options := false;
   Eraze := 0;
```

```
    badCOMMAND := false;
    if GetGraphmode = 2 then
      SetBkcolor(1);
    setColor(11);
    restorePART;
    GetBar(1,0,GetmaxY-13,126,GetmaxY,13);
    OutTEXTxy(1,GetmaxY-9,'X = GOTO DESIGN');
    MenuBar;
    GetBar(1,GetmaxX-230,0,GetmaxX,23,15);

    OutTEXTxy(GetmaxX-199,2,'COMMAND WINDOW');
    Com := '    ROOT MENU';
    OutTEXTxy(GetmaxX-50,2,'Pen');

    if GetGraphMode = 2 then
      setColor(11);
    setTEXTstyle(2,0,5);
    OutTEXTxy(60,9,'D');  OutTEXTxy(61,9,'D');
    OutTEXTxy(110,9,'J'); OutTEXTxy(111,9,'J');
    OutTEXTxy(160,9,'E'); OutTEXTxy(161,9,'E');
    OutTEXTxy(215,9,'O'); OutTEXTxy(216,9,'O');
    OutTEXTxy(283,9,'X'); OutTEXTxy(284,9,'X');
    OutTEXTxy(320,9,'H'); OutTEXTxy(321,9,'H');

    setcolor(15);
    setTEXTstyle(2,0,4);
    OutTEXTxy(68,11,'RAW');
    OutTEXTxy(118,11,'UMP');
    OutTEXTxy(168,11,'RASE');
    OutTEXTxy(223,11,'PTIONS');
    OutTEXTxy(278,10,'E');
    OutTEXTxy(291,10,'IT');
    OutTEXTxy(328,11,'ELP');

    SetTEXTstyle(0,0,0);
    setColor(10);
    line(0,24,GetmaxX,24);
    line(0,GetmaxY-14,GetmaxX,GetmaxY-14);
    setcolor(0);
    GetPen(x0,y0)
end;

procedure drawMENU;
begin
    badCOMMAND := false;
    Eraze := 0;
    if GetGraphmode = 2 then
      SetBkcolor(1);
    EscBar('Esc ends DRAW');
    MenuBar;
    OutTEXTxy(310,3,'Arrow keys ');
    OutTEXTxy(310,16,'F3 - F8  ');
    setcolor(15);

    if GetGraphMode = 2 then
      setColor(11);
    setTEXTstyle(2,0,5);
    OutTEXTxy(1,10,'R      I');
    OutTEXTxy(2,10,'R      I');
    OutTEXTxy(110,10,'C      V    N');
    OutTEXTxy(111,10,'C      V    N');
    OutTEXTxy(253,10,'E');
    OutTEXTxy(254,10,'E');

    setcolor(15);
    setTEXTstyle(2,0,4);
    OutTEXTxy(9,12,'ESISTOR  NDUCTOR');
    OutTEXTxy(119,12,'APACITOR  CCS');
    OutTEXTxy(224,12,'ODE');
    OutTEXTxy(262,12,'ARTH');

    SetTEXTstyle(0,0,0);
    setColor(15)
end;

procedure jumpMENU;
begin
    if GetGraphmode = 2 then
      SetBkcolor(1);
    EscBar('Esc ends JUMP');
    MenuBar;
    OutTEXTxy(70,3,'Arrow-keys      Home     PageUp');
    OutTEXTxy(70,15,'F5 - F8        End      PageDown');
    setcolor(15)
end;

procedure eraseMENU;
begin
    if GetGraphmode = 2 then
      SetBkcolor(1);
    EscBar('Esc ends ERASE');
    MenuBar;
    OutTEXTxy(8,14,' Use F5 - F8 or the arrow keys ...');
    setcolor(15)
end;

procedure optionsMENU;
begin
  Eraze := 0;
  if GetGraphmode = 2 then
    SetBkcolor(1);
  clrPART;
  EscBar('Esc ends OPTIONS');
  MenuBar;
  setcolor(15);

  if GetGraphMode = 2 then
    setColor(11);
  setTEXTstyle(2,0,5);
  OutTEXTxy(10,10,'S');
  OutTEXTxy(11,10,'S ');
  OutTEXTxy(60,10,'G');
  OutTEXTxy(61,10,'G');
  OutTEXTxy(120,10,'U');
  OutTEXTxy(121,10,'U');
  OutTEXTxy(175,10,'V');
  OutTEXTxy(176,10,'V');
  OutTEXTxy(210,10,'R');
  OutTEXTxy(211,10,'R');
  OutTEXTxy(280,10,'M');
  OutTEXTxy(281,10,'M');

  setcolor(15);
  setTEXTstyle(2,0,4);
```

```
  OutTEXTxy(19,12,'CALE');
  OutTEXTxy(69,12,'RID');
  OutTEXTxy(107,12,'SO');
  OutTEXTxy(129,12,'ND');
  OutTEXTxy(162,12,'SA');
  OutTEXTxy(184,12,'E');
  OutTEXTxy(219,12,'ETRIEVE');
  OutTEXTxy(289,12,'ODIFY');

  SetTEXTstyle(0,0,0);
  setColor(15)
end;

procedure COMs;
  procedure COMcases;
  begin
    if not badCOMMAND then
    begin
      if drawing then
      case ch of
        'R': Com := Com + 'RESISTOR ';
        'I': Com := Com + 'INDUCTOR ';
        'C': Com := Com + 'CAPACITOR ';
        'V': Com := Com + 'VCCS ';
        'N': begin
               if sameNode then
                 nN := nN-1;
               str(nN,s);
               Com := Com + 'NODE ' + s;
               if sameNode then
                 nN := nN+1
             end;
        'E': Com := Com + 'GROUND '
      end;
      case ch of
        #72: Com := Com + ' '+#24 ;
        #75: Com := Com + #27 ;
        #77: Com := Com + '  '+#26 ;
        #80: Com := Com + ' '+#25
      end;
      if options then
      case ch of
        'S': Com := Com + ' SCALE ';
        'G': Com := Com + ' GRID ';
        'V': Com := Com + ' SAVE ';
        'R': Com := Com + ' RETRIEVE ';
        'M': Com := Com + ' MODIFY '
      end
    end
  end;

begin
  if drawing and
    (chh < > 'R') and (chh < > 'I') and
      (chh < > 'C') and (chh < > 'V') then
        Com := 'DRAW ';
  if jumping then
    Com := ' JUMP   ';
  if erasing then
    Com := ' ERASE  ';
  if options then
    Com := ' OPTIONS';
    COMcases;
    Command;
    badCOMMAND := false
end;

procedure MENUs;
begin
  if drawing then
    drawMENU;
  if jumping then
    jumpMENU;
  if erasing and (chE='1') then
    eraseMENU
end;

procedure Value(ss: STRING; jj: INTEGER; var ij: REAL);
var strD: STRING; ii: INTEGER;
begin
  s := '... value in '+ss+',  '+#17;
  ii := length(s);
  OutTEXTxy(8,14,s);
  s := #196#217+' when over ————————————————————————';
  ii := ii+length(s);
  OutTEXTxy(13+8*jj-1,14,s);
  GotoXY(ii-12,1);
  read(ij)
end;

procedure Defaults;
var SG:STRING;
begin
  if ch2 < > #43 then
  begin
    nR := 0;
    nL := 0;
    nC := 0;
    nGm := 0;
    nN := 0;
    nP := 0;
    nT := 0;
    vR := 0;
    vL := 0;
    vC := 0;
    vGm := 0;
    Eraze := 0;
    chNODE := '+';
    Scl := 1;
    Grid := 10;
    bell := true;
    sameNode := false;
    earthh := true;
    x0 := GetmaxX div 2;
    y0 := GetmaxY div 2;
    x1 := x0;
    y1 := y0;
    for i := 1 to nodes do
    for j := 1 to 2 do
    where[i,j] := 0;
    moveto(x0,y0)
  end;
  GetBar(1,GetmaxX-236,GetmaxY-13,
  GetmaxX,GetmaxY,14);
```

```
  str(Scl,SG);
  OutTEXTxy(GetmaxX-236,GetmaxY-9,' Scale=' + SG);
  str(grid,SG);
  OutTEXTxy(GetmaxX-146,GetmaxY-9,'Grid=' + SG);
  if bell then
    OutTEXTxy(GetmaxX-50,GetmaxY-9,'SOUND')
  else
    OutTEXTxy(GetmaxX-65,GetmaxY-9,'NO-SOUND')
end;
END. { OF THE WHOLE UNIT ROOT }


{File name: COMPNNTS.PAS}

UNIT COMPNNTS;

                interface
(*********  ————————————————  *******)

(********************)
USES GRAPH, CRT, ROOT;
(********************)
procedure handleMENU(message:string;j:integer; var ij:real);
procedure RR(scl,dir:INTEGER);
procedure LL(scl,dir:INTEGER);
procedure CC(scl,dir:INTEGER);
procedure VCCS(scl,dir:integer);
procedure NODE;
procedure EARTH;

                implementation
(****  ————————————————————————  *****)

Procedure handleMENU(message:string;j:integer; var ij:real);
begin
  Posn;
  MenuBar;
  Value(message,j,ij);
  RestorePart;
end; { of handleMENU }

Procedure RR(scl,dir:INTEGER);
var xR,yR: integer;
  Procedure horizontalR;
  begin
    line(x0,y0,x0+scl*9,y0);
    line(x0+scl*40,y0,x0+scl*50,y0);
    rectangle(x0+scl*9,y0-scl*3,x0+scl*40,y0+scl*3);
    x1 := x0+scl*50;
    y1 := y0;
    xR := round(x0+(x1-x0)/2);
    yR := y0-scl*5;
    outTEXTxy(xR-10,yR-7,'R');
    str(nR,s);
    outTEXTxy(xR,yR-7,s)
  end; { of forizontalR }

  Procedure verticalR;
  begin
    line(x0,y0,x0,y0+scl*9);
    line(x0,y0+scl*40,x0,y0+scl*50);
    rectangle(x0-scl*3,y0+scl*9,x0+scl*3,
    y0+scl*40);
    x1 := x0;
    y1 := y0+scl*50;
    yR := round(y0+(y1-y0)/2);
    xR := x0+scl*5;
    outTEXTxy(xR,yR,'R');
    str(nR,s);
    outTEXTxy(xR+8,yR,s)
  end;{ of verticalR }

begin { RR }
  nR := nR+1;
  nT := nT+1;
  setColor(15);
  case dir of
    1: horizontalR; {horizontal, left to right.}
   10: begin {horizontal, right to left.}
         if done = 9 then
         begin
           x0 := x0 - 5;
           GetBar(black,x0,y0,x0+4,y0,black);
         end;
         SetColor(white);
         x0 := x0-50*scl;
         horizontalR;
         x1 := x1-50*scl;
         if done = 9 then
           GetBar(black,x1+45,y1,x1+49,y1,black)
       end;
   11: begin  {vertical, up.}
         if done = 10 then
         begin
           y0 := y0 - 5;
           GetBar(black,x0,y0,x0,y0+4,black);
         end;
         y0 := y0-50*scl;
         SetColor(white);
         verticalR;
         y1 := y1-50*scl;
         if done = 10 then
           GetBar(black,x1,y1+45,x1,y1+49,black)
       end;
  110: verticalR  {vertical, down.}
  end;
  with UGX^ do
  begin
    handleMENU('Ohms',20,X[nT]);
    U[nT,0] := 1;
    U[nT,1] := nR
  end
end;{ of RR }

Procedure LL(scl,dir:INTEGER);
var xR,yR:integer;
  Procedure horizontalL;
  begin
    Line(x0,y0,x0+scl*10,y0);
    Arc(x0+scl*15,y0,0,180,scl*5);
    Arc(x0+scl*25,y0,0,180,scl*5);
    Arc(x0+scl*35,y0,0,180,scl*5);
    Line(x0+scl*40,y0,x0+scl*50,y0);
```

```
    x1 := x0+scl*50;
    y1 := y0;
    xR := round(x0+(x1-x0)/2);
    yR := y0-5*scl;
    outTEXTxy(xR-10,yR-10,'L');
    str(nL,s);
    outTEXTxy(xR,yR-10,s)
  end;{ of horizontalL}

  Procedure verticalL;
  begin
    Line(x0,y0,x0,y0+scl*10);
    Arc(x0,y0+scl*15,270,450,scl*5);
    Arc(x0,y0+scl*25,270,450,scl*5);
    Arc(x0,y0+scl*35,270,450,scl*5);
    Line(x0,y0+scl*40,x0,y0+scl*50);
    x1 := x0;
    y1 := y0+scl*50;
    yR := round(y0+(y1-y0)/2);
    xR := x0+5*scl;
    outTEXTxy(xR+4,yR,'L');
    str(nL,s);
    outTEXTxy(xR+13,yR,s)
  end;{ of verticalL }

begin { LL }
  nL := nL+1;
  nT := nT+1;
  setColor(15);
  case dir of
      1: horizontalL;
     10: begin
           x0 := x0-50*scl;
           horizontalL;
           x1 := x1-50*scl
         end;
     11: begin
           y0 := y0-50*scl;
           verticalL;
           y1 := y1-50*scl
         end;
    110: verticalL
  end;
  with UGX^ do
  begin
    handleMENU('Henrys',22,X[nT]);
    U[nT,0] := 2;
    U[nT,1] := nL
  end
end; { of LL }

Procedure CC(scl,dir:INTEGER);
var xR,yR:integer;
  Procedure horizontalC;
  begin
    Line(x0,y0,x0+scl*24,y0);
    Line(x0+scl*24,y0-scl*3,x0+scl*24,y0+scl*3);
    Ellipse(x0+scl*29,y0,90,270,scl*2,scl*3);
    Line(x0+scl*27,y0,x0+scl*50,y0);
    x1 := x0+scl*50;
    y1 := y0;
    xR := round(x0+(x1-x0)/2);
    yR := y0-scl*5;
    outTEXTxy(xR-9,yR-7,'C');
    str(nC,s);
    outTEXTxy(xR,yR-7,s)
  end; { of horizontalC }

  Procedure verticalC;
  begin
    Line(x0,y0,x0,y0+scl*24);
    Line(x0-scl*3,y0+scl*24,x0+scl*3,y0+scl*24);
    Ellipse(x0,y0+scl*29,0,180,scl*3,scl*2);
    Line(x0,y0+scl*27,x0,y0+scl*50);
    x1 := x0;y1 :=
    y0+scl*50;
    yR := round(y0+(y1-y0)/2);
    xR := x0+scl*5;
    outTEXTxy(xR,yR-5,'C');
    str(nC,s);
    outTEXTxy(xR+9,yR-5,s)
  end; { of verticalC }

begin { CC }
  nC := nC+1;
  nT := nT+1;
  setColor(15);
  case dir of
     1: horizontalC;
    10: begin
          x0 := x0-50*scl;
          horizontalC;
          x1 := x1-50*scl
        end;
    11: begin
          y0 := y0-50*scl;
          verticalC;
          y1 := y1-50*scl
        end;
   110: verticalC
  end;
  with UGX^ do
  begin
    handleMENU('Farads',22,X[nT]);
    U[nT,0] := 3;
    U[nT,1] := nC
  end
end; { of CC }

Procedure VCCS(scl,dir:integer);
var xR,yR: integer;
  procedure horizontalVCCS;
  begin
    line(x0,y0,x0+scl*15,y0);
    circle(x0+scl*25,y0,10*scl);
    line(x0+scl*35,y0,x0+scl*50,y0);
    line(x0+scl*20,y0,x0+scl*30,y0);
    x1 := x0+scl*50;
    y1 := y0;
    xR := round(x0+(x1-x0)/2);
    yR := y0-scl*12;
    outTEXTxy(xR-8,yR-7,'Gm');
    str(nGm,s);
    outTEXTxy(xR+8,yR-7,s)
```

```
  end; { horizontalVCCS }

  procedure verticalVCCS;
  begin
    line(x0,y0,x0,y0+scl*15);
    circle(x0,y0+scl*25,10*scl);
    line(x0,y0+scl*35,x0,y0+scl*50);
    line(x0,y0+scl*20,x0,y0+scl*30);
    x1 := x0;
    y1 := y0+scl*50;
    yR := round(y0+(y1-y0)/2);
    xR := x0+scl*12;
    outTEXTxy(xR,yR-2,'Gm');
    str(nGm,s);
    outTEXTxy(xR+16,yR-2,s)
  end; { of verticalVCCS }

begin { VCCS }
  nGm := nGm+1;
  nT := nT+1;
  setColor(15);
  case dir of
      1:begin
          horizontalVCCS;
          line(x0+scl*26,y0-scl*2,x0+scl*30,y0);
          line(x0+scl*26,y0+scl*2,x0+scl*30,y0)
        end;
     10:begin
          x0 := x0-50*scl;
          horizontalVCCS;
          line(x0+scl*24,y0-scl*2,x0+scl*20,y0);
          line(x0+scl*24,y0+scl*2,x0+scl*20,y0);
          x1 := x1-50*scl
        end;
    11: begin
          y0 := y0-50*scl;
          verticalVCCS;
          line(x0-scl*2,y0+scl*24,x0,y0+scl*20);
          line(x0+scl*2,y0+scl*24,x0,y0+scl*20);
          y1 := y1-50*scl
        end;
   110: begin
          verticalVCCS;
          line(x0-scl*2,y0+scl*26,x0,y0+scl*30);
          line(x0+scl*2,y0+scl*26,x0,y0+scl*30)
        end
  end;
  MenuBar;
  OutTEXTxy(8,14,'... VCCS type:
  1=Gm  2=Gm/s  3=sGm  ———');
  GotoXY(41,1);
  read(UGX^.U[nT,0]);
  MenuBar;
  OutTEXTxy(8,14,
  '... VOLTAGE SENDING NODES: +  -');
  GotoXY(29,1);
  with UGX^ do
  begin
    read(G[nGm,2],G[nGm,3]);
    handleMENU('mhos',20,X[nT]);
    U[nT,0] := -U[nT,0];
    U[nT,1] := nGm
  end
end;{ of VCCS }

Procedure NODE;
begin
  with UGX^ do
  begin
    setColor(15);
    circle(x0,y0,2);
    circle(x0,y0,1);
    x1 := x0;
    y1 := y0;
    if sameNode then
      nN := nN-1;
    nN := nN+1;
    sameNode := false;
    for i := 1 to nN do
    begin
      if (where[i,1] = x1) and (where[i,2] = y1) then
      begin
        sameNode := true;
        U[nT+1,2] := i
      end
    end;
    if chNode = '-' then
    for i := 1 to nN do
    begin
      if (where[i,1] = x1) and (where[i,2] = y1) then
      begin
        sameNode := true;
        U[nT,3] := i
      end
    end;
    if (where[0,1] = x1) and (where[0,2] = y1) then
      sameNode := true;
    if not sameNode then
    begin
      where[nN,1] := x1;
      where[nN,2] := y1;
      str(nN,s);
      if nN < 10 then
        OutTEXTxy(x0-10,y0-12,s)
      else
        OutTEXTxy(x0-19,y0-12,s);
      if nN < 10 then
        circle(x0-7,y0-9,5)
      else
        ellipse(x0-13,y0-9,0,360,10,5);
      if chNode = '-' then
        U[nT,3] := nN;
      if chNode='+' then
        U[nT+1,2] := nN
    end;
    if (nT = 0) and not sameNode then
      U[nT+1,2] := nN;
    chNode := '+';
    ch2 := #2
  end
end; { of NODE }

Procedure EARTH;
begin
```

```
  with UGX^ do
  begin
    setColor(15);
    if earthh then
    begin
      earthh := false;
      circle(x0,y0,2);
      circle(x0,y0,1);
      OutTEXTxy(x0-10,y0-12,'0');
      circle(x0-7,y0-9,5);
      Line(x0,y0,x0,y0+17);
      Line(x0-8,y0+17,x0+8,y0+17);
      Line(x0-6,y0+22,x0+5,y0+22);
      Line(x0-4,y0+27,x0+2,y0+27);
      where[0,1] := x1;
      where[0,2] := y1
    end;
    chNode := '+';
    U[nT,3] := 0;
    x1 := x0;
    y1 := y0;
    ch2 := #2
  end
end; { of EARTH }
END. { OF THE WHOLE UNIT COMPNNTS}


(* ********************************************************)
(*  ANALOPT: analysis and optimization program for ppond.  *)
(*    - consists and links network analysis & optimization routines*)
(*    - written as part of an M. Sc. Thesis                *)
(*        by Solomon Zewde, School of Graduate Studies,    *)
(*        Electrical Engineering Dept., Addis Ababa University. *)
(*                                                         *)
(*              March 1992 - March 1993 G. C.              *)
(* ********************************************************)

{$N+}
PROGRAM ANALOPT;{$R-}

(*---------------------------------------------------------*)
uses crt, Anal0, Anal1;
var
  state: STRING[13];
  nodesOPT, EOnest: BOOLEAN;
  RttOPT, LttOPT, CttOPT, GttOPT: INTEGER;
(*---------------------------------------------------------*)

procedure NTK_FREQS_IO;  {reads network, frequencies, i/o nodes}
begin
  with Let do
  begin
    if not ct then
    begin
      LASTexit := 1;
      if not ioNODES then
        if not M1 then
        begin
          if not freqs then
            if not M2 then
              extractNETinfo;
          if not modify then
    begin
      if not freqs then
        if not M2 then
          M1 := true;
      if not M1 then
      begin
        if not M2 then
        begin
          windows(6);
          getFREQS;
        end;
        M2 := true
      end
    end;
  end;
if not M2 then
begin
  if not modify then
  begin
    if not ioNODES then
    begin
      if symbolic = 3 then
        indicateFILE;
      if M1 then
        MENU1; { M1var is determined here }
      case M1var of
        4: LASTexit := 4;
        5: LASTexit := 16;
      end;
      if LASTexit in [4,16] then
        exit;
      if (M2var > 1) and (M1var = 5) then
      begin
        LASTexit := 4;
        exit
      end
    end;

    if ioNODES then
    begin
      windows(4);
      shadow(' TYPE INPUT - OUTPUT NODES ');
      GotoXY(whereX-30, whereY-3);
      write('   Ready to modify nodes   ');
      window(whereX-27,whereY+3,whereX-1,whereY+3);
    end
    else
      if (state = 'READ_IO_NODES') or newINPUT then
        write('   INPUT NODE, OUTPUT NODE   ');

    if (state = 'READ_IO_NODES') or
      ioNODES or newINPUT then
        begin
          getINT('ENTER INPUT NODE',
            ' INPUT NODE ?',iNODE,22,7);
          getINT('ENTER OUTPUTNODE',
            ' OUTPUT NODE? ',oNODE,22,7)
        end;

    nodesOPT := true;
    state := 'STOP_IO_NODES';
    windows(4);
```

```
            windows(6);
            sNODE := iNODE;
            switch := 0;

            if M2var = 5 then
            begin
              LASTexit := 5;
              exit
            end
          end;
          maxS := Ltt + Ctt + Gm_s + sGm
        end
      end
    end
end; { of NTK_FREQS_IO }

procedure PARALLELS;
var
  addBRANCH,parallelBRANCH: BOOLEAN;
  xCODE,i,ii: INTEGER;

  procedure getNEXT;
  begin
    xCODE := 1;
    if U^[i, 4] > 0 then
      exit;

    parallelBRANCH := true;

    if U^[i, 2] <> U^[ii, 2] then
      parallelBRANCH := false;

    if parallelBRANCH then
      if U^[i, 3] <> U^[ii, 3] then
        exit;

    REPEAT
      if parallelBRANCH then
      begin
        U^[ii, 4] := i;
        U^[i, 4] := ii;
        xCODE := 4;
        exit
      end;
      addBRANCH := false;
      if U^[ii, 2] <> U^[i, 3] then
        exit;
      parallelBRANCH := true;
      if U^[ii, 3] = U^[i, 2] then
        addBRANCH := true
    UNTIL not addBRANCH
  end; { of getNEXT }

begin { of PARALLELS }
  with Let do
  begin
    if U^[ii, 4] > 0 then
      exit;
    for i := ii+1 to RLCGtt do
    begin
      getNEXT;
      if xCODE = 4 then
        exit
    end
  end
end; { of PARALLELS }

procedure FILLincidence; { fill modified incidence matrix V }
var i,j,posEND,negEND: INTEGER;
begin
  for i := 0 to ROWz do
  for j := 0 to tNODES do
  V[i, j] := 0;
  for j := 0 to tNODES do
  begin
    V[ROWz-1, j] := W[j+1];
    V[0, j] := 0;
    V[ROWz, j] := 0
  end;
  for i := 1 to RLCGtt do
  begin
    posEND := U^[i, 2];
    negEND := U^[i, 3];
    V[ROWz, posEND] := V[ROWz, posEND] + 1;
    V[ROWz, negEND] := V[ROWz, negEND] + 1;
    V[V[ROWz, posEND], posEND] := i;
    V[V[ROWz, negEND], negEND] := i
  end
end; { of FILLincidence }

procedure connectionDATA; { stores ct connection in a .CIR file }
var i,j: INTEGER;
  procedure ifRLC( name: CHAR; RLC: SHORTINT );
  begin
    write(name, U^[i, 1]:1);
    if RLC < 10 then
      write(' ');
    if RLC >= 10 then
      write(' ');
    write(fVAR1, name, U^[i, 1]:1);
    if RLC < 10 then
      write(fVAR1, ' ');
    if RLC >= 10 then
      write(fVAR1, ' ')
  end; { of ifRLC }

  procedure ifGm( name: STRING; Gms: SHORTINT );
  begin
    write(name, U^[i, 1]:1);
    if Gms < 10 then
      write(' ');
    if Gms >= 10 then
      write(' ');
    write(fVAR1, name, U^[i, 1]:1);
    if Gms < 10 then
      write(fVAR1, ' ');
    if Gms >= 10 then
      write(fVAR1, ' ')
  end; { of if Gms }

begin { connectionDATA }
  clrscr;
  InputFile('.LOD');
  XRead(fVAR1,InFile);
```

```
    i := 1;
    readln(fVAR1);
    read(fVAR1,U^[i,0]);
    close(fVAR1);
    InputFile('.CIR');
    writeln(InFile, ': NETWORK-CONNECTION-DATA. ');
    Open(fVAR1, InFile);
    for i := 1 to RLCGtt do
    begin
      case U^[i, 0] of
        1 : ifRLC( 'R', Rtt );
        2 : ifRLC( 'L', Ltt );
        3 : ifRLC( 'C', Ctt );
       -1 : ifGm( 'G0', Gtt );
       -2 : ifGm( 'GB', Gtt );
       -3 : ifGm( 'GS', Gtt )
      end;
      for j := 2 to 6 do
      begin
        if i in [RLCtt+1 .. RLCGtt] then
        begin
          case j of
            2: begin
                 write(fVAR1, G[i-RLCtt, 1], '   ',
                   G[i-RLCtt, 2], '   ');
                 write(G[i-RLCtt, 1], '   ',
                   G[i-RLCtt, 2], '   ')
               end;
            3: begin
                 write(fVAR1, U^[i, j], '   ');
                 write(U^[i, j], '   ')
               end;
            4: begin
                 write(fVAR1, U^[i, j-2], '   ');
                 write(U^[i, j-2], '   ')
               end;
            6: begin
                 write(fVAR1, C[i]:14);
                 write(C[i]:14)
               end
          end
        end
        else
          case j of
            2,3: begin
                   write(U^[i, j], '   ');
                   write(fVAR1, U^[i, j], '   ')
                 end;
              6: begin
                   write(C[i]:14);
                   write(fVAR1, C[i]:14)
                 end
          end
      end;
      writeln(fVAR1);
      writeln
    end;
    Close(fVAR1);
    state := 'READ_IO_NODES';
  end; { of connectionDATA }

  procedure GETandMANIPULATEdata;

    { performs several tasks:
        . reads in network, frequencies, in/out nodes;
        . returns an error message if network has input errors;
        . assembles network if network has no input error;
        . fills modified incidence matrix;
        . rearranges modified incidence matrix entries;
        . shows circuit connection data;
        . shows tree generation data, if required. }
    var
      order,rowZERO: BOOLEAN;
      i,ii,jj,plusN,minusN,NODEcount: INTEGER;
      ch: CHAR;
        procedure printERRORmessage;
        var s: STRING;
        begin
          if NODEcount < > tNODES then
          begin
            windows(1);
            s := Copy(InFile, 1, length(InFile)-4);
            s := 'INPUT ERRORS: ';
            s := s + 'NO OUTPUT FOR NETWORK ';
            s := s + Copy(InFile, 1, length(InFile)-4) + '.';
            shadow(#7 + s);
            windows(6);
            ch := readkey;
            Let.LASTexit := 15;
            FreeMem(U, (RLCGtt + 1) * 6 * sizeof(itemUVWG));
            exit
          end
        end; { of printERRORmessage }

    begin { GETandMANIPULATEdata }
      if not Let.evals then
        if not Let.list then
        begin
          NTK_FREQS_IO;
          if Let.LASTexit = 4 then
            Let.LASTexit := 6;
          if Let.LASTexit in [4,16] then
            exit;
          rowZERO := false;
          if Let.LASTexit = 5 then
            rowZERO := true;
          if not Let.M2 then
          begin
            if not Let.ct then
            begin
              if not rowZERO then
              begin
                for jj := 1 to RLCGtt-1 do
                PARALLELS;
                { FILL MODIFIED INCIDENCE MATRIX }
                ROWz := 2;
                for jj := 1 to tNODES+1 do
                begin
                  W[jj] := 0;
                  for ii := 1 to RLCGtt do
                  begin
                    if U^[ii, 2] = jj-1 then
                      W[jj] := W[jj] + 1;
                    if U^[ii, 2] < > jj-1 then
                      if U^[ii, 3] = jj-1 then
```

```
          W[jj] := W[jj] + 1
    end;
    if W[jj] > (ROWz-2) then
      ROWz := W[jj] + 2
end;
FILLincidence;
{ re-arrange modified incidence matrix entries }
NODEcount := 0;
for ii := 1 to V[ROWz-1, 0] do
begin
  plusN := U^[V[ii, 0], 2];
  minusN := U^[V[ii, 0], 3];
  if plusN = 0 then
    plusN := minusN;
  for jj := 1 to V[ROWz-1, plusN] do
  begin
    if V[jj, plusN] = V[ii, 0] then
    begin
      if V[0, plusN] <= 0 then
      begin
        V[0, plusN] := jj;
        NODEcount := NODEcount + 1
      end
    end
  end
end;
REPEAT
  order := true;
  for jj := 1 to tNODES do
  if order then
  begin
    if V[0, jj] <> 0 then
    begin
      if V[ROWz-1, jj] <> 1 then
      begin
        for ii := 1 to V[ROWz-1, jj] do
        if order then
        begin
          if V[0, jj] <> ii then
          begin
            plusN := U^[V[ii, jj], 2];
            minusN := U^[V[ii, jj], 3];
            if plusN = jj then
              plusN := minusN;
            if plusN <> 0 then
            begin
              if V[0, plusN] <= 0 then
              begin
                for i := 1 to
                V[ROWz-1, plusN] do
                begin
                  if V[i, plusN]
                    = V[ii, jj] then
                      V[0, plusN] := i
                end;
                NODEcount := NODEcount + 1;
                order := false
              end
            end
          end
        end
      end
          end
        end
      UNTIL order;
      if NODEcount = tNODES then
        set_UNITY;
      printERRORmessage;
      if Let.LASTexit = 15 then
        exit;
      if NODEcount <> tNODES then
        rowZERO := true
    end;
    if rowZERO then
    begin
      for jj := 1 to tNODES do
      V[0, jj] := 1;
      Let.LASTexit := 1;
      rowZERO := false
    end;
    V[0, sNODE] := 0; { column sNODE of row_0 = 0 }
    Let.LASTexit := 7;
    exit
  end;
  if (GO <> 'EL') and (GO <> 'NO') then
    windows(4);
  connectionDATA
end;
Let.M2 := true { necessary to skip next block }
end;
end; { of GETandMANIPULATEdata }

procedure Analyze;

  procedure treeGENERATIONdata; { show modified incidence matrix}
  var
    i,j: INTEGER;
  begin
    writeln;
    writeln('TREE GENERATION DATA...');
    for i := 0 to ROWz do
    begin
      for j := 0 to tNODES do
      write(V[i, j]:5);
      writeln
    end;
    windows(5)
  end;

  procedure printCOMPONENT; { print components' list used }
  var i: INTEGER;

    procedure GetComponent(message: STRING);
    begin
      writeln(i, ' ', message, U^[i, 1]);
      writeln(fVAR1, i, ' ', message, U^[i, 1])
    end; { of GetComponent }

  begin { printCOMPONENT }
    clrscr;
    writeln('BRANCH - COMPONENT LIST');
    InputFile('.BCL');
    Open(fVAR1, InFile);
    for i := 1 to RLCGtt do
```

```
  begin
    case U^[i, 0] of
      1 : GetComponent('1/R');
      2 : GetComponent('1/SL');
      3 : GetComponent('SC');
     -1 : GetComponent('GM');
     -2 : begin
            writeln(i, ' ', 'GM', U^[i, 1], '/S');
            writeln(fVAR1, i, ' ', 'GM', U^[i, 1], '/S')
          end;
     -3 : GetComponent('SGM')
    end
  end;
  Close(fVAR1)
end; { of printCOMPONENT }

procedure printTRANSFERfunction; {numeric & symbolic forms}
var
  DEGREE, posOVERALL, DECIMALS, OVER: INTEGER;
  CH: CHAR;
  NUMERATOR,DENOMINATOR,s: STRING;

(* DEGREE = exponent of the complex frequency variable s
  posOVERALL = length of the numer - denom divider line
  DECIMALS = total number of significant decimal places
  OVER = (normally) screen lines needed for numer/denom
  ch = selects b/n numeric & symbolic transfer functions *)

  procedure symbolicTF(title, polynomial: STRING);
  (* title = Numerator / Denomenator Symbolic Terms
    polynomial = used to collect numer / denom terms *)
  var
    j: INTEGER;
    TERM : STRING; {anitem of denomerator / NUMERATOR}
    numDENOM: STRING[5]; {determined from T. F. analysis }
    fVAR3: TEXT;
  begin
    writeln(title);
    for j := 0 to maxS do
    begin
      write('S^', j);
      InputFile('.SYM');
      XRead(fVAR3, InFile);
      begin
        while not EOF(fVAR3) do
        begin
          readln(fVAR3, numDENOM, DEGREE, TERM);
          if numDENOM = polynomial then
            if DEGREE = j then
              write(TERM)
        end;
        writeln
      end;
      close(fVAR3)
    end
  end; { of symbolicTF }

  procedure numericTF(numDENOM: INTEGER);
  var j, posPREVIOUS,
     posCURRENT, posNUMERATOR: INTEGER;

    procedure checkLINE;
    begin
      posCURRENT := whereX;
      if posPREVIOUS > posCURRENT then
        OVER := OVER + 1;
      posPREVIOUS := posCURRENT
    end; { of checkLINE }

    procedure initialSETTINGS;
    begin
      posPREVIOUS := whereX;
      posCURRENT := posPREVIOUS;
      OVER := 1
    end; { of initialSETTINGS }

    procedure finalSETTINGS;
    begin
      if OVER > 1 then
        posOVERALL := X2
      else
        posOVERALL := posCURRENT
    end; { of finalSETTINGS }

  begin { numericTF }
    DENOMINATOR := '';
    initialSETTINGS;
    for j := maxS downto 0 do
    begin
      if D[numDENOM, j] < > 0 then
        if j < > 0 then
        begin
          write(D[numDENOM, j]:0:DECIMALS,'S',chr(024),j);
          str(D[numDENOM,j]:0:DECIMALS,s);
          DENOMINATOR := DENOMINATOR + s + 'S'+
          chr(024);
          str(j,s);
          DENOMINATOR := DENOMINATOR + s;
          checkLINE;
          if (D[numDENOM, j] < > 0) and
            (D[numDENOM, 0] < > 0) then
              write(' + ');
          if (D[numDENOM, j] < > 0) and
            (D[numDENOM, 0] < > 0) then
          DENOMINATOR := DENOMINATOR + ' + ';
          checkLINE
        end
      else
        begin
          write(D[numDENOM, j]:0:DECIMALS);
          str(D[numDENOM,j]:0:DECIMALS,s);
          DENOMINATOR := DENOMINATOR + s;
          checkLINE
        end
    end;
    if numDENOM = 1 then
      for j := 1 to 2 do
        writeln;
    finalSETTINGS;
    case numDENOM of
      1: begin
           NUMERATOR := DENOMINATOR;
           posNUMERATOR := posOVERALL
         end;
```

```
        2: if posNUMERATOR > posOVERALL then
             posOVERALL := posNUMERATOR
      end
    end; { of numericTF }

begin { printTRANSFERfunction }
  clrscr;
  window(X2 div 2 - 4, Y2 div 2, X2 div 2 + 30, Y2 div 2 + 5);
  TextAttr := LightCyan + 16*Red;
  clrscr;
  writeln;
  writeln('  Form of transfer function ?');
  writeln('         1 = NUMERIC T.F.');
  write('         2 = SYMBOLIC T.F.');
  repeat
    window(X2 div 2 + 12, Y2 div 2 + 4, X2 div 2 + 29,
    Y2 div 2 + 5);
    ch := readkey;
    if not (ch in ['1','2']) then
      write(^G,ch,' -- Entry Error')
  until ch in ['1','2'];
  windows(4);
  initVIDEO;
  clrscr;
  case ch of
    '1': begin
           getINT('Specify no. of decimal places ',
             'DECIMALS ? ', DECIMALS, 34, 7);
           windows(4);
           write('TRANSFER FUNCTION BETWEEN NODES ',
           iNODE,' AND ', oNODE, ' TO ', DECIMALS, '
           DECIMAL PLACE');
           if DECIMALS > 1 then
             writeln('S =  ')
           else
             writeln(' = ');
           writeln;
           numericTF(1);
           numericTF(2);
           GotoXY(1,whereY - OVER);
           for OVER := 1 to posOVERALL do
           write(chr(196));
           if posOVERALL < 80 then
           begin
             writeln;
             OVER := length(DENOMINATOR);
             for DEGREE := 1 to OVER do
             write(' ');
             GotoXY(1 + (posOVERALL - OVER) div 2, 5);
             write(DENOMINATOR)
           end;
           OVER := length(NUMERATOR);
           if OVER < 80 then
           begin
             GotoXY(1,3);
             for DEGREE := 1 to OVER do
             write(' ');
             GotoXY(1 + (posOVERALL - OVER) div 2, 3);
             write(NUMERATOR)
           end
         end;
    '2': begin
           symbolicTF('Numerator Symbolic Terms', 'numer');
           symbolicTF('Denominator Symbolic Terms', 'denom')
         end
  end;
  windows(5);
  Let.switch := 1
end; { of printTRANSFERfunction }

procedure print_TF_SENSITIVITY; { TF = transfer function }
var i,j,k: INTEGER;
begin
  clrscr;
  writeln('TRANSFER FUNCTION SENSITIVITY: ');
  GotoXY(whereX+15, whereY);
  writeln('D(NUM)/D(CPT)       D(DEN)/D(CPT)');
  for i := 1 to RLCGtt do
  begin
    k := U^[i, 0];
    if k < 0 then
      k := 4;
    writeln('COMPONENT ', copy(compSET, 2*k - 1, 2), U^[i, 1]);
    for j := 0 to maxS do
    writeln('S^ ', j:2, '       ',
    E^[i, j]:11:4, '           ', F^[i, j]:11:4)
  end;
  Let.switch := 2
end;

BEGIN { Analyze }
(* Performs network analysis using information from file
   "InFile" or directly from keyboard. Units Anal0 and
   Anal1 are used. Result is stored and used for eventual
   network optimization.

   Five files are also made available:

     File name              description
     ---------              -----------
     InFile.CIR ..... circuit connection information;
     InFile.BCL ..... branch-component list;
     InFile.TRL ..... tree list;
     InFile.SYM ..... symbolic transfer function list;
     InFile.LOD ..... coded network information. *)

initials;
state := 'READ_IO_NODES';
with Let do
REPEAT
 REPEAT
  REPEAT
   REPEAT
    REPEAT
     REPEAT
      REPEAT
       REPEAT
        REPEAT
         REPEAT
          symbolic := 1;
          REPEAT
            if not more then
              GETandMANIPULATEdata;
            if symbolic in [1,3] then
            begin
```

```
    InputFile('.SYM');
    Open(fVAR2, InFile);
  end;
  symbolic := 2;
  if LASTexit = 15 then
  begin
    close(fVAR2);
    halt {exit}
  end;
  if not (LASTexit in [6, 7, 16]) then
  begin
    if not M2 then
      if not evals then
      begin { TREE LIST }
        if not more then
        begin
          aTREES := 0;
          pTREES := 0;
          tTREES := 0;
          V[0, 0] := 0;
          initDEF;
          if M1var = 1 then
          begin
            windows(4);
            printCOMPONENT;
            InputFile('.TRL');
            Open(fVAR1, InFile)
          end
        end;
        TREES;
        if LASTexit < > 14 then
        begin
          writeln('There are a total of ',tTREES,
            ' trees for this network.');
          writeln(fVAR1,'There are a total of ',
            tTREES,' trees for this network.');
          close(fVAR1);
          windows(5)
        end
      end;
    M2 := true;
    if LASTexit = 14 then
      M2 := false;
    if not M2 then { skip block }
    begin
      { TRANSFER FUNCTION GENERATION }
      if V[0, 0] < > 1 then
      begin
        TF_plus_sensty;
        more := true
      end;
      if tTREES = 1 then
      begin
        shadow('Please wait.');
        windows(6);
        write('COMMAND: ');
        HighVideo;
        with Let do
        case M1var of
          2: write('DO TF');
          3: write('DO SE');
        end;
          initVideo
        end;
        if not more then
        begin
          windows(4);
          if M1var = 2 then
            close (fVAR2);
          if M1var = 2 then
            printTRANSFERfunction
        end
      end
    end;
    if LASTexit in [6, 7, 16] then
      more := false;
  UNTIL not more;
  if M1var < > 2 then
    close(fVAR2);
  if not (LASTexit in [6, 7, 16]) then
  begin
    if not evals then
    begin
      if not M2 then
        if M1var-1 = 1 then
          M2 := true;
      if not M2 then
      begin
        if M1var-1 = 2 then
          windows(4);
        print_TF_SENSITIVITY
      end
    end;
    CTFF := false;
    REPEAT
      check_TF_error;
      M2 := true;
      if LASTexit = 8 then
        M2 := false;
      if not M2 then
      begin
        if POINTS = 0 then
          freqs := true;
        if (M1var-1) > switch then
          CTFF := true;
        if not CTFF then
        begin
          check_FREQ_RESP_error;
          FR_plus_sensty;
          windows(5);
          LASTexit := 1
        end
      end
    UNTIL not CTFF
  end;
  if not (LASTexit = 16) then
  begin
    M2 := true;
    if LASTexit in [6, 7] then
      M2 := false;
    if not M2 then
    begin
      if M2var > 0 then
        M2 := true;
```

```
            LASTexit := 1;
            if not M2 then
            begin
              windows(5);
              switch := 0
            end
          end;
          MENU2; { M2var is determined here }
          if (GO <> 'EL') and (GO <> 'NO') then
            windows(4);
          M2 := false;
        end;
        initM2VARS;
        filterM2VARS;
        LASTexit := 1;
       UNTIL not ct
      UNTIL not evals
     UNTIL not list
    UNTIL not ioNODES
   UNTIL not freqs
  UNTIL not M1
 UNTIL not modify;
 if newINPUT then
   FreeMem(U, (RLCGtt + 1) * 6 * sizeof(itemUVWG));
UNTIL not newINPUT
UNTIL not continue
end; { of procedure analyze }

procedure OPTIMIZATION_overall;
(* consists of all the optimization routines. Files InFile.SPC,
   whcich contains the design criteria or specifications, and
   InFile.OPT, which contains the optimum component values, are
   generated. Once created, file InFile.SPC is re-used when the
   same network is to be considered again.                    *)
type
  itemABAT = EXTENDED;
  arrayABAT = array [1..21, 1..21] of itemABAT;
  arrayOPTIM = array [1..21] of itemABAT;
  matsABAT = ^arrayABAT;
  matsOPTIM = ^arrayOPTIM;
var
  ERROR: matsOPTIM;
  AOPT, BOPT, AOPT_T, inverse, TEMPOMAT: matsABAT;
  x, y, z, zz, errorLEVEL, errorNOW, errorPRE,
  errorNOW2,cutBACK: REAL;
  pd, CONJUGATE: BOOLEAN;
  lambda: EXTENDED;
  ch: CHAR;
  i, m, sizeA, damping, iteration: INTEGER;

  procedure transposeA;
  var i,j: INTEGER;
  begin
    for i:=1 to POINTS do
    for j:=1 to sizeA do
    AOPT_T^[j,i] := AOPT^[i,j];
  end; { of transposeA }

  Procedure multMAT(AA,BB: matsABAT;
                    n1,n2,n3: INTEGER;
                      var CC: matsABAT);
  var i,j1,j2 :integer;
  begin
    for i := 1 to n1 do
    for j1 := 1 to n3 do
    begin
      CC^[i,j1] := 0.0;
      for j2 := 1 to n2 do
      CC^[i,j1] := CC^[i,j1] + AA^[i,j2] * BB^[j2,j1]
    end
  end; { of multMAT }

Procedure multVECT( AA: matsABAT;
                    BB: matsOPTIM;
                    n1,n2: INTEGER;
                    var CC: arrayC);
var i,j :integer;
begin
  for i := 1 to n1 do
  begin
    CC[i] := 0.0;
    for j := 1 to n2 do
    CC[i] := CC[i] + AA^[i,j] * BB^[j]
  end
end; { of multVECT }

procedure invert; { Gauss - Jordan ellimination is employed }
var tempo: EXTENDED;

  Procedure forwardELLIM;
  var i,j1,j2,k,n: INTEGER;
  begin
    for i:=1 to RLCGtt do
    for j1:=1 to RLCGtt do
    begin
      inverse^[i,j1]:=0;
      inverse^[i,i]:=1;
    end;
    i:=1;
    for n:=1 to RLCGtt do
    begin
      if BOPT^[n,n]=0.0 then { row pivotting }
      begin
        j2:=n;
        while (BOPT^[n,n] = 0.0) and (j2 < RLCGtt ) do
        begin
          for k:=1 to RLCGtt do
          begin
            tempo:=BOPT^[n,k]; { row interchange }
            BOPT^[n,k]:=BOPT^[j2+1,k];
            BOPT^[j2+1,k]:=tempo
          end;
          j2:=j2+1;
        end;
      end;

      if BOPT^[n,n]=0 then { column pivotting }
      begin
        j2:=n;
        while (BOPT^[n,n] = 0.0) and (j2 < RLCGtt ) do
        begin
          for k:=1 to RLCGtt do
          begin
            tempo:=BOPT^[k,n]; { column interchange }
```

```
            BOPT^[k,n]:=BOPT^[k,j2+1];
            BOPT^[k,j2+1]:=tempo;
          end;
          j2:=j2+1;
        end;
      end;
      if BOPT^[n,n]=0 then
      begin
        shadow(#7'COMPUTATIONAL ERRORS MET, NO
        OPTIMIZATION!');
        Let.LASTexit := 17;
        exit
      end;

      tempo:=BOPT^[i,i];
      for j1:=1 to RLCGtt do { normalize }
      begin
        BOPT^[i,j1]:=BOPT^[i,j1]/tempo;
        inverse^[i,j1]:=inverse^[i,j1]/tempo;
      end; { of normalize }
      for i:=i+1 to RLCGtt do { set lower triangular entries 0 }
      begin
        tempo:=BOPT^[i,n];
        for j1:=1 to RLCGtt do
        begin
          BOPT^[i,j1]:=BOPT^[i,j1] - tempo*BOPT^[n,j1];
          inverse^[i,j1]:=inverse^[i,j1] - tempo*inverse^[n,j1];
        end;
      end; { of forward ellimination for current row }
      i:=n+1;
    end;
  end; { of nested procedure forwardELLIM }

  procedure backwardELLIM; { set upper triangular entries 0 }
  var i,j1,j2: INTEGER;
  begin
    for j1:=2 to RLCGtt do
    begin
      for i:=1 to j1-1 do
      begin
        tempo:=BOPT^[i,j1];
        for j2:=1 to RLCGtt do
        begin
          BOPT^[i,j2]:=BOPT^[i,j2]-tempo*BOPT^[j1,j2];
          inverse^[i,j2]:=inverse^[i,j2]-tempo*inverse^[j1,j2];
        end;
      end;
    end;
  end; { of nested procedure backwardELLIM }

begin { invert }
  forwardELLIM;
  if Let.LASTexit = 17 then
    exit;
  backwardELLIM;
end; { of invert }

procedure optimized_elements;
type STRING3 = STRING[3];
  procedure ACCEPT(number: INTEGER; name: STRING3);
  var I, J, Location: INTEGER;
  begin
    if name = ' R?' then
      Location := 0;
    if name = ' L?' then
      Location := Rtt;
    if name = ' C?' then
      Location := Rtt+Ltt;
    if name = 'Gm?' then
      Location := Rtt + Ltt + Ctt;
    for i := 1 to number do
    begin
      write(name,' ');
      readln(J);
      writeln(fVAR1, J + Location )
    end
  end; { of ACCEPT }

begin { optimized_elements }
  with Let do
  if not ill then
  begin
    window(48, Y2-20, X2-14, Y2-20);
    clrscr;
    readln(RttOPT, LttOPT, CttOPT, GttOPT);
    Open(fVAR1, 'MODIFY.');
    ACCEPT(RttOPT, ' R?');
    ACCEPT(LttOPT, ' L?');
    ACCEPT(CttOPT, ' C?');
    ACCEPT(GttOPT, 'Gm?');
    Close(fVAR1)
  end;
  window(48, Y2-20, X2-11, Y2-20);
  write(RttOPT, ' ', LttOPT, ' ', CttOPT, ' ', GttOPT);
end; { of optimized_elements }

procedure OPT_INFO1;
  procedure TITLE(i: INTEGER; message: STRING);
  begin
    TextAttr := magenta + cyan * 16;
    window(4, Y2-i, length(message) + 4, Y2-i);
    write(message);
    window(length(message)+4, Y2-i,
      length(message)+25, Y2-i);
    initVideo;
    clrscr
  end; { of TITLE }

  procedure VALUES;
  begin
    optimized_elements;
    getFREQS;

    window(33, Y2-6, X2-31, Y2-6);
    clrscr;
    if not Let.ill then
      if not nodesOPT then
        readln(iNODE, oNODE);
          write(iNODE,' ', oNODE);

    window(28, Y2-4, X2-31, Y2-4);
    clrscr;
    readln(leastP);
    write(leastP:6);
```

```
    window(29, Y2-2, X2-31, Y2-2);
    clrscr;
    readln(errorLEVEL);
    write(errorLEVEL:16);

    windows(6)
  end; { of VALUES }

begin { OPT_INFO1 }
  windows(5);
  window(X1, Y2-22, X2-10, Y2-1);
  clrscr;
  if not Let.ill then
    write('  Information for optimization ... ')
  else
    write('  modify p and/or error level ... ');
  window(X1+2, Y2-21, X2-2, Y2-1);
  TextAttr := LightGray + brown * 16;
  clrscr;
  TITLE(20, '   Number of R, L, C, VCCS to be optimized ');
  TITLE(18, '   Initial Frequency     ');
  TITLE(16, '   Frequency Increment  ');
  TITLE(14, '   Frequency Scale       ');
  TITLE(12, '   Frequency points      ');
  TITLE(10, '          Gain           ');
  TITLE(8, '         weight          ');
  TITLE(6, '    Input, Output Nodes ');
  TITLE(4, '    Desired level of p  ');
  TITLE(2, '    Desired error level ');
  initVideo;
  VALUES;
  write('  Proceed with optimization, Y/N ?');
  REPEAT
    ch := upcase(readkey);
    if ch = 'N' then
    begin
      GotoXY(1,whereY);
      write('                                  ');
      VALUES
    end;
    GotoXY(1,whereY);
    write('  Proceed with optimization, Y/N ?')
  UNTIL ch ='Y';
  windows(6);
  windows(4)
end; { of OPT_INFO1 }

procedure OPT_INFO2;
var i: INTEGER;
  procedure initWINDOW(w1,w2,w3,w4: INTEGER);
  begin
    window(X1+w1,Y1+w2,X1+w3,Y1+w4);
    clrscr
  end;

begin { OPT_INFO2 }
  window(X1+5,Y1+6,X2-5,Y2-11);
  TextAttr := black + LightGray * 16;
  clrscr;
  window(X1+6,Y1+7,X2-6,Y1+9);
  writeln('Optimization in progress ');
  for i := 1 to X2 - 13 do
    write(chr(205));
  writeln;
  window(X1+31,Y1+7,X2-7,Y1+7);
  initVideo;
  clrscr;
  window(X1+6,Y1+9,X2-7,Y1+11);
  TextAttr := black + black * 16;
  clrscr;
  window(X1+20,Y1+9,X2-21,Y1+11);
  TextAttr := Green + black * 16;
  for i := 1 to 39 do
  write(chr(196));
  if leastP = 2 then
  begin
    GotoXY(8, whereY-1);
    writeln(' LEAST SQUARES OPTIMIZATION ')
  end
  else
  begin
    if leastP < 10 then
      GotoXY(10, whereY - 1)
    else
      GotoXY(9, whereY - 1);
    writeln(' LEAST ',leastP,'TH OPTIMIZATION ')
  end;
  window(X1+6,Y1+10,X2-6,Y1+11);
  GotoXY(2, whereY);
  write('DESIRED ERROR LEVEL ');
  if errorLEVEL = 0 then
    write(chr(61))
  else
    write(chr(243));
  i := RttOPT + LttOPT + CttOPT + GttOPT;
  write(errorLEVEL:16,';  NUMBER OF VARIABLES = ',i);
  TextAttr := Black + LightGray * 16;
  window(X1+6,Y1+11,X2-6,Y1+11);
  write(' ITERATION    SAMPLE    DAMPING      ');
  write('  ATTAINED ERROR LEVEL  ');
  initVideo;
  initWINDOW(X1+ 6, 12, X1+15, 12);
  initWINDOW(X1+21, 12, X1+26, 12);
  initWINDOW(X1+32, 12, X1+43, 12);
  initWINDOW(X2-30, 12, X2-8, 12)

end; { of OPT_INFO2 }

{procedure Damp_All_deltaC; -- an alternative method
var i, j, k, m, variable: INTEGER;
begin
  i := 0;
  damping := 0;
  XRead(varEPS, 'MODIFY.');
  while not EOF(varEPS) do
  begin
    readln(varEPS, variable);
    for m := 1 to RLCGtt do
    if m = variable then
    begin
      i := i + 1;
      if abs(deltaC[i]) > abs(C[m])/2 then
      begin
        for j := 1 to sizeA do
```

```
          deltaC[j] := deltaC[j]*abs(C[m]/(2*deltaC[i]));
          damping := damping + 1
        end
      end
    end;
    Close(varEPS)
end;  of Damp_All_deltaC }

procedure Damp_Single_deltaC;
var i, m, variable: INTEGER;
begin
    i := 0;
    damping := 0;
    XRead(varEPS, 'MODIFY.');
    while not EOF(varEPS) do
    begin
      readln(varEPS, variable);
      for m := 1 to RLCGtt do
      if m = variable then
      begin
        i := i + 1;
        if abs(deltaC[i]) > abs(C[m])/2 then
        begin
          deltaC[i] := deltaC[i]*C[m]/abs(2*deltaC[i]);
          damping := damping + 1
        end
      end
    end;
    Close(varEPS)
end; { of Damp_Single_deltaC }

procedure Test_pd; { Uses Sylvester's expansion }
{ pd returns true if approximate Hessian is positive definite }
var ii,jj,positives,nextMINOR: INTEGER;
    tempo,det: EXTENDED;
    zis: TEXT;

    procedure determinant(leadingMINOR: integer);
    { evaluates the determinant of matrix BOPT^ as det by first
      performing Gaussian ellimination and then taking products
      of terms on the main diagonal. Sign is included by taking
      (-1)^signs, where signs = no. of row/column interchanges. }
    var n,k,i,j,j1,j2,minusONES: INTEGER;
    begin
      i := 1;
      minusONES := 0;
      for n := 1 to leadingMINOR do
      begin
        if BOPT^[n,n] = 0.0 then { row pivotting }
        begin
          minusONES := minusONES + 1;
          j2 := n;
          while (BOPT^[n,n] = 0.0) and (j2 < leadingMINOR) do
          begin
            for k := 1 to leadingMINOR do
            begin
              tempo := BOPT^[n,k]; { row interchange }
              BOPT^[n,k] := BOPT^[j2+1,k];
              BOPT^[j2+1,k] := tempo
            end;
            j2 := j2 + 1
          end
        end;

        if BOPT^[n,n] = 0 then { column pivotting }
        begin
          minusONES := minusONES + 1;
          j2 := n;
          while (BOPT^[n,n] = 0.0) and (j2 < leadingMINOR) do
          begin
            for k := 1 to leadingMINOR do
            begin
              tempo := BOPT^[k,n]; { column interchange }
              BOPT^[k,n] := BOPT^[k,j2+1];
              BOPT^[k,j2+1] := tempo
            end;
            j2 := j2 + 1
          end
        end;

        for i := i+1 to leadingMINOR do
        begin { set lower triangular entries 0 }
          tempo := BOPT^[i,n];
          for j1 := 1 to leadingMINOR do
            BOPT^[i,j1] :=
              BOPT^[i,j1] - tempo*BOPT^[n,j1] / BOPT^[n,n]
        end; { of forward ellimination for current row }
        i := n+1
      end;
      det := Raise(-1,minusONES);
      for i := 1 to leadingMINOR do
        det := det*BOPT^[i,i]
    end; { of determinant }

begin
    for ii := 1 to sizeA do
      for jj := 1 to sizeA do
        TEMPOMAT^[ii,jj] := BOPT^[ii,jj];
    positives := 0;
    for nextMINOR := 1 to sizeA do
    begin
      determinant(nextMINOR);
      if det > 0 then
        positives := positives + 1;
      for ii := 1 to sizeA do
        for jj := 1 to sizeA do
          BOPT^[ii,jj] := TEMPOMAT^[ii,jj]
    end;
    if positives = sizeA then
      pd := true
end;

procedure Gradient_Information;
begin
    V[0,0] := 0;
    initDEF;
    Let.more := false;
    Let.M1var := 3;

    with Let do
    REPEAT
      TREES;
      M2 := true;
      if LASTexit = 14 then
```

```
      M2 := false;
    if not M2 then { else skip block }
    begin
      if V[0, 0] < > 1 then
      begin
        TF_plus_sensty;
        more := true
      end
    end;
    if LASTexit in [6, 7, 16] then
      more := false
  UNTIL not more;

  Let.M2var := 3;
  FR_plus_sensty
end;

procedure Use_Levenberg_Marquardt;
var i,m,variable: INTEGER;
    x, qeew, trace,
    projected_gradient,
    quadratic_form,
    deltaERROR,
    deltaMODEL,
    quadraticRATIO : EXTENDED;
begin
  { compute projected gradient }
  for i :=1 to sizeA do
  begin
    x := 0;
    for m := 1 to POINTS do
    x := ERROR^[m] * AOPT^[m,i] + x;
    TEMPOMAT^[1,i] := - leastP * x
  end;
  x := 0;
  for i := 1 to sizeA do
  projected_gradient := TEMPOMAT^[1,i] * deltaC[i] + x;

  { compute quadratic form }
  x := 0;
  for i := 1 to sizeA do
  for m :=1 to sizeA do
  x := deltaC[m] * BOPT^[m,i] + x;
  TEMPOMAT^[1,i] := x;
  x := 0;
  for i := 1 to sizeA do
  quadratic_form := TEMPOMAT^[1,i] * deltaC[i] + x;

  { compute multiplying factor }
  qeew := 2 * (projected_gradient + errorPRE - errorNOW);
  qeew := qeew / projected_gradient;
  if qeew > 10 then
    qeew := 10;
  if qeew < 2 then
    qeew := 2;

  { compute quadratic ratio }
  deltaMODEL := projected_gradient + 0.5*quadratic_form;
  deltaERROR := errorNOW - errorPRE;
  {if deltaMODEL = 0 then
    deltaMODEL := 1e-20;}
  quadraticRATIO := deltaERROR / deltaMODEL;

  { compute trace }
  trace := 0;
  for i := 1 to sizeA do
  trace := trace + BOPT^[i,i];
  trace := 0.01 * trace / sizeA;

  { proceed with modified Levenberg-Marquardt adjustments ... }
  if quadraticRATIO < 0.25 then
    lambda := qeew {usually 4} * lambda;
  if quadraticRATIO > 0.75 then
  begin
    lambda := lambda / 2;
    if lambda < trace then
      lambda := 0
  end;
  i := 0;
  XRead(varEPS, 'MODIFY.');

  {if quadraticRATIO > 0 then}
  while not EOF(varEPS) do
  begin
    readln(varEPS, variable);
    for m := 1 to RLCGtt do
    if m = variable then
    begin
      i := i + 1;
      C[variable] := C[variable] + deltaC[i]
    end
  end;
  Close(varEPS)
end; { of Use_Levenberg_Marquardt }

procedure CONJUGATE_GRADIENT_METHOD; { an alternative }
label 100,3,4,8,7,18,12,15,10,11,23,21,13,19,22,20,29,53,50;
var
  m,ier,k,icount,iii,N1: INTEGER;
  gg,oldgg,beta,yb,vb,ss,alfa,ambda,est,ya,va,t,z,w:EXTENDED;
  eps,limit,vc,errorPRE2: REAL;
  s: array[1..10] of real;

  procedure prmchng(ambda: real);
  var i,m,variable: integer;
  begin
    for i := 1 to sizeA do
    deltaC[i] := deltaC[i] +  ambda*s[i];
    Damp_Single_deltaC;
    i := 0;
    XRead(varEPS, 'MODIFY.');
    while not EOF(varEPS) do
    begin
      readln(varEPS, variable);
      for m := 1 to RLCGtt do
      if m = variable then
      begin
        i := i + 1;
        C[variable] := C[variable] + deltaC[i]
      end
    end;
    Close(varEPS)
  end;

  procedure compute_error_and_gradient;
```

```
  var m: integer;
  begin
    XRead(varERR, 'ERRMATX.');
    for m := 1 to POINTS do
    readln(varERR, ERROR^[m]);
    close(varERR);

    { save old changes on parameter values }
    Open(varEPS, 'CHANGES.');
    for m := 1 to sizeA do
    writeln(varEPS, deltaC[m]);
    close(varEPS);

    { calculate gradient vector as TEMPOMAT^[m,1] }
    multVECT(AOPT_T, ERROR, sizeA, POINTS, deltaC);
    for m := 1 to sizeA do
    TEMPOMAT^[m,1] := -LeastP * deltaC[m];

    { retrieve changes on parameter values }
    XRead(varEPS, 'CHANGES.');
    for m := 1 to sizeA do
    readln(varEPS, deltaC[m]);
    close(varEPS);

    errorNOW := 0;
    XRead(varEPS, 'EPSMATX.');
    while not EOF(varEPS) do
    begin
      readln(varEPS, x);
      errorNOW := errorNOW + x;
    end;
    close(varEPS);
  end;

begin { CONJUGATE_GRADIENT_METHOD }
  { initialize variables }
  est := 1e-06;
  eps := 1e-10;
  limit := 50;
  CONJUGATE := true;
  icount := 0;
  ier := 0;
  N1 := sizeA + 1;

  { calculate initial error and gradient }
  compute_error_and_gradient;

  { begin iteration loop }
100:for iii := 1 to N1 do
  begin
    icount := icount + 1;
    errorPRE2 := errorNOW;

    { compute square of gradient and check for zero value }
    gg := 0.0;
    for m := 1 to sizeA do
    gg := gg + TEMPOMAT^[m,1]*TEMPOMAT^[m,1];
    if gg = 0 then goto 50;

    {use steepest descent at beginning of every sizeA+1 iterations}
    if iii > 1 then goto 3;
    for m := 1 to sizeA do
    s[m] := -TEMPOMAT^[m,1];
    goto 4;
  3: beta := gg / oldgg;
    for m := 1 to sizeA do
    s[m] := -TEMPOMAT^[m,1] + beta*s[m];

    { begin linear search }
  4: yb := errorNOW;
    vb := 0;
    ss := 0;
    for m := 1 to sizeA do

    { save old parameter values & find directional derivatives }
    begin
      s[m+sizeA] := deltaC[m];
      vb := vb + TEMPOMAT^[m,1]*s[m];
      ss := ss + s[m]*s[m];
    end;

    { check that direction of search will decrease error }
    if vb >= 0 then goto 20;

    { estimate initial step size }
    alfa := 2 * (est-errorNOW)/vb;
    ambda := 1/sqrt(ss);
    if alfa <= 0 then goto 7;

  8: if (alfa * alfa * ss) >= 1 then goto 7;
    ambda := alfa;

  7: alfa := 0.0;

    { double step size until minimum is bounded }
  15: ya := yb;
    va := vb;

    { step parameters along search direction }
    prmchng(ambda);

    { calculate function and gradient for new parameters }
    Gradient_Information;
    compute_error_and_gradient;
    Gradient_Information;

    { calculate new directional derivatives -- if positive
      minimum has been passed -- if zero minimum has been
      found }
    vb := 0;
    for m := 1 to sizeA do
  10: vb := vb + TEMPOMAT^[m,1]*s[m];
    if errorNOW < errorPRE then exit;
    if vb < 0 then goto 11;
    if vb = 0 then goto 12;
    if vb > 0 then goto 21;

    { check to see if error has increased }
  11: if yb > ya then goto 21;

    { double step size and repeat above procedure }
    ambda := ambda + alfa;
    alfa := ambda;
```

```
    { terminate if step size is too large }
    if ss*ambda < 1e10 then goto 15;
23: ier := 3;
    exit;

    { perform cubic interpolation }
21: t := 0;
13: if ambda = 0 then goto 12;
    z := 3 * (ya-yb)/ambda + va +vb;
    w := sqrt(z*z - va*vb);
    alfa := ambda * (vb+w-z) / (vb+ 2*w - va);
    prmchng(t-alfa);

    { terminate if error is less than values at each end
      otherwise reduce interval }
    Gradient_Information;
    compute_error_and_gradient;
    {if errorNOW < errorPRE then exit;}
    if (errorNOW < = ya) and (errorNOW < = yb) then
       goto 12;
    vc := 0;
    for m := 1 to sizeA do
19: vc := vc + TEMPOMAT^[m,1]*s[m];
    if vc > = 0 then goto 18;
    ya := errorNOW;
    va := vc;
    ambda := alfa;
    t := ambda;
    goto 13;
18: yb := errorNOW;
    vb := vc;
    ambda := ambda -alfa;
    goto 21;

    { compute length of change vector }
12: t := 0;
    for m := 1 to sizeA do
    begin
       k := m + sizeA;
       s[k] := deltaC[m] - s[k];
       t := t + abs(s[k]);
    end;

    { check to see if length of change vector is insignificant
      if a t least sizeA+1 iterations have been taken }
    if icount < N1 then goto 22;
    if t < eps then goto 29;

    { terminate if error has decreased during last iteration }
22: if errorNOW > (errorPRE2+eps) then goto 23;
20: oldgg := gg;
    if icount > limit then goto 53;
  end;

  goto 100;

  { check for sufficiently small gradient }
29: if (gg-eps) < = 0 then goto 50;
  if (gg-eps) > 0 then goto 23;

53: ier := 2;
  exit;

50: ier := 1;
  exit

end;

procedure OPTIMIZATION_basic;
var i, m,
    variable: INTEGER;
begin { OPTIMIZATION_basic }
  Let.optimality := true;
  CONJUGATE := false;
  iteration := -1;
  errorNOW := 0;
  errorPRE := 0;
  OPT_INFO1;
  OPT_INFO2;
  New(AOPT);
  New(BOPT);
  New(AOPT_T);
  New(inverse);
  New(TEMPOMAT);
  Open(fVAR3, 'Err_Iter.');
  REPEAT
    iteration := iteration + 1;
    window(X1+6, Y1+12, X1+15, Y1+12);
    write('  ',iteration);

    for i := 1 to POINTS do
    for m := 1 to POINTS do
    BOPT^[i,m] := 0;

    Gradient_Information;

    XRead(varA,'MODIFY.');
    sizeA := 0;
    while not EOF(varA) do
    begin
      readln(varA);
      sizeA := sizeA + 1;
    end;
    close(varA);
    XRead(varA,'AMATRIX.');
    for m:=1 to POINTS do
    begin
      for i:=1 to sizeA do
      readln(varA, AOPT^[m,i])
    end;
    close(varA);

    XRead(varB, 'BMATRIX.');
    for m := 1 to POINTS do
    readln(varB, BOPT^[m,m]);
    Close(varB);

    XRead(varERR, 'ERRMATX.');
    for m := 1 to POINTS do
    readln(varERR, ERROR^[m]);
    close(varERR);

    transposeA;
    multMAT(AOPT_T, BOPT, sizeA, POINTS, POINTS,
    TEMPOMAT);
```

```
multMAT(TEMPOMAT, AOPT, sizeA, POINTS, sizeA,
BOPT);

{ INITIAL LAMBDA EQUALS TRACE OF "A_T B A"
  INVERSE }
if iteration = 1 then
begin
  lambda := 0;
  for i := 1 to sizeA do
  lambda := lambda + BOPT^[i,i];
  lambda := 0.01 * lambda / sizeA
end;

{ CHECK IF "A_T B A" IS POSITIVE DEFINITE, IF NOT
  IMPROVE }
if iteration > 0 then
begin
  pd := false;
  Test_pd;
  if not pd then
  REPEAT
    for i := 1 to sizeA do
    BOPT^[i,i] := BOPT^[i,i] + lambda;
    Test_pd;
    if not pd then
      lambda := 4 * lambda
  UNTIL pd
end;

invert;

if Let.LASTexit = 17 then
  exit;

multMAT(inverse, AOPT_T, sizeA, sizeA, POINTS,
TEMPOMAT);
multVECT(TEMPOMAT, ERROR, sizeA, POINTS, deltaC);

for i := 1 to sizeA do
deltaC[i] := deltaC[i] / ( Leastp - 1 );

{ DAMP PARAMETER CHANGES WHEN THESE ARE
  OUT OF BOUND }
Damp_Single_deltaC;

{ SAVE PARAMETER CHANGES IN MATRIX INVERSE }
for m := 1 to sizeA do
inverse^[m,1] := deltaC[m];

{if CONJUGATE then
 - this is an alternative - UPDATE PARAMETER VECTOR -
begin
  i := 0;
  XRead(varEPS, 'MODIFY.');
  while not EOF(varEPS) do
  begin
    readln(varEPS, variable);
    for m := 1 to RLCGtt do
    if m = variable then
    begin
      i := i + 1;
      C[variable] := C[variable] + deltaC[i]
    end
  end;
  Close(varEPS);
  CONJUGATE := false;
  Let.M2var := 3;
  FR_plus_sensty;
end;}

{ COMPUTE CURRENT VALUE OF OBJECTIVE
  FUNCTION }
errorNOW := 0;
XRead(varEPS, 'EPSMATX.');
while not EOF(varEPS) do
begin
  readln(varEPS, x);
  errorNOW := errorNOW + x;
end;
close(varEPS);

{ SAVE OPTIMAL RESULTS IN FILES ERR_FREQ AND
  OPT_GAIN }
if iteration > 0 then
  if (abs(errorPRE - errorNOW) <= 1e-12) or
    (errorNOW < errorLEVEL) then
    begin
      Open(fVAR4, 'Err_Dist.');
      Open(fVAR5, 'Err_Freq.');
      XRead(varERR, 'Ini_Gain.');
      XRead(varEPS, 'EPSMATX.');
      while not EOF(varEPS) do
      begin
        readln(varERR, x, y); { get frequency }
        readln(varEPS, z);    { get error }
        writeln(fVAR4, x, ' ', z);{ freq vs error }
        writeln(fVAR5, x, ' ', z) { freq vs error }
      end;
      close(varERR);
      close(varEPS);
      close(fVAR4);
      close(fVAR5);

      Let.Optimum_Gain := true;
      Let.M2var := 3;
      FR_plus_sensty;
      Let.Optimum_Gain := false
    end;

{ FLETCHER'S MODIFICATION OF
  THE LEVENBERG-MARQUARDT METHOD }
if iteration > 0 then
  Use_Levenberg_Marquardt;

{if iteration > 0 then - an alternative approach -
  if errorNOW > errorPRE then
    CONJUGATE_GRADIENT_METHOD;}

{ SHOW CURRENT ERROR AND DAMPING LEVELS ON
  THE SCRREN }
window(X1+32, 13, X1+43, 13);
clrscr;
write('      ',damping);
window(X2-30, 13, X2-8, 13);
```

```
        write('    ', errorNOW:17);
        writeln(fVAR3, iteration, ' ', errorNOW);

        if iteration = 0 then { UPDATE PARAMETER VECTOR }
        begin
          i := 0;
          XRead(varEPS, 'MODIFY.');
          while not EOF(varEPS) do
          begin
            readln(varEPS, variable);
            for m := 1 to RLCGtt do
            if m = variable then
            begin
              i := i + 1;
              C[variable] := C[variable] + deltaC[i]
            end
          end;
          Close(varEPS);
        end;

        { CHECK IF CONVERGENCE HAS OCCURRED }
        if iteration > 0 then
        if abs(errorPRE - errorNOW) <= 1e-12 then
        begin
          Let.LASTexit := 18;
          Close(fVAR3);
          exit
        end;
        errorPRE := errorNOW
      UNTIL errorNOW <= errorLEVEL;
      Close(fVAR3)
    end; { of OPTIMIZATION_basic }

  begin { OPTIMIZATION_overall }
    if not Let.ill then
    begin
      Open(varEPS, 'VALUES.');
      for m := 1 to RLCGtt do
      writeln(varEPS, C[m]);
      close(varEPS)
    end
    else
    begin
      XRead(varEPS, 'VALUES.');
      for m := 1 to RLCGtt do
      readln(varEPS, C[m]);
      close(varEPS)
    end;

    OPTIMIZATION_basic;

    Open(fVAR1,'Ini_SpcG.');
    XRead(fVAR2,'Spc_Gain.');
    XRead(fVAR3,'Ini_Gain.');
    writeln(fVAR1, copy(InFile,1,length(InFile)-4));
    while not EOF(fVAR3) do
    begin
      readln(fVAR3, x, y); { x = frequency  y = initial gain }
      readln(fVAR2, x, z); { z = desired gain }
      writeln(fVAR1,x, ' ', y, ' ', z)
    end;
    Close(fVAR3);
    Close(fVAR2);
    Close(fVAR1);

    Open(fVAR1,'Opt_SpcG.');
    XRead(fVAR2,'Spc_Gain.');
    XRead(fVAR3,'Opt_Gain.');
    while not EOF(fVAR3) do
    begin
      readln(fVAR3, x, y); { x = frequency  y = optimum gain }
      readln(fVAR2, x, z); { z = desired gain }
      writeln(fVAR1,x, ' ', y, ' ', z)
    end;
    Close(fVAR3);
    Close(fVAR2);
    Close(fVAR1);

    Open(fVAR1,'FIS_Gain.');
    XRead(fVAR2,'Opt_SpcG.');
    XRead(fVAR3,'Ini_Gain.');
    while not EOF(fVAR3) do
    begin
      readln(fVAR3, x, y); { x = frequency  y = initial gain }
      readln(fVAR2, x, z, zz); { z = final, zz = desired gains }
      writeln(fVAR1,x, ' ', z, ' ', y, ' ' ,zz)
    end;
    Close(fVAR3);
    Close(fVAR2);
    Close(fVAR1);

    InputFile('.OPT');
    Open(fVAR1,InFile);
    COMMAND := ' ********************************';
    COMMAND := COMMAND +
    '********************************';
    writeln(fVAR1,COMMAND);
    writeln(fVAR1,' Optimization results ...');
    writeln(fVAR1);
    write(fVAR1,'  I    Name of input file  : ');
    writeln(fVAR1,copy(InFile, 1, length(InFile)-4));
    writeln(fVAR1,'       Number of nodes     : ',tNODES);
    writeln(fVAR1,'       Number of branches  : ',RLCGtt);
    write(fVAR1,'       ');
    writeln(fVAR1,'Total R, L, C, Gm   : ',Rtt,',',Ltt,',',Ctt,',',Gtt);
    writeln(fVAR1);
    i := RttOPT+LttOPT+CttOPT+GttOPT;
    writeln(fVAR1,' II   NUMBER OF VARIABLES :  ',i);
    writeln(fVAR1,'       LEAST p LEVEL USED  :  ',leastP);
    writeln(fVAR1,'       TOTAL SAMPLE POINTS :  ',POINTS);
    writeln(fVAR1,'       DESIRED ERROR LEVEL :
    ',errorLEVEL:17);
    writeln(fVAR1,'       ATTAINED ERROR LEVEL:
    ',errorNOW:17);
    writeln(fVAR1,'       TOTAL ITERATIONS    :  ',iteration);
    writeln(fVAR1);
    writeln(fVAR1,' III   Optimized component values: ');
    for i := 1 to RLCGtt do
    begin
      m := U^[i, 0];
      if m < 0 then
        m := 4;
      write(fVAR1,'       ');
      writeln(fVAR1,copy(compSET, 2*m - 1, 2), U^[i, 1], ' ',C[i])
```

```
  end;
  writeln(fVAR1,COMMAND);
  close(fVAR1);

  window(X1, Y1+15, X2, Y2-2);
  write('     ');
  writeln('Optimized component values: ');
  for i := 1 to RLCGtt do
  begin
    m := U^[i, 0];
    if m < 0 then
      m := 4;
    write('     ');
    TextAttr := LightGray + black;
    writeln(copy(compSET, 2*m - 1, 2), U^[i, 1], ' ',C[i]);
    initVideo;
    if i = 6 then
      window(X1+32, Y1+16, X2, Y2-2);
  end;
  windows(6);
  write('  Press any key to continue ...');
  ch := readkey;
  with Let do
  if LASTexit = 18 then
  begin
    windows(6);
    write('  Increase p to realize the desired error level');
    write(', ( Y/N ) ?');
    REPEAT
      ch := upcase(readkey)
    UNTIL ch in ['Y','N'];
    case ch of
      'Y': begin
             EOnest := false;
             ill := true;
             OPTIMIZATION_overall;
             if EOnest then
               exit;
             ill := false
           end;
      'N': ill :=false
    end
  end;
  Let.LASTexit := 1;
  windows(6);
  TextAttr := yellow + (red * 16);
  clrscr;
  GotoXY(whereX+22, whereY);
  write('Success. Press any key to continue ...');
  if not ( ch in ['Y','N'] ) then
    ch := readkey;
  initVideo;

  Let.optimality := false;
  Dispose(TEMPOMAT);
  Dispose(inverse);
  Dispose(AOPT_T);
  Dispose(BOPT);
  Dispose(AOPT);
  EOnest := true
end; { of OPTIMIZATION_overall }
```

```
procedure callANAL2;
begin
  SAVEDexitPROC := exitPROC; { ** save exitPROC pointer ** }
  exitPROC := @handleERR;   { install handleERR procedure }
  freqsOPT := false;
  nodesOPT := false;
  Analyze;
  let.ill := false;
  repeat
    Let.LASTexit := 1;
    OPTIMIZATION_overall;
    let.ill := true
  until Let.LASTexit < > 17;
  windows(1);
end;

BEGIN { program ANALOPT }
  Mark(heap);
  callANAL2;
  Release(heap)
END. { of program ANALOPT }
```

```
(* ************************************************** *)
(*  unit Anal0: ANALYSIS MODULE 0 FOR P-POND.         *)
(*      - consists of global declarations & 15 parent procedures  *)
(*      - written as part of an M. Sc. Thesis         *)
(*        by Solomon Zewde, School of Graduate Studies,  *)
(*        Electrical Engineering Dept., Addis Ababa University.  *)
(*                                                    *)
(*              March 1992 - March 1993 G. C.         *)
(* ************************************************** *)

{$N+}
unit Anal0;{$R-}

                        interface
(***************** ____________________ ******************)
uses crt; {$R-}
const
  compSET  = ' R L CGM';{ admissible components: R,L,C,VCCS}
  components = 200; { ********** maximum number per network}
  nodes    = 100; { ********** maximum number per network}
  maximumS  = 20; { highest degree of s in transfer function }
type
  STRING72  = STRING[72];
  itemUVWG  = INTEGER;
  itemB     = -1..1;
  itemCDEF  = REAL;
  arrayU    = array [0..0, 0..5] of itemUVWG;
  arrayV    = array [0..20, 0..6] of itemUVWG;
  arrayW    = array [0..21] of itemUVWG;
  arrayG    = array [0..10, 1..2] of itemUVWG;
  arrayB    = array [0..0, 0..nodes] of itemB;
  arrayC    = array [0..40] of EXTENDED;
  arrayD    = array [0..2, 0..30] of itemCDEF;
  arrayEF   = array [0..50, 0..40] of itemCDEF;

  GLOBALcontrolVARIABLES = { of the analysis phase }
    record
      fileEXISTS, { **********************input file exists }
      M1, M2,     { ..... select menu1, select menu2 }
```

```
      ct,        { .... display circuit connection information }
      list, more, { ..... list trees, generate more trees }
      ioNODES,   { ..... modify input/output nodes }
      freqs,     { read frequencies for freq. resp. evaluation }
      evals,     { .... begin evaluation of frequency response }
      CTFF,      { ..... compute transfer function first }
      FRS,       { .... compute frequency response sensitivity }
      newINPUT,  { ..... analyse another circuit }
      modify,    { ..... modify type and values of components }
      continue,  { ..... proceed with analysis }
      ill,       { ..... the matrix product ATBA is invertible }
      optimality, { ******** perform optimization of components }
      Optimum_Gain{ *** check for optimum number of iterations }
      : BOOLEAN; { return true when shown conditions are met. }

      M1var : 1..5;{controls selection of commands from MENU1 }
      M2var : 0..8;{controls selection of commands from MENU2 }
      switch: 0..2; { **** controls evaluation sequences ******}
      LASTexit: 1 .. 18; { saves exit status from program block }
    end;

  pfiREC ={of GRAPHER Unit; PFI = plot format information }
    record
      ML,MR,MT,MB,          { Margins: Left/Right/Top/Bottom }
      XNO,XLO,YNO,YLO,TIO, { x/y no./label & Title: Offsets }
      XNF,XLF,YNF,YLF,TIF, {                    ... Fonts }
      XNS,XLS,YNS,YLS,TIS, {                    ... Sizes }
      XNC,XLC,YNC,YLC,TIC, {                    ... Colors }
      AS,DS,CS,            { Axes/Divisions/Curve: LineStyles }
      AP,DP,CP,            {                    ... Patterns }
      AT,DT,CT,            {                    ... thicknesses }
      AC,DC,CC       : BYTE; {                  ... Colors }
      XL,YL,Title    : STRING72;  { x/y Labels, Graph Title }
      XRL,XRH,YRL,YRH : EXTENDED;{Ranges,Low,High }
      XFW,XFD,YFW,YFD : INTEGER;{x/y no. Width Decimal }
      XAD,YAD        : BYTE;  { x/y Axes Divisions }
      XMAX,YMAX      : INTEGER; { GetmaxX & GetmaxY }
      xa1,ya1,xa2,ya2 : EXTENDED;{ data points, xyFile }
      CN             : STRING72;  { Curve Name }
      CNF,CNS,CND,CNC : BYTE;{font,size,dir,colr:curve name}
      CNx,CNy        : REAL; { Center of Curve Name }
      CSC            : REAL; {        Curve Scale }
    end;  { of pfiREC }
var { variables of the analysis phase }
  deltaC: arrayC;  { increment on parameter vector }

  U :  ^arrayU; { stores coded network information }
  V :  arrayV;  { ..... modified incidence matrix }
  W :  arrayW;  { ..... nodes of test branches }
  G :  arrayG;  { ..... v - sending branch nodes }
  C :  arrayC;  { ..... parameter vector}
  D :  arrayD;  { . voltage gain transfer function }
  E :  ^arrayEF;{ ..... numerator sensitivities }
  F :  ^ArrayEF;{ ..... denominator sensitivities }

  SN : STRING[2]; { signs: transfer function terms }
  GO : STRING[2]; { variable to hold menu commands }

  tNODES,  { ..... total number of nodes }
  sNODE,   { ..... shorted node }
  iNODE,   { ..... input node }
  oNODE,   { ..... output node }
  Rtt,     { ..... total number of resistors }
  Ltt,     { ..... total number of inductors }
  Ctt,     { ..... total number of capacitors }
  Gtt,     { ..... number of total VCCSes }
  Gm_s,    { ... number of total Gm/s - type VCCSes }
  sGm,     { ... number of total s*Gm - type VCCSes }
  RLCGtt,  { ... number of total network components }
  RLCtt,   { .....total no. of passive components }
  maxS,    { ..... total no. of resistors }
  ROWz,    { ..... 1 + total rows of V - matrix }
  pTREES,  { ..... total number of passive trees }
  aTREES,  { ..... total number of active trees }
  tTREES,  { ..... total number of trees }
  X1,      { 1 + min value of screen x - coordinate }
  X2,      { 1 + max value of screen x - coordinate }
  Y1,      { 1 + min value of screen y - coordinate }
  Y2,      { 1 + max value of screen y - coordinate }
  POINTS,  { ..... total number of frequency points }
  leastP   { ..... exponent for the error index }
  : INTEGER;

  sgnDENOM,  { sign: transfer function denominator }
  sgnNUM,    { sign: transfer function numerator }
  desG,      { desired gain in deci-Bells }
  weight     { weight set on desired gain }
  : REAL;

  heap: POINTER;
  lambdaC: EXTENDED;
  SAVEDexitPROC: pointer;
  freqsOPT, recalculate: BOOLEAN;
  symbolic: 1 .. 3;  { 1,2: symbolic TF;  3: use keyboard. }
  InFile: STRING72;  { saves file name of current network. }
  fVAR1, fVAR2, fVAR3, fVAR4, fVAR5, fVAR6,
  varA, varB, varERR, varEPS: TEXT; { Text file variables }
  Let: GLOBALcontrolVARIABLES;

var { variables of the GRAPHER Unit }
  Command,          { currently used plotting command }
  LastCommand: STRING72; {  Last - active plotting command  }

{$F+} procedure handleERR; {$F-}

procedure getINT(inf1,inf2: STRING72; var r: INTEGER; x,y:
INTEGER);

procedure getDBL(inf1,inf2: STRING72; var r: DOUBLE; x,y:
                                        INTEGER);

procedure getUPPERcase(var s: STRING72);

procedure Open(var fileVAR: TEXT; fileNAME: STRING72);

procedure XRead(var fileVAR: TEXT; fileNAME: STRING72);

procedure InputFile( ext: STRING72 );

procedure windows(windw: INTEGER);

procedure inMODE;

procedure hiLIT(message: STRING72);
```

```
procedure keybrdMENU;

procedure initVideo;

procedure shadow(message: STRING72);

procedure indicateFILE;

procedure extractNETinfo;

                    implementation
(*********  ————————————————————  **************)

{$F+} procedure handleERR; {$F-}
{ a procedure to handle runtime errors }
var ch: CHAR;
begin
  if errorADDR < > NIL then
  begin
    windows(6);
    TextAttr := yellow + 16 * red + blink;
    write('Exiting');
    TextAttr := yellow + 16 * red;
    write(' because of ');
    write('error ',exitCODE,' at address ');
    write(seg(errorADDR^),':', ofs(errorADDR^),'. ');
    GotoXY(4, whereY);
    ch := readkey;
    errorADDR := NIL;
    exitCODE := 0
  end;
  exitPROC := savedEXITproc
end; { of handleERR }

procedure getINT(inf1,inf2: STRING72; var r: INTEGER; x,y:
INTEGER);
{ Allows error-free reading of integer values }
var
  err: INTEGER;
   s: STRING72;
begin
  window(X2 div 2 - 15, Y2 div 2 - 1, X2 div 2 + x + 2, Y2 div 2
  + y + 1);
  TextAttr := LightCyan  +  16 * LightCyan;
  ClrScr;
  window(X2 div 2 -13, Y2 div 2, X2 div 2 + x, Y2 div 2 + y);
  TextAttr := LightCyan  +  16 * Red;
  ClrScr;
  window(X2 div 2 - 12, Y2 div 2 + 1, X2 div 2 + x - 1, Y2 div 2
  + 3);
  writeln('  ',inf1);
  for err := 1 to x do
  write(chr(205));
  window(X2 div 2 - 11, Y2 div 2 + 3, X2 div 2 + x, Y2 div 2 +
  y);
  GotoXY(1,hi(WindMax));
  repeat
    TextColor(White);
    write(inf2);
    TextColor(Yellow);
    if s = #0#13 then
      readln(s)
    else
      readln(s);
    val(s, r, err);
    if err < > 0 then
      writeln('^': length(inf2) + err, ' -- Entry error '#7)
  until err = 0;
  initVIDEO;
end; { of getINT }

procedure getDBL(inf1,inf2: STRING72; var r: DOUBLE; x,y:
INTEGER);
{ Allows error-free reading of numbers with double precision }
var
  err: INTEGER;
   s: STRING72;
begin
  window(X2 div 2 - 15, Y2 div 2 - 1, X2 div 2 + x + 2, Y2 div 2 +
  y + 1);
  TextAttr := LightCyan  +  16 * LightCyan;
  ClrScr;
  window(X2 div 2 -13, Y2 div 2, X2 div 2 + x, Y2 div 2 + y);
  TextAttr := LightCyan  +  16 * Red;
  ClrScr;
  window(X2 div 2 - 12, Y2 div 2 + 1, X2 div 2 + x - 1, Y2 div 2 +
  3);
  writeln('  ',inf1);
  for err := 1 to x do
  write(chr(205));
  window(X2 div 2 - 11, Y2 div 2 + 3, X2 div 2 + x, Y2 div 2 + y);
  GotoXY(1,hi(WindMax));
  repeat
    TextColor(White);
    write(inf2);
    TextColor(Yellow);
    if s = #0#13 then
      readln(s)
    else
      readln(s);
    val(s, r, err);
    if err < > 0 then
      writeln('^': length(inf2) + err, ' -- Entry error '#7)
  until err = 0;
  initVIDEO;
end; { of getDBL }

procedure getUPPERcase(var s: STRING72);
{ Convert all chars in STRING s to their upper case equivalents }
var i: INTEGER;
begin
  for i := 1 to length(s) do
  s[i] := UpCase(s[i])
end; { of getUPPERcase }

procedure Open(var fileVAR: TEXT; fileNAME: STRING72);
begin
  assign(fileVAR,fileNAME);
  rewrite(fileVAR)
end;

procedure XRead(var fileVAR: TEXT; fileNAME: STRING72);
begin
  assign(fileVAR,fileNAME);
```

```
  reset(fileVAR)
end;

procedure InputFile( ext: STRING72 ); { creates .ext file from InFile
}
begin
  if copy(InFile, length(InFile)-3, 1) = '.' then
    InFile := copy(InFile, 1, length(InFile)-4);
  InFile := InFile + ext
end; { of InputFile }

procedure windows(windw: INTEGER);
begin
  case windw of
    1: window(X1,Y1,X2,Y2);     { ..... whole screen }
    2: window(X1,Y1,X2,Y1+4);   { .. main Keyboard inputting
       menu }
    3: window(X1,Y1+1,X2,Y1+1); { ..... input file indicator }
    4: window(X1,Y1+1,X2,Y2-2); { ..... main output window
       }
    5: window(X1,Y2-1,X2,Y2);   { MENU1, MENU2, frequency
       heading }
    6: window(X1,Y2,X2,Y2);     { I/O nodes, frequencies,
       command }
    7: window(X1,Y1+5,X2,Y1+9); { ... accepts keyboard input
       data }
    8: window(X1,Y1+9,X2,Y2-2); { ..... goes with 7 above }
    9: window(X1,Y1,X2,Y1);     { header:
       Network-Connection-Data }
   10: window(45,Y1+3,X2-5,Y2-4){window for input
       modification/s }
  end;
  clrscr
end; { of windows }

procedure initVideo; { initial text attributes, LightGray-on-blue }
begin
  TextAttr := LightGray + blue * 16
end;

procedure shadow(message: STRING72); { shows error messages }
var ii,posX: INTEGER;

  procedure paint;
  var i: INTEGER;
  begin
    for i := 1 to length(message) + 8 do
    write(' ')
  end; { paint }

begin { shadow }
  posX := -4 + (X2-length(message)) div 2;
  GotoXY(1, Y2 div 3);
  TextAttr := black + black * 16; { black on black, i.e., shadow }
  for ii:=1 to 5 do
  begin
    gotoXY(posX, whereY + 1);
    paint
  end;
  TextBackGround(white); { black on white }
  GotoXY(posX + 2, whereY - 5);
  paint;
  GotoXY(posX + 2, whereY + 1);
  paint;
  GotoXY(posX + 2, whereY + 1);
  write('   ',message,'   ');
  if copy(message,1,1) = #7 then
    write(' ');
  for ii:=1 to 2 do
  begin
    GotoXY(posX + 2, whereY + 1);
    paint
  end;
  initVideo
end; { of shadow }

procedure indicateFILE;
begin
  windows(1);
  if symbolic = 3 then
    InputFile('.LOD');
  symbolic := 3;
  GotoXY(whereX + ((X2 - 36) div 2 ), whereY);
  TextAttr := black + brown * 16; { black on brown }
  write(' Name of current network: ');
  TextColor(yellow); { yellow on brown }
  write(' ',copy(InFile, 1, length(InFile) - 4), '. ');
  shadow('Select from menu below.')
end;

procedure inMODE; { selects b/n keyboard & file input modes }
var
  ch,chh: CHAR;
  i: INTEGER;

  procedure getCH; { read key and store as ch }
  begin
    ch := readkey;
    if ch = #13 then
      ch := readkey;
    writeln(ch);
    ch := UpCase(ch);
  end;

  procedure ifk; { slected if input mode is keyboard }
  begin
    symbolic := 3;
    exit
  end;

  procedure ifef; { selected if input mode is file }
  var mistakes: WORD;
  begin
    i := 0;
    Repeat
      if i = 0 then
      begin
        write('  name of input file ... ');
        Readln(InFile);
        getUPPERcase(InFile);
        if Infile='' then
          readln(InFile);
        InputFile('.LOD')
      end
```

```
        else
        repeat
          write(#7'File not found. Create a new one, Y/N? ');
          getCH;
          case ch of
            'Y' : begin
                    Open(fVAR1, InFile);
                    close(fVAR1);
                    Let.fileEXISTS := False;
                    Exit
                  end;
            'N' : begin
                    chh := ch;
                    repeat
                      write('Try retyping, Y/N? ');
                      getCH;
                      case ch of
                        'Y' : begin
                                write('Retype file name ... ');
                                Readln(InFile);
                                InputFile('.LOD')
                              end;
                        'N' : begin
                                Let.fileEXISTS := False;
                                symbolic := 3;
                                Exit
                              end
                      end
                    until ch in ['Y','N'];
                    ch := chh
                  end
          end;
          i := i + 1
        until ch in['Y', 'N'];
        Assign(fVAR1, InFile);
        {$I-}
          Reset(fVAR1);
          mistakes := IOresult
        {$I+};
        If mistakes = 0 then
          Let.fileEXISTS := true;
        i := i + 1
      Until mistakes = 0;
      Close(fVAR1);
      indicateFILE
    end; { of ifef }

  begin { inMODE }
    windows(4);
    windows(5);
    Let.fileEXISTS := False;
    i := 0;
    repeat
      if i > 0 then
        write(#7'Input mode, file/keyboard? ')
      else
        write('Input mode, file/keyboard? ');
      HighVideo;
      GotoXY(13, whereY);
      write('f');
      GotoXY(18, whereY);
      write('k');
      GotoXY(28, whereY);
      LowVideo;
      getCH;
      i := i + 1
    until ch in['F','K'];
    case ch of
      'F': ifef;
      'K': ifk
    end
  end; { of inMODE }

  procedure hiLIT(message: STRING72); { highlights the message }
  begin
    HighVideo;
    write(message);
    lowVideo;
  end; { of hiLIT }

  procedure keybrdMENU; { selected if input file is not available }
  var i: INTEGER;
  begin
    windows(2);
    GotoXY(10, 1);
    for i := 1 to 55 do
    HiLit(chr(196));
    GotoXY(25, 1);
    HiLit(' Network-Connection-Data ');
    GotoXY(2, 2);
    HiLit('PASSIVE INPUTS:');
    write(' component number, +  - nodes, value;');
    GotoXY(2, 3);
    HiLit(' ACTIVE INPUTS:');
    write(' vccs number, sending and receiving nodes, ');
    write('vccs type, value;');
    GotoXY(18, 4);
    write('* for vccs type  1=GM, 2=GM/S, 3=SGM.');
  end; { of keybrdMENU }

  procedure extractNETinfo; { extract network information }
  var Rttm,Lttm,Cttm,Gttm,ww,xx,yy,zz: INTEGER;
  begin
    new(E);
    new(F);
    with Let do
    if not modify then
      inMODE;
    with Let do
    if not FileExists then { use keyboard for input }
    begin
      if not modify then
      begin
        keybrdMENU;
        windows(7);
        GotoXY(25, 1);
        write('New file name ?  ');
        readln(InFile);
        ww := 0;
        repeat
          if length(InFile) = 0 then
          begin
            GotoXY(42, whereY - 1);
            if not odd(ww) then
```

```
        write('Bad file name. Retype. ')
      else
      begin
        GotoXY(42, whereY);
        write('                    ');
        GotoXY(42, whereY);
        write('Retype. ')
      end;
      readln(InFile);
      ww := ww + 1
    end
  until length(InFile) > 0;
  InputFile('.LOD');
  Open(fVAR1, InFile);
  GotoXY(23, whereY);
  write('number of nodes =  ');
  readln(tNODES);
  GotoXY(15, whereY);
  write('number of R, L, C, VCCS =  ');
  readln(Rtt, Ltt, Ctt, Gtt);
  writeln(fVAR1,tNODES,' ',Rtt,' ',Ltt,' ',Ctt,' ',Gtt);
  Rttm := Rtt;
  Lttm := Ltt;
  Cttm := Ctt;
  Gttm := Gtt;
  {FILL BRANCH DATA MATRIX: no element modification}
  RLCGtt := Rtt + Ltt + Ctt + Gtt;
  RLCtt := RLCGtt - Gtt;

  GetMem(U, (RLCGtt + 1) * 6 * sizeof(itemUVWG));
  for yy := 0 to RLCGtt do
  for zz := 0 to 5 do
  U^[yy, zz] := 0;
  for yy := 0 to RLCGtt do
  C[yy] := 0;
  GotoXY(10, whereY);
  for ww := 1 to 55 do
  HiLit(chr(196));
  writeln;
  windows(8)
end;
if modify then  { modify number & nature of elements }
begin
  windows(6);
  write(' number of modified R, L, C, VCCS =  ');
  readln(Rttm, Lttm, Cttm, Gttm);
  switch := 0;
  InputFile('.LOD');
  Open(fVAR1, InFile);
  writeln(fVAR1,tNODES,' ',Rtt,' ',Ltt,' ',Ctt,' ',Gtt);
  window(43,Y1+2,X2-3,Y2-3);
  Textattr := red + red * 16;
  clrscr;
  windows(10);
  Textattr := black + LightGray * 16;
  clrscr;
  writeln;
  write('*** COMPONENT MODIFICATIONS ***');
  for yy := 1 to X2 - 49 do
  write(chr(205))
end;
for yy := 1 to Rttm do
begin
  write('    R');
  readln(xx, U^[xx, 2], U^[xx, 3], C[xx]);
  U^[xx, 0] := 1;
  U^[xx, 1] := xx
end;
for yy := 1 to Lttm DO
begin
  write('    L');
  readln(xx, U^[xx + Rtt, 2], U^[xx + Rtt, 3], C[xx + Rtt]);
  U^[xx + Rtt, 0] := 2;
  U^[xx + Rtt, 1] := xx
end;
zz := Rtt + Ltt;
for yy := 1 to Cttm DO
begin
  write('    C');
  readln(xx, U^[zz + xx, 2], U^[zz + xx, 3], C[zz + xx]);
  U^[zz + xx, 0] := 3;
  U^[zz + xx, 1] := xx
end;
Gm_s := 0;
sGm := 0;
zz := zz + Ctt;
for yy := 1 TO Gttm DO
begin
  write('   GM');
  read(xx, G[xx,1], G[xx,2],
    U^[zz + xx,3], U^[zz + xx, 2]);
  readln(U^[zz + xx, 0], C[zz + xx]);
  U^[zz + xx, 0] := -U^[zz + xx, 0];
  U^[zz + xx, 1] := xx;
  case abs(U^[zz + xx, 0]) of
    2: Gm_s := Gm_s + 1;
    3: sGm := sGm + 1
  end
end;
for yy := 1 to RLCGtt do
begin
  for zz := 0 to 4 do
  begin
    if zz < 4 then
      write(fVAR1, U^[yy, zz]:5);
    if zz = 4 then
      write(fVAR1, '   ', C[yy]:14);
    if (zz = 4) and (yy in[RLCtt+1 .. RLCGtt]) then
      write(fVAR1, G[yy-RLCtt, 1]:5, G[yy-RLCtt, 2]:5)
  end;
  writeln(fVAR1)
end;
initVideo;
close(fVAR1)
end;

Let.M2var := 0;
POINTS := 0;
with Let do
if not modify then
begin
  if FileExists then { read it!! }
  begin
    XRead(fVAR1, InFile);
```

```
      readln(fVAR1, tNODES, Rtt, Ltt, Ctt, Gtt);
      RLCGtt := Rtt + Ltt + Ctt + Gtt;
      GetMem(U, (RLCGtt+1) * 6 * sizeof(itemUVWG));
      RLCtt := RLCGtt - Gtt;
      for yy := 1 to RLCGtt do
      begin
        for zz := 0 to 3 do
        read(fVAR1, U^[yy, zz]);
        read(fVAR1, C[yy]);
        if (zz = 3) and (yy in[RLCtt+1 .. RLCGtt]) then
          read(fVAR1, G[yy-RLCtt, 1], G[yy-RLCtt, 2]);
        readln(fVAR1)
      end;
      for zz := 0 to 6 do
      U^[0, zz] := 0;
      for yy := 1 to RLCGtt do
      for zz := 4 to 5 do
      U^[yy, zz] := 0;
      Gm_s := 0;
      sGm := 0;
      Gttm := Gtt;
      for yy := 1 to Gttm DO
      begin
        case abs(U^[yy + RLCtt, 0]) of
          2: Gm_s := Gm_s + 1;
          3: sGm := sGm + 1
        end
      end;
      close(fVAR1)
    end
  end;
  Let.FileExists := False
end; { of extractNETinfo }

END. { of Anal0 }
```

```
(* ********************************************************)
(*  unit ANAL1: ANALYSIS MODULE 1 FOR P-POND.             *)
(*      - contains 16 procedures and a function used by ANALOPT*)
(*      - written as part of an M. Sc. Thesis             *)
(*          by Solomon Zewde, School of Graduate Studies, *)
(*          Electrical Engineering Dept., Addis Ababa University.*)
(*                                                        *)
(*                March 1992 - March 1993 G. C.           *)
(* ********************************************************)

{$N+}
unit ANAL1;{$R-}

                    interface
(************  ____________________  ******************)

uses crt, Anal0;

var count,NODE: INTEGER;

procedure getFREQS;

function raise(number: EXTENDED; power: INTEGER):
EXTENDED;

procedure common(jj: INTEGER; var kk: INTEGER; LL: STRING);

procedure MENU1;

procedure MENU2;

procedure passiveTEST;

procedure activeTEST;

procedure TREES;

procedure TF_plus_sensty;

procedure FR_plus_sensty;

procedure initials;

procedure set_UNITY;

procedure initDEF;

procedure initM2VARS;

procedure filterM2VARS;

procedure check_TF_error;

procedure check_FREQ_RESP_Error;
```

```
                    implementation
(*********  ____________________________  ************)
var
  initialF, (***  initial frequency in Hertz  **************)
  f_incr,   (***  frequency increment or step size **********)
  f_scale,  (***  frequency scale factor  ********************)
  frequency (***  point to evaluate frequency response *****)
  : DOUBLE;

procedure getFREQS;
{   reads frequencies either from file or keyboard and
consisits of two nested procedures, readMAG and ACCEPT,
which perform, respectively, the reading of frequencies
from keyboard and file. }
var
   i, mistakes: INTEGER;
   transfer_points, transfer_gains: REAL;

procedure readMAG(freqPRMPT: STRING; var magnitude:
                                        DOUBLE);
{ nested procedure to read frequencies from keyboard }
begin
  getDBL('FREQUENCY INFORMATION', freqPRMPT,
    magnitude, 30,7)
end; { of readMAG }

procedure ACCEPT(vv: INTEGER; ww: DOUBLE);
{ nested procedure to read frequencies from file }
begin
  window(29,Y2-vv,X2-31,Y2-vv);
```

```
    clrscr;
    if not Let.ill then
      if not freqsOPT then
        readln(ww);
    write(ww:12)
  end;

begin { getFREQS }
  if not Let.optimality then { read ferquencies }
  begin
    windows(5);
    write('Meanings of frequency domain variables:          ');
    windows(6);
    write(' initial = INITIAL FREQUENCY. ');
    ReadMag(' initial ? ', initialF);
    windows(6);
    write(' increment = FREQUENCY INCREMENT OR STEP. ');
    ReadMag(' increment ? ', f_incr);
    windows(6);
    write(' scale = FREQUENCY SCALE, USUALLY LINEAR.
    ');
    ReadMag(' scale ? ', f_scale);
    windows(6);
    write(' samples = TOTAL NUMBER OF FREQUENCIES OR
    SAMPLE POINTS.');
    getINT('FREQUENCY INFORMATION','samples ?
    ',POINTS,30,7);
    windows(5);
    freqsOPT := true
  end
  else { read frequencies if not supplied before }
  begin
    ACCEPT(18, initialF);
    ACCEPT(16, f_incr);
    ACCEPT(14, f_scale);
    window(37,Y2-12,X2-31,Y2-12);
      clrscr;
        if not Let.ill then
          if not freqsOPT then
            readln(POINTS);
              write(POINTS:4);
    if copy(InFile, length(InFile)-3, 1) = '.' then
      InFile := copy(InFile, 1, length(InFile)-4);
    InFile := InFile + '.SPC'; { use spec file if it exists }
    Assign(fVAR1, InFile);
    {$I-}
      Reset(fVAR1);
      mistakes := IOresult
    {$I+};
    { NOTE: A "Close(fVAR1)" at this point generates error. }
    if mistakes = 0 then { remark that a file has been used }
    begin
      window(22, Y2-10, X2-35, Y2-10);
      clrscr;
      write(' As per ', InFile);
      window(22,Y2-8, X2-35, Y2-6);
      write(' As per ', InFile)
    end
    else { read specifications from keyboard }
    begin
      Open(fVAR2,InFile);
      for i:=1 to POINTS do
      begin
        window(22, Y2-10, X2-35, Y2-10);
        clrscr;
        write(' dB Gain',i,' = ');
        window(22,Y2-8, X2-35, Y2-8);
        clrscr;
        write(' weight',i,' = ');

        window(22, Y2-10, X2-35, Y2-10);
        GotoXY(14,1);
        Read(desG);
        write(' dB Gain',i,': ',desG:0:3);

        window(22,Y2-8, X2-35, Y2-8);
        GotoXY(14,1);
        Read(weight);
        write(' weight',i,': ',weight:0:3);

        windows(6);
        write('  NOTES: Gain',i,' = ',desG:0:3,';   ',
          'weight',i,' = ',weight:0:3,'.');
        writeln(fVAR2,desG,weight)
      end;
      close(fVAR2);
      window(22, Y2-10, X2-35, Y2-10);
      clrscr;
      write(' As per ', InFile);
      window(22,Y2-8, X2-35, Y2-6);
      write(' As per ', InFile)
    end
  end;
  if copy(InFile,length(InFile)-2,length(InFile)) = 'SPC' then
  begin
    i := 0;
    transfer_points := initialF;
    assign(fVAR1,'SPC_GAIN.');
    rewrite(fVAR1);
    assign(fVAR2,InFile);
    reset(fVAR2);
    while not EOF(fVAR2) do
    begin
      if i > 0 then
        transfer_points := (transfer_points + f_incr) * f_scale;
      i := i + 1;
      Readln(fVAR2,transfer_gains);
      writeln(fVAR1, transfer_points, ' ', transfer_gains)
    end;
    close(fVAR2);
    close(fVAR1)
  end
end; { of getFREQS }

function raise(number: EXTENDED; power: INTEGER): EXTENDED;
{  This function performs the operation of exponentiation as
raise = number ^ power. Pascal does not support the function. }

var result: EXTENDED;   i: INTEGER;
begin
  result := 1;
  for i := 1 to power do
    result := number * result;
  raise := result
```

```
end; { of raise }

procedure common(jj: INTEGER; var kk: INTEGER; LL: STRING);
{This procedure is common to both MENU1 and MENU2 listed
next.}
var i,j: INTEGER;
begin
  windows(6);
  initVideo;
  LowVideo;
  writeln;
  write('SELECTION = ');
  for i := 1 to 100 do
  begin
    GO := Upcase(readkey);
    if GO = chr(27) then
      halt;
    GO := GO + Upcase(readkey);
    if (GO < > 'EL') and (GO < > 'NO') then
      windows(4);
    windows(6);
    write('COMMAND:');
    HighVideo;
    write(' ', GO);
    LowVideo;
    for j := 1 to jj do
    begin
      kk := j;
      if ( GO = copy(LL, 2*kk-1, 2) ) then
      begin
        j := jj;
        i := 100;
      end;
    end;
    if ( GO < > copy(LL, 2*kk-1, 2) ) then
    begin
      windows(4);
      shadow(#7'Bad command: reselect from menu below. ');
      window(12,Y2,X2-12,Y2);
      REPEAT
      UNTIL KEYPRESSED;
      jj := jj+1
    end
  end
end; { of common }

procedure MENU1; { first of the two analysis menus }
const commandSET = 'TRTFSE//CI';
begin
  windows(5);
  TextAttr := blue + LightGray * 16; { blue on LightGray }
  write('  MENU1  ');
  initVideo; { LightGray on blue }
  write('  CIRCUIT  TREES  TF=TRANSFER FUNCTION');
  write('  SENSITIVITIES  ');
  HighVideo;
  TextBackGround(red); { LightGray on red on blue }
  gotoxy(15, Y1);
  write('CI');
  gotoxy(25, Y1);
  write('TR');
  gotoxy(33, Y1);
  write('TF');
  gotoxy(55, Y1);
  write('SE');
  TextBackGround(Green); { LightGray on green on blue }
  gotoxy(70, Y1);
  write(' // = NEXT  ');
  gotoxy(80, Y1);
  count:=Let.M1var;
  common(5, count, commandSET);
  Let.M1var := count
end; { of MENU1 }

procedure MENU2; { second of the two analysis menus }
const commandSET = 'OPREDOELNO\\FRLO';
begin
  windows(5);
  write('  LOAD  FREQUENCIES  RESPONSE');
  write('  ELEMENTS  NODES  DO  OPTIMIZE');
  TextBackGround(Green);
  gotoxy(1, Y1);
  HighVideo;
  write(' \\ = MENU1 ');
  TextBackGround(Red);
  gotoxy(16, Y1);
  write('LO');
  gotoxy(23, Y1);
  write('FR');
  gotoxy(37, Y1);
  write('RE');
  gotoxy(48, Y1);
  write('EL');
  gotoxy(59, Y1);
  write('NO');
  gotoxy(67, Y1);
  write('DO');
  gotoxy(72, Y1);
  write('OP');
  gotoxy(77, Y1);
  count:=Let.M2var;
  common(8, count, commandSET);
  Let.M2var := count
end; { of MENU2 }

procedure passiveTEST; { performs passive tree test. }
var plusNODE,minusNODE,pNODE,mNODE,xCODE:INTEGER;

  procedure CONNECT;
    procedure checkLOOP;
      procedure getNODES;
      begin
        xCODE := 1;
        plusNODE := U^[V[V[0, NODE], NODE], 2];
        minusNODE := U^[V[V[0, NODE], NODE], 3];
        if plusNODE = sNODE then
          plusNODE := 0;
        if minusNODE = sNODE then
          minusNODE := 0;
        for count := 1 to pNODE do
        begin
          mNODE := minusNODE;
          if W[count] = plusNODE then
            xCODE := 3;
```

```
        if W[count] = plusNODE then
          exit;
        mNODE := plusNODE;
        if W[count] = minusNODE then
          xCODE := 3;
        if W[count] = minusNODE then
          exit
      end
    end; { of getNODES }

  begin { checkLOOP }
    pNODE := pNODE+1;
    for NODE := 1 to tNODES do
    begin
      if NODE < > sNODE then
      begin
        if V[ROWz, NODE] < > 1 then
        begin
          getNODES;
          if xCODE = 3 then { else skip next block }
          begin
            for count := 1 to pNODE do
            if W[count] = mNODE then
              exit;
            W[pNODE+1] := mNODE;
            V[ROWz, NODE] := 1;
            checkLOOP;
          end
        end
      end
    end
  end; { of checkLOOP }

begin { CONNECT }
  repeat
    xCODE := 1;
    for NODE := 1 to tNODES do
    begin
      V[ROWz, NODE] := 0;
      W[1+NODE] := 0;
    end;
    pNODE := 1;
    if V[0, 0] < > 1 then
      if V[0, 0] = 2 then
        xCODE := 4;
    if V[0, 0] = 2 then
      exit;
    if sNODE = 1 then
      pNODE := 2;
    V[ROWz, pNODE] := 1;
    W[1] := U^[V[V[0, pNODE], pNODE], 2];
    if W[1] = sNODE then
      W[1] := 0;
    W[2] := U^[V[V[0, pNODE], pNODE], 3];
    if W[2] = sNODE then
      W[2] := 0;
    pNODE := 1;
    checkLOOP;
    mNODE := 1;
    if sNODE = 0 then
      mNODE := 0;
    if pNODE = tNODES + 1 - mNODE then
      V[0, 0] := 2;
    if sNODE = 0 then
      xCODE := 5;
    if sNODE = 0 then
      exit;
    { COMMON TREE TEST - sNODE AND - oNODE }
    if V[0, 0] = 1 then
      xCODE := 6;
    if V[0, 0] = 1 then
      exit
  until V[0, 0] < > 2
end; { of CONNECT }

procedure CHECKcommonTREE;
begin
  xCODE := 1;
  for NODE := 1 to V[ROWz-1, sNODE] do
  begin
    mNODE := U^[V[NODE, sNODE], 2];
    if mNODE = sNODE then
      mNODE := U^[V[NODE, sNODE], 3];
    if V[V[0, mNODE], mNODE] = V[NODE, sNODE] then
    begin
      if mNODE = oNODE then
        xCODE := 7;
      if mNODE = oNODE then
        exit;
      V[ROWz, mNODE] := 1;
      W[pNODE] := mNODE;
      pNODE := pNODE+1
    end
  end
end; { of CHECKcommonTREE }

procedure continue;
var k: INTEGER;
begin
  xCODE := 1;
  for k := 1 to pNODE-1 do
  begin
    mNODE := U^[V[V[0, NODE], NODE], 3];
    if W[k] = U^[V[V[0, NODE], NODE], 2] then
    begin
      xCODE := 8;
      exit;
    end;
    mNODE := U^[V[V[0, NODE], NODE], 2];
    if W[k] = U^[V[V[0, NODE], NODE], 3] then
    begin
      xCODE := 8;
      exit
    end
  end
end; { of continue }

procedure CHECKendROW;
begin
  xCODE := 1;
  if pNODE = 1 then
    xCODE := 9;
  if pNODE = 1 then
    exit;
```

```
      for NODE := 1 to tNODES do
      begin
        if NODE < > sNODE then
        begin
          if V[ROWz, NODE] < > 1 then
          begin
            continue;
            if xCODE = 8 then
            begin
              xCODE := 1;
              if mNODE = oNODE then
                xCODE := 10;
              if mNODE = oNODE then
                exit;
              W[pNODE] := mNODE;
              pNODE := pNODE+1;
              V[ROWz, NODE] := 1;
              CHECKendROW
            end
          end
        end
      end
    end; { of CHECKendROW }

  begin { passiveTEST }
    xCODE := 1;
    pTREES := pTREES + 1;
    CONNECT;
    if xCODE in [5, 6] then
      exit;
    xCODE := 1;
    CHECKcommonTREE;
    if xCODE < > 7 then
    begin
      xCODE := 1;
      CHECKendROW;
      if xCODE = 9 then
        exit;
      if xCODE < > 10 then
      begin
        xCODE := 1;
        exit
      end
    end;
    V[0, 0] := 3;
    sgnNUM := trunc(raise(-1, iNODE+oNODE))
  end; { of passiveTEST }

  procedure activeTEST; { performs active tree test }
  var
    viGRAF,NODE1,NODE2,tempo,jj,xCODE:INTEGER;
    sign: REAL;
    B: ^ArrayB;

    procedure determinant;
    begin
      if viGRAF >= 3 then { viGRAF = 1, i - receiving branches }
      begin             {       = 2, v - sending branches   }
        sNODE := oNODE; {       = 3, determinant evaluation }
        for NODE := 1 to tNODES do
        B^[iNODE, NODE] := B^[oNODE, NODE];
        sign := 1;
        if ABS(oNODE-iNODE) > 1 then
          sign := trunc(raise(-1, abs(oNODE-iNODE)+1));
        exit
      end;
      sign := 1
    end; { of determinant }

    procedure doTEST;
    var
      nextSUBST,detEVAL,xBLOCK: BOOLEAN;
      i,j: INTEGER;

      procedure deleteN1orN2;
      var
        xxBLOCK: BOOLEAN;
        ii: INTEGER;
      begin
        xCODE := 1;
        jj := jj-1;
        if jj < > sNODE then
        begin
          if B^[jj, jj] = 0 then
          begin
            if jj = 1 then
              xCODE := 4;
            if jj = 1 then
              exit;
            NODE := jj;
            repeat
              NODE := NODE-1;
              if NODE < > sNODE then
              if B^[jj, NODE] < > 0 then
                xCODE := 7
            until (NODE < = 1) or (xCODE = 7);
            if xCODE < > 7 then
            begin
              xCODE := 5;
              exit
            end;
            for ii := 1 to jj do
            begin
              tempo := B^[ii, jj];
              B^[ii, jj] := B^[ii, NODE];
              B^[ii, NODE] := tempo
            end;
            sign := -sign
          end;
          if jj < > 1 then
          begin
            ii := jj;
            repeat
              ii := ii-1;
              if ii < > sNODE then
              begin
                xxBLOCK := false;
                if B^[ii, jj] < > 0 then
                begin
                  if B^[ii, jj] < > B^[jj, jj] then
                  begin
                    for NODE := 1 to jj-1 do
                    B^[ii,NODE]:=B^[ii,NODE]+B^[jj,NODE];
                    xxBLOCK := true
```

```
                end;
                if not xxBLOCK then
                  for NODE := 1 to jj-1 do
                  B^[ii,NODE]:=B^[ii,NODE]-B^[jj,NODE]
              end
            end
          until ii <= 1
        end;
        if B^[jj, jj] < 0 then
          sign := -sign
      end;
      if jj > 1 then
        deleteN1orN2
    end; { of deleteN1orN2 }

begin { doTEST }
  GetMem(B, tNODES * tNODES * sizeof(itemB));
  repeat
    for i := 1 to tNODES do
    for j := 1 to tNODES do
    B^[i, j] := 0;
    for NODE := 1 to tNODES do
    begin
      if V[0, NODE] <> 0 then
      begin
        xBLOCK := false;
        if viGRAF = 1 then
        begin
          NODE1 := U^[V[V[0, NODE], NODE], 2];
          NODE2 := U^[V[V[0, NODE], NODE], 3];
          xBLOCK := true
        end;
        if not xBLOCK then
        begin
          if viGRAF in [2, 3] then
          begin
            if V[V[0, NODE], NODE] <= RLCtt then
            begin
              NODE1 := U^[V[V[0, NODE], NODE], 2];
              NODE2 := U^[V[V[0, NODE], NODE], 3];
              xBLOCK := true
            end;
            if not xBLOCK then
            begin
              NODE1 := G[U^[V[V[0,NODE],NODE],1],1];
              NODE2 := G[U^[V[V[0,NODE],NODE],1],2]
            end
          end
        end;
        B^[NODE, NODE1] := 1;
        B^[NODE, NODE2] := -1
      end;
    end;
    determinant;
    jj := tNODES + 1;
    deleteN1orN2;
    nextSUBST := false;
    if not (xCODE in [4..5]) then
    begin
      case viGRAF of
        1: begin
             sgnDENOM := sign;
             sgnNUM := sign;
             viGRAF := 2;
             nextSUBST := true
           end;
        2: begin
             sgnDENOM := sgnDENOM * sign;
             V[0, 0] := 4;
             viGRAF := 3;
             nextSUBST := true
           end;
        3: sgnNUM := sgnNUM * sign
      end;
      if not nextSUBST then
      begin
        case V[0, 0] of
          1: V[0, 0] := 5;
          4: V[0, 0] := 6
        end
      end;
      if not nextSUBST then
      begin
        case viGRAF of
          1: begin
               xCODE := 6;
               exit
             end;
          2: begin
               viGRAF := 3;
               nextSUBST := true
             end;
          3: sNODE := iNODE
        end
      end
    until not nextSUBST;
    FreeMem(B, tNODES * tNODES * sizeof(itemB))
  end; { of doTEST }

begin { activeTEST }
  xCODE := 1;
  aTREES := aTREES + 1;
  pTREES := pTREES - 1;
  viGRAF := 1;
  doTEST;
  if xCODE = 6 then
    exit
end; { of activeTEST }

procedure TREES;
var cc,dd: INTEGER;

  procedure getTREE; { cyclic tree generation algorithm }
  var
    aTEST,pTEST,hiNODE: BOOLEAN;
    xCODE,pN,mN: BYTE;

    procedure networkTYPE;
    begin
      repeat
        NODE := NODE + 1;
        aTEST := false;
        if V[V[0, NODE], NODE] > RLCtt then
```

```
        aTEST := true;
      if V[V[0, NODE], NODE] > RLCtt then
        exit;
    until NODE = tNODES
end; { of networkTYPE }

procedure step_DOWN;
begin
  if not hiNODE then
  repeat
    xCODE := 1;
    pN := pN - 1;
    if pN = 0 then
      xCODE := 4;
    if pN = 0 then
      exit;
  until pN < > sNODE;
  hiNODE := false;
  if V[0, pN] < V[ROWz-1, pN] then
    exit;
  V[0, pN] := 1;
  step_DOWN
end; { of step_DOWN }

procedure cyclicMEAT;
begin
  hiNODE := false;
  repeat
    step_DOWN;
    if xCODE = 4 then
      exit;
    V[0, pN] := V[0, pN] + 1;
    if pN = U^[V[V[0, pN], pN], 2] then
      mN := U^[V[V[0, pN], pN], 3];
    if pN = U^[V[V[0, pN], pN], 3] then
      mN := U^[V[V[0, pN], pN], 2];
    if mN < > 0 then
    begin
      if mN < > sNODE then
      begin
        if V[V[0, pN], pN] < >
          U^[V[V[0, mN], mN], 4] then
            if V[V[0, pN], pN] < >
              V[V[0, mN], mN] then
                exit;
        if mN > pN then
          pN := mN;
        hiNODE := true;
      end
    end;
  until not hiNODE
end; { of cyclicMEAT }

procedure Cyclic;
begin
  if V[0, 0] < > 0 then
  begin
    V[0, 0] := 1;
    pN := tNODES + 1;
    cyclicMEAT;
    if xCODE = 4 then
      exit
    end
  end; { of Cyclic }

begin { getTREE }
  xCODE := 1;
  repeat
    pTEST := false;
    Cyclic;
    if xCODE = 4 then
      exit;
    { TEST FOR A POSSIBLE TREE }
    V[0, 0] := 1;
    sgnDENOM := 1;
    sgnNUM := 1;
    passiveTEST;
    if V[0, 0] = 1 then
      pTEST := true;
    if not pTEST then
    begin
      if Gtt < > 0 then
      begin
        NODE := 0;
        networkTYPE;
        if aTEST then
        begin
          V[0, 0] := 1;
          activeTEST;
          if V[0, 0] = 1 then
            pTEST := true
        end
      end
    end
  until not pTEST;
  tTREES := tTREES + 1
end; { of getTREE }

procedure treeSTATUS;
begin
  with Let do
  case V[0, 0] of
    1: LASTexit := 9;
    2: LASTexit := 10;
    3: LASTexit := 11;
    4: LASTexit := 12;
  5,6: LASTexit := 13
  end;
  exit
end; { of treeSTATUS }

procedure doMORE;
begin
  if Let.LASTexit in [11, 10] then
  begin
    if not Let.optimality then
    begin
      write(' D');
      write(fVAR1, ' D');
      if cc = 0 then
        write(' ');
      if cc = 0 then
        write(fVAR1, ' ');
      cc := cc - 17;
```

```
      end;
      exit
    end;
    if Let.LASTexit in [11, 10, 13] then
    begin
      if sgnNUM*raise(-1, iNODE + oNODE) < 1 then
        SN[1] := '-';
      if not Let.optimality then
      begin
        write(SN[1], 'N');
        write(fVAR1, SN[1], 'N');
      end;
      if V[0, 0] = 5 then
        exit
    end;
    if Let.LASTexit in [11, 10, 13, 12] then
    begin
      if sgnDENOM < 1 then
        SN[2] := '-';
      if not Let.optimality then
      begin
        write(SN[2], 'D');
        write(fVAR1, SN[2], 'D')
      end
    end;
    Let.LASTexit := 1
  end; { of doMORE }

begin { TREES }
  cc := 0;
  repeat
    if cc = 100 then
      writeln;
    Let.more := false;
    getTREE;
    if Let.M1var in [2, 3] then
      Let.LASTexit := 14;
    if Let.M1var in [2, 3] then
      exit;
    if Let.M1var = 1 then
      SN := '++';
    cc := 0;
    treeSTATUS;
    if Let.LASTexit = 9 then
      exit;
    if not Let.optimality then
    if Let.LASTexit = 11 then
    begin
      write(' N');
      cc := 17;
      write(fVAR1, ' N')
    end;
    doMORE;
    {if not Let.optimality then
    begin}
      write(' ');
      write(fVAR1, ' ');
      GotoXY(5, whereY);
      for dd := 1 to tNODES do
      if V[0, dd] <> 0 then
      begin
        write(V[V[0, dd], dd]:4);
        write(fVAR1, V[V[0, dd], dd]:5)
      end;
      writeln(fVAR1);
      cc := 100;
    {end}
  until cc > cc
end; { of TREES }

procedure TF_plus_sensty; { transfer function + sensitivity }
const impedanceSET  = 'GSGBG0  RLC';
var
  strP,NO,cnvrt: STRING;
  i,delS: INTEGER;
  realP: EXTENDED;
begin  { TF_plus_sensty }
  realP := 1;
  strP := '';
  for i := 1 to Rtt+Ltt do
    U^[i,5] := 1;
  for i := Rtt+Ltt+1 to RLCGtt do
    U^[i,5] := 0;
  delS := Ltt+Gm_s;
  for i := 1 to tNODES do
  U^[V[ V[0,i],i ],5] := 1 - U^[V[ V[0,i],i ], 5];
  for i := 1 to RLCGtt do
  begin
    if U^[i, 5] <> 0 then
    begin
      realP := realP*C[i];
      str(U^[i, 1], NO);
      if U^[i, 0] > 0 then
        strP := strP + copy(impedanceSET, U^[i, 0]+8, 1) + NO;
      if U^[i, 0] < 0 then
        strP := strP + copy(impedanceSET, U^[i,0]*2+7,2) + NO;
      if abs(U^[i, 0]) = 3 then
        delS := delS + 1;
      if U^[i, 0] = -2 then
        delS := delS-1
    end;
    if U^[i, 5] = 0 then
      if U^[i, 0] = 2 then
        delS := delS-1
  end;
  if V[0, 0] <> 5 then
    D[2, delS] := D[2, delS] + sgnDENOM*realP;
  if V[0, 0]-1 in [2, 4, 5] then
    D[1, delS] := D[1, delS] +
      sgnNUM*realP*raise(-1, iNODE + oNODE);
  str(sgnDENOM:0:0, cnvrt);
  if cnvrt = '1' then
    cnvrt := '+'
  else
    cnvrt := '-';

  if not Let.optimality then
  begin
    if V[0, 0] <> 5 then
      writeln(fVAR2, 'denom ', delS, ' ', cnvrt + strP);
    if V[0, 0]-1 in [2, 4, 5] then
      writeln(fVAR2, 'numer ', delS, ' ', cnvrt + strP);
  end;
  if V[0, 0] - 1 in [1, 3] then
```

```
    if Let.M1var = 2 then
      exit;

  if Let.M1var < > 2 then
  for i := 1 to RLCGtt do { TF sensitivity: the direct method }
  begin
    if U^[i, 5] < > 0 then
    begin
      if V[0, 0] < > 5 then
        F^[i, delS] := F^[i, delS] + sgnDENOM*realP/C[i];
      if V[0, 0]-1 in [2, 4, 5] then
        E^[i, delS] := E^[i, delS] + sgnNUM*realP*
          raise(-1, iNODE + oNODE)/C[i];
      if V[0, 0]-1 in [1, 3] then
    end
  end
end; { of TF_plus_sensty }

procedure FR_plus_sensty; { frequency response + sensitivities }
var
  D1,D2,S1,S2,Gain,Phase,dD1,dD2,
  sS1,sS2,GainS,phaseS,RP,IP,denom_2,
  f5,RPs,IPs,dmag_2,epsilon: EXTENDED;
  xCODE,i,m,COMPARE: INTEGER;
  varMODIFY: TEXT;

  procedure body;
  var j: INTEGER;
  begin
    dD1 := 0;
    dD2 := 0;
    D1 := 0;
    D2 := 0;
    j := 0;

    { Frequency response evaluation }
    repeat
      dD1 := dD1 + D[1, j]*raise(-1, 2+j DIV 2)*raise(f5,j);
      D1 := D1 + D[2, j]*raise(-1, 2+j DIV 2)*raise(f5, j);
      j := j + 2
    until j > maxS;
    j := 1;
    repeat
      dD2 := dD2 + D[1, j]*raise(-1, trunc(1.5 + j/2))
        * raise(f5, j);
      D2 := D2 + D[2, j]*raise(-1, trunc(1.5 + j/2))
        * raise(f5, j);
      j := j + 2
    until j > maxS;
    if frequency > 1E18 then
    begin
      write(' ^C Too big frequency. ');
      write('Calculation aborted at this point.');
      delay(2000);
      xCODE := 11;
      exit
    end;
    denom_2 := D1*D1 + D2*D2;
    RP := (dD1*D1 + dD2*D2) / denom_2;

    IP := (dD2*D1 - D2*dD1) / denom_2;

    { . phase: deg = (180/pi)*rad
      . gain in nepers = ln(gain)
      . gain in dBs = 20 * log(gain)
                    = 20 * ln(gain) / ln(10)
                    = [20 /ln(10)] * gain in nepers }

  Phase := (180/Pi) * arctan(IP/RP);

  dmag_2 := RP*RP + IP*IP;
  Gain := 10 / ln(10)*ln(dmag_2);

  if Let.optimality then
  begin
    readln(fVAR2, desG, weight);
    epsilon := weight * (desG - Gain);
    writeln(varEPS,Raise(epsilon, leastP));
    writeln(varERR, Raise(epsilon, leastP-1));
    writeln(varB, Raise(epsilon, leastP-2))
  end;

  { Save final or optimum gain into file OPT_GAIN }
  if Let.Optimum_Gain then
    writeln(fVAR6, frequency, ' ', Gain);
  if Let.M2var = 2 then
  begin
    if not Let.optimality then { concerned with 1st iteration }
    begin
      write(frequency:12,' ',Gain:10:4,' ',
        Phase:10:4,' ',RP:12);
      writeln(' ', IP:12);

      { Save initial gain into file INI_GAIN }
      writeln(fVAR1, frequency, ' ', Gain);

      { Save real-part of phase into file REA_PART }
      writeln(fVAR3,frequency, ' ', RP);

      { Save imaginary-part of phase into file IMG_PART }
      writeln(fVAR4,frequency, ' ', IP);

      { Save initial phase into file PHAS_INI }
      writeln(fVAR5,frequency, ' ', Phase)
    end;
    exit
  end;

  i := 0;
  while i < RLCGtt do { F.R. sensitivity: the direct method }
  begin
    i := i + 1;
    sS1 := 0;
    sS2 := 0;
    S1 := 0;
    S2 := 0;
    j := 0;
    repeat
      sS1 := sS1 + E^[i, j]*raise(-1,j DIV 2)*raise(f5,j);
      S1 := S1 + F^[i, j]*raise(-1, j DIV 2)*raise(f5, j);
      j := j + 2
    until j > maxS;
    j := 1;
    repeat
```

```
        sS2 := sS2 + E^[i, j]*raise(-1, trunc(1.5 + j/2))
          * raise(f5, j);
        S2 := S2 + F^[i, j]*raise(-1, trunc(1.5 + j/2))
          * raise(f5, j);
        j := j + 2
      until j > maxS;
      RPs := ( (sS1 + S2*IP - S1*RP)*D1 +
            (sS2 - S2*RP - S1*IP)*D2 ) / denom_2;
      IPs := ( (sS2 - S2*RP - S1*IP)*D1 -
            (sS1 + S2*IP - S1*RP)*D2 ) / denom_2;
      GainS := 20 / ln(10)*( (RPs*RP + IPs*IP) / dmag_2 );
      PhaseS := 180 / pi*( (IPs*RP - RPs*IP) / dmag_2 );

      if Let.optimality then
      begin
        if (i > COMPARE) and not EOF(varMODIFY) then
          readln(varMODIFY, COMPARE);
        if i = COMPARE then
          writeln(varA, GainS)
      end;
      sS1 := U^[i, 0];
      if sS1 < 0 then
        U^[i, 0] := 4;

      if not Let.optimality then
        writeln(copy(compSET,2*U^[i,0]-1,2),U^[i,1],'  ',
          GainS:12,'  ',PhaseS:12,'  ',RPs:12,'  ',IPs:12)
    end;
    if not Let.optimality then
      writeln
  end; { of body }

  procedure closeALL;
  begin
    if Let.optimality then
    begin
      close(fVAR2);
      close(varEPS);
      close(varERR);
      close(varB);
      close(varA)
    end
  end;

begin { FR_plus_sensty }
  if not Let.optimality then
  begin
    Open(fVAR1, 'INI_GAIN.');
    Open(fVAR3, 'REA_PART.');
    Open(fVAR4, 'IMG_PART.');
    Open(fVAR5, 'PHAS_INI.');
    if Let.M2var = 2 then
    begin
      ClrScr;
      write(' FREQ., HZ    GAIN, DB   PHASE, DEG.  ');
      writeln('REAL PART    IMAG. PART')
    end
  end
  else
  begin
    Open(varA, 'AMATRIX.');
    Open(varB, 'BMATRIX.');
    Open(varERR, 'ERRMATX.');
    Open(varEPS, 'EPSMATX.');
    XRead(fVAR2, InFile)
  end;

  if Let.Optimum_Gain then
   Open(fVAR6,'OPT_GAIN.');
  frequency := initialF;
  m := 0;
  if Let.optimality then
  begin
    window(X1+21, Y1+12, X1+26, Y1+12);
    clrscr
  end;
  while m < POINTS do
  begin
    Let.FRS := true;
    m := m + 1;
    if Let.optimality then
    begin
      window(X1+31,Y1+7,X2-7,Y1+7);
      TextAttr := Black + Green * 16 + blink;
      if m = 1 then
        Clrscr;
      GotoXY(m, 1);
      write(chr(046));
      InitVideo;
      window(X1+31,Y1+7,X2-7,Y1+7);
      if m = 1 then
        Clrscr;
      GotoXY(m-1, 1);
      write(chr(046));
      window(X1+21, Y1+12, X1+27, Y1+12);
      write('  ',m)
    end;
    if m <> 1 then
      frequency := (frequency + f_incr) * f_scale;
    f5 := 2*Pi*frequency; { radian frequency }
    COMPARE := 0;
    if Let.optimality then
      XRead(varMODIFY, 'MODIFY.');
    body;
    if Let.optimality then
      close(varMODIFY);
    if xCODE = 11 then
    begin
      closeALL;
      exit
    end
  end;
  closeALL;
  if Let.Optimum_Gain then
  Close(fVAR6);
  if not Let.optimality then
  begin
    Close(fVAR1);
    Close(fVAR3);
    Close(fVAR4);
    Close(fVAR5)
  end
  else
  begin
```

```
      window(X2-30,Y1+12,X2-8,Y1+12);
      GotoXY(16, 1)
    end
end; { of FR_plus_sensty }

procedure initials;
{ . clear screen,
  . store top-left and bottom-right default window coordinates,
  . initialize text attributes as LightGray on blue,
  . initiaize global control variables in record "Let". }
begin
  clrscr;
  X1 := Lo(WindMin) + 1;
  Y1 := Hi(WindMin) + 1;
  X2 := Lo(WindMax) + 1;
  Y2 := Hi(WindMax) + 1;
  initVideo;
  with Let do
  begin
    M2var := 0;
    more := false;
    newINPUT := false;
    modify := false;
    M1 := false;
    M2 := false;
    freqs := false;
    ioNODES := false;
    list := false;
    evals := false;
    ct := false;
    CTFF := false;
    FRS := false;
    optimality := false;
    Optimum_Gain := false;
    LASTexit := 1
  end
end; { of initials }

procedure set_UNITY;
{ operations on modified incidence matrix:
    . row zero elements are set to unity
    . row 1 is arranged to contain a possible tree }
var
  jj,kk,LL: INTEGER;
begin
  for jj := 1 to tNODES do
  begin
    if V[0, jj] <> 1 then
    begin
      kk := V[1, jj];
      LL := V[V[0, jj], jj];
      V[V[0, jj], jj] := kk;
      V[1, jj] := LL;
      V[0, jj] := 1
    end
  end
end; { of set_UNITY }

procedure initDEF; { matrices D, E, F are set to zero }

  procedure initD;
  var i,j: INTEGER;
  begin
    for i := 1 to 2 do
    for j := 0 to maxS do
    D[i, j] := 0
  end; { of initD }

  procedure initEF;
  var i, j: INTEGER;
  begin
    for i := 0 to RLCGtt do
    for j := 0 to maxS do
    begin
      E^[i, j] := 0;
      F^[i, j] := 0
    end
  end; { of initEF }

begin
  if Let.optimality then
  begin
    initD;
    initEF;
    exit
  end;
  case Let.M1var of
    2: initD;
    3: initEF
  end;
  if Let.FRS then
  begin
    initD;
    initEF
  end
end; { of initDEF }

procedure initM2VARS;
{ initialization of variables in the second analysis menu }
begin
  if Let.LASTexit = 16 then
    Let.ct:=true
  else
  with Let do
    case M2var of
      8 : newINPUT := true;
      7 : freqs := true;
      6 : M1 := true;
      5 : ioNODES := true;
      4 : modify := true;
      3 : list := true;
      2 : evals := true;
      1 : continue := false
    end
end; { of initM2VARS }

procedure filterM2VARS;
{ selection of active command from the second analysis menu }
begin
  if Let.LASTexit <> 16 then
  begin
    Let.ct:=false;
    with Let do
    begin
```

```
        if M2var < > 8 then
          newINPUT := false;
        if M2var < > 7 then
          freqs := false;
        if M2var < > 6 then
          M1 := false;
        if M2var < > 5 then
          ioNODES := false;
        if M2var < > 4 then
          modify := false;
        if M2var < > 3 then
          list := false;
        if M2var < > 2 then
          evals := false;
        if M2var < > 1 then
          continue := true
      end
    end
end; { of filterM2VARS }

procedure check_TF_error;
{ depicts error message when computing frequency response
  without first obtaining transfer function }
begin
  Let.LASTexit := 1;
  if Let.evals then
    Let.M2 := false;
  if not Let.M2 then
  begin
    { EVALUATION }
    windows(4);
    with Let do
    if (switch>0) and not (M1var in [4,5]) then
    begin
      LASTexit := 8;
      exit
    end;
    ClrScr;
    shadow(#7'FATAL ERROR: COMPUTE TRANSFER
    FUNCTION FIRST.');
    Let.CTFF := false
  end;
end; { of check_TF_error }

procedure check_FREQ_RESP_error;
{ checks if sensitivities are computed before frequency response }
begin
  windows(4);
  if Let.M1var < > 2 then
    if Let.FRS then
      writeln('CPT NO.  DB SEN       DEG SEN    ',
        '  RP SEN       IP SEN');
  if Let.M1var < > 2 then
    if not Let.FRS then
      shadow(#7'Frequency response sensitivities not ready.')
end; { of check_FREQ_RESP_error }
END. { of unit ANAL1 }


{$N+}
PROGRAM GRAPHER;
(*-------------------------------------------------------------
```

This program plots and indicates results from program ANALOPT & is written as part of the M. Sc. thesis "design of electrical networks through computer-aided optimization techniques." There are 21 procedures and one function. Ten of the procedures are "parent" and five of these nest ten other procedures & the function. All of the 22 routines are finally integrated by the main program. A complete listing of the 22 routines is idicated below where nested routines are indicated indented from their nesting counterparts ...

```
  procedure GetReady;

  procedure initVideo;

  procedure GRAPHERmessage;

  procedure Get_Command;
   procedure SHADING;

  procedure seek_FILE;
   procedure HELP;

  procedure grapherCONST;

  procedure plot_ID;

  procedure BackGround;
   Function ConvertReal(R: REAL; w,d: INTEGER): STRING;
   procedure setNMBRS(clr, fnt, dire, syze, HORjust, VERjust:
                                                  INTEGER);
   procedure setLBLS(clr,fnt,dire,syze,HORjust,VERjust:
                                          INTEGER);
   procedure axsMARK(dvsn,change1,change2: INTEGER);
   procedure axsLBL(dvsn: INTEGER; a,z: REAL; fld,deci:
                                          INTEGER);
   procedure axsLBL_in_EXP(dvsn: INTEGER; a,z: REAL);

  procedure the_MEAT_of_GRAPHER;

  procedure StartPlot;
   procedure MinMax(message: STRING; bb: INTEGER);
   procedure CALLHELP;

  procedure FinishPlot;
   procedure NumberOfData;
-------------------------------------------------------------*)


{.....................}
Uses Crt,Graph,Anal0;
{.....................}

(*-------------------------------------------------------------*)
var
  tempoC  : TEXT; { tempo file for information in file "Command" }
  PFI     : pfiREC;{ PFI = plot format information, a record var }
  yI,yF,yT: BYTE;  { Initial/Final/Total y-columns in "Command" }
  s       : STRING; { used in a process of string conversion }
  hold    : CHAR;   { selects b/n zero- & first-order-hold plots }
  Look    : BOOLEAN;{ true if procedure seek_FILE is used }
  i       : INTEGER;{ used as the starting index in FOR statements }
  ch      : CHAR;   { selects between exit and continue plot      }
(*-------------------------------------------------------------*)
```

```
procedure GetReady; { Prepares Turbo Pascal for graphics tasks }
var d,m: INTEGER;
begin
   DetectGraph(d,m);
   Initgraph(d,m,'c:\TP5')
end; { of GetReady }

procedure initVideo; { Initial text attributes = LightGray-on-Blue }
begin
  TextAttr := LightGray + blue * 16
end; { of initVideo }

procedure GRAPHERmessage;
begin
  TextColor(LightCyan);
  writeln('welcome to the OUTPUT stage!!');
  TextColor(LightGray);
  writeln;
  writeln('    Graphic and numeric commands from the OUTPUT
    MENU below are used to display');
  writeln('results obtained so far. The Graphic commands are
    S,I,O,M,R,P,T,D,E,N,G,F while');
  writeln('letter L is used in the numeric case.');
  writeln;
  writeln('    The latter chioce leads to a LOOK WHAT prompt,
  where the contents of a file');
  writeln('may be read by typing the name of the file.  Also, there is
    a HELP available at');
  writeln('the LOOK WHAT prompt and furnishes list of PPOND
    files ready for LOOKing.');
  writeln;
  writeln('    The command to quit an OUTPUT stage is Q.  Letter x
    followed by  the  ENTER');
  writeln('key takes back to the OUTPUT MENU from the LOOK
    WHAT prompt.');
  writeln;
  write('press any key to continue ... ');
  ch := readkey;
  gotoxy(1,whereY);
  write('                              ');
end; { of GRAPHERmessage }

procedure Get_Command;
(*** Fifteen commands are defined and used ***)
var
   i: INTEGER;
   Letter: CHAR;
   message: STRING;

   procedure SHADING;
   begin
     TextAttr := LightGray + 16 * Blue;
     writeln;
     TextAttr := LightGray + Brown * 16;
     GotoXY(whereX+13, whereY);
   end;

begin { Get_Command }
  writeln;
  TextColor(LightCyan);
  writeln;
  writeln;
  TextAttr := Yellow + 16 * Cyan;
  GotoXY(whereX+13, whereY);
  write('       OUTPUT MENU: select command ...        ');
  SHADING;
  write(' SPC_GAIN       INI_GAIN          OPT_GAIN ');
  SHADING;
  write(' IMG_PART       REA_PART          PHAS_INI ');
  SHADING;
  write(' ERR_ITER       ERR_DIST          ERR_FREQ ');
  SHADING;
  write(' INI_SPCG       OPT_SPCG          FIS_GAIN ');
  SHADING;
  write(' LOOK           CALLHELP          QUIT     ');
  SHADING;
  write('                                          ');
  TextAttr := LightGray + 16 * Blue;
  writeln;
  TextAttr := LightGray + 16 * Blue;
  GotoXY(whereX+1,whereY-6);
  highvideo;
  GotoXY(16, whereY);
  write('S');
  GotoXY(whereX+16, whereY);
  write('I');
  GotoXY(whereX+18, whereY);
  write('O');
  GotoXY(17, whereY+1);
  write('M');
  GotoXY(whereX+15, whereY);
  write('R');
  GotoXY(whereX+18, whereY);
  write('P');
  GotoXY(21, whereY+1);
  write('T');
  GotoXY(whereX+15, whereY);
  write('D');
  GotoXY(whereX+14, whereY);
  write('E');
  GotoXY(17, whereY+1);
  write('N');
  GotoXY(whereX+22, whereY);
  write('G');
  GotoXY(whereX+11, whereY);
  write('F');
  GotoXY(16, whereY+1);
  write('L');
  GotoXY(whereX+20, whereY);
  write('H');
  GotoXY(whereX+14, whereY);
  writeln('Q');
  lowvideo;
  writeln;
  writeln;
  i := 0;
  repeat
    if i = 0 then
    begin
      TEXTcolor(11);
      message := 'SELECTION: '
    end
    else
    begin
```

```
        TEXTcolor(12);
        if odd(i) then
          message := #7'Retype. '
        else
          message := #7' Retype.'
      end;
      write(message);

      if message < > 'SELECTION: ' then
        GotoXY(whereX - length(message) + 1, whereY);
      Letter := UpCase(readkey);
      if Letter = #0 then
      begin
        Letter := readkey;
        Letter := '$';
      end;
      i := i + 1
    until Letter in
    ['H','S','I','O','M','R','P','T','D','E','N','G','F','L','Q'];
    TextColor(11);
    case Letter of
      'H': Command := 'CALLHELP';
      'S': Command := 'SPC_GAIN';
      'I': Command := 'INI_GAIN';
      'O': Command := 'OPT_GAIN';
      'M': Command := 'IMG_PART';
      'R': Command := 'REA_PART';
      'P': Command := 'PHAS_INI';
      'T': Command := 'ERR_ITER';
      'D': Command := 'ERR_DIST';
      'E': Command := 'ERR_FREQ';
      'N': Command := 'INI_SPCG';
      'G': Command := 'OPT_SPCG';
      'F': Command := 'FIS_GAIN';
      'L': Command := 'LOOK';
      'Q': Command := 'QUIT'
    end;
    writeln(Command);
    if (Command < > 'QUIT') and
      (Command < > 'LOOK') and
      (LastCommand < > Command) and
      (Command < > 'CALLHELP') then
      begin
        write('Nature of plot:  1 = First-Order-Hold,');
        write(' 0 = Zero-Order-Hold ? ');
        repeat
          hold := readkey
        until hold in ['0','1']
      end
  end; { of Get_Command }

  procedure seek_FILE;
  (*** Shows both contents and length of a disk file ***)
  var i,mistakes,file_LENGTH: INTEGER;
    page, ch: CHAR;
    zis: FILE OF CHAR; ziss: TEXT;

    procedure HELP;
    begin
      TextColor(Yellow);
      writeln;
      writeln(' HELP on LOOKable files of network ', InFile,': ');
      writeln('   SPC_GAIN       INI_GAIN       OPT_GAIN ');
      writeln('   IMG_PART       REA_PART       PHAS_INI ');
      writeln('   ERR_ITER       ERR_DIST       ERR_FREQ ');
      writeln('   INI_SPCG       OPT_SPCG       FIS_GAIN ');
      write('   CALLHELP');
      GotoXY(20,whereY);
      write(InFile + '.TRL');
      GotoXY(36,whereY);
      writeln(InFile + '.LOD');
      GotoXY(4,whereY);
      write(InFile + '.SYM');
      GotoXY(20,whereY);
      write(InFile + '.BCL');
      GotoXY(36,whereY);
      writeln(InFile + '.CIR');
      GotoXY(4,whereY);
      write(InFile + '.SPC');
      GotoXY(20,whereY);
      writeln(InFile + '.OPT');
      writeln;
      TextColor(LightCyan);
      write('   press any key to continue ... ');
      ch := readkey
    end;

  begin { seek_FILE }
    repeat
      repeat
        clrscr;
        write('LOOK WHAT, [x = EXIT;   h = HELP] ? ');
        repeat
          readln(Command);
          if command = 'x' then
            Command := 'X';
          if command = 'h' then
          begin
            Command := 'H';
            HELP
          end;
          if (Command < > 'X') and (Command < > 'H') then
          begin
            assign(zis, Command);
            {$I-}
              reset(zis);
              mistakes := IOresult
            {$I+};
            if mistakes < > 0 then
              write('File not found. Retype ... ')
          end;
          if Command = 'H' then
          begin
            clrscr;
            write('LOOK WHAT, [x = EXIT;   h = HELP] ? ')
          end
        until (mistakes = 0) or (Command = 'X');
        if Command < > 'X' then
        begin
          repeat
            write('Show by page, Y/N ?');
            page := UpCase(readkey);
            if not (page in ['Y','N']) then
            begin
```

```
            write(^G);
            GotoXY(1,whereY)
          end
        until page in ['Y','N'];
        writeln;
        XRead(ziss,Command);
        file_LENGTH := -1;
        for i := 0 to filesize(zis)-1 do
        begin
          seek(zis,i);
          read(zis,ch);
          read(ziss,ch);
          write(ch);
          if EOLn(ziss) then
          begin
            mistakes:=mistakes+1;
            file_LENGTH := file_LENGTH + 1;
          end;
          if mistakes = hi(windMAX) then
          begin
            mistakes := 0;
            if page = 'Y' then
              ch := readkey
          end
        end;
        writeln;
        getUPPERcase(Command);
        if file_length = 0 then
          writeln('END OF ',Command,': FILE IS EMPTY!')
        else if file_length > -1 then
          writeln('END OF ',Command,': ',file_LENGTH,'
            LINES.');
        close(zis);
        if (file_length = -1) and (mistakes = 0) and
          (Command < > '') then
            writeln('Large file. Cannot LOOK!');
        if Command = '' then
        writeln('No file name used!');
        write('Press any key to continue.');
        GotoXY(whereX-17, whereY);
        ch := readkey
      end
    until (Command = 'CALLHELP')
          or
        (Command = 'SPC_GAIN')
          or
        (Command = 'INI_GAIN') or (Command = 'OPT_GAIN')
          or
        (Command = 'IMG_PART') or (Command =
          'REA_PART') or
        (Command = 'PHAS_INI') or (Command = 'ERR_ITER')
          or
        (Command = 'ERR_DIST') or (Command = 'ERR_FREQ')
          or
        (Command = 'INI_SPCG') or (Command = 'OPT_SPCG')
          or
        (Command = 'FIS_GAIN') or (Command = 'LOOK')
          or
        (Command = 'QUIT') or (Command = 'X')
  until Command = 'X';
  Look := true
end; { of seek_FILE }
```

```
procedure grapherCONST;
(*** Stores plotting constants used in program GRAPHER ***)
var
   kk: INTEGER;
begin { grapherCONST }
  with PFI do
  begin

    { Margins: Left/Right/Top/Bottom }
    ML := 75;      MR := 20;
    MT := 60;      MB := 50;

    { x-numbering/labeling, y-numbering/lableling, title ...  }
      (* offsets *)
    XNO := 5;      XLO := 25;
    YNO := 5;      YLO := 65;     TIO := 10;

      (* fonts *)
    XNF := 2;      XLF := 2;
    YNF := 2;      YLF := 2;      TIF := 2;

      (* sizes *)
    XNS := 5;      XLS := 6;
    YNS := 5;      YLS := 6;      TIS := 7;

      (* colors *)
    XNC := 12;     XLC := 15;
    YNC := 12;     YLC := 15;     TIC := 15;

    { SetLineStyle constants: Axes/Divisions ... }
    AS := 0;      DS := 1;  (* LineStyle *)
    AP := 0;      DP := 0;  (* pattern   *)
    AT := 1;      DT := 1;  (* thickness *)

    AC := 3;      DC := 3   { Colors: Axes/Divisions }
  end
End; { of grapherCONST }

procedure plot_ID;
{ Curve Name CN is written using font CNF/Size CNS/Direction CND/
  Color CNC & central location (CNx,CNy). The plot margins ML/MT/
  MR/MB, coordinates XMAX/YMAX are used for scaling + shifting.}

var
  h,w: INTEGER; { height/width of CNS string. }
begin { plot_ID }
  with PFI do
  begin
    if (XRH-XRL) = 0 then { do not halt }
      CNx := ((XMAX-ML-MR)*(CNx-XRL)/0.00001)
    else { update CNx }
      CNx := ((XMAX-ML-MR)*(CNx-XRL)/(XRH-XRL));
    if (YRL-YRH) = 0 then { do not halt }
      CNy := YMAX - MT - MB -
        ((YMAX-MT-MB)*(CNy-YRL)/0.00001)
    else { update CNy }
      CNy := YMAX - MT - MB -
        ((YMAX-MT-MB)*(CNy-YRL)/(YRH-YRL));

    h := Round(1.1 * TextHeight(CN));
    w := Round(1.1 * TextWidth(CN));
    SetTextStyle(CNF,CND,CNS);
```

```
    SetColor(CNC);
    SetTextJustify(CenterText,CenterText);

    { write curve name }
    if CN = 'ERR_ITER' then
      OutTextXY(round(CNx),round(CNy-16),CN)
    else if CN = 'OPT_GAIN' then
      OutTextXY(round(CNx + 24),round(CNy+8),CN)
    else
      OutTextXY(round(CNx),round(CNy+8),CN)
  end
end; { plot_ID }

procedure BackGround;
{ Draws axes/labels/numbers/title using PFI information }
var x1,y1,x2,y2,incr,Xlen,Ylen,ii,i,d,m: INTEGER; sss: STRING;

  Function ConvertReal(R: REAL; w,d: INTEGER): STRING;
  { Real to string conversion in the format R:w:d. }
  begin
    Str(R:w:d, s);
    ConvertReal := s
  end; { of ConvertReal }

  procedure setNMBRS(clr,fnt,dire,syze,HORjust,VERjust:
                                        INTEGER);
  { The environment upon which numbers are written is defined }
  begin
    SetColor(clr);              {          ... color }
    SetTextStyle(fnt,dire,syze); { ... font/direction/size}
    SetTextJustify(HORjust,VERjust)    {...justifications}
  end; { of setNMBRS }

  procedure setLABELS(clr,fnt,dire,syze,HORjust,VERjust:
                                        INTEGER);
  { The environment upon which strings are written is defined }
  begin
    SetColor(clr);              {          ... color }
    SetTextStyle(fnt,dire,syze); { ... font/direction/size}
    SetTextJustify(HORjust,VERjust)    {...justifications}
  end; { of setLABELS }

  procedure axsMARK(dvsn,change1,change2:INTEGER);
  { Marks x/y axes with bars for easy identification of values }
  begin { axsMARK }
    SetLineStyle(0,1,3);
    if dvsn <= 10 then
      Line(x1, y1, x1+change1, y1-change2);
    incr := 0;
    repeat
      if (dvsn > (10+incr)) and (dvsn <= (20+incr)) then
        if (i mod ((20+incr) div 10))=0 then
          Line(x1, y1, x1+change1, y1-change2);
      incr := incr + 10
    until incr > 30;
    with PFI do
      SetLineStyle(DS,DP,DT)
  end; { of axsMARK }

  procedure axsLBL(dvsn: INTEGER; a,z: REAL; fld,deci,axs:
                                        INTEGER);
  { Labels axes; a and z are x/y low and high range values. }
  begin { axsLBL }
    if dvsn <= 10 then
      OutTextXY(x1,y1,
        ConvertReal((a+i*(z-a)/dvsn)/PFI.CSC,fld,deci));
    incr := 0;
    repeat
      if (dvsn > (10+incr)) and (dvsn <= (20+incr)) then
        if (i mod ((20+incr) div 10)) = 0 then
          OutTextXY(x1,y1,
            ConvertReal((a+i*(z-a)/dvsn)/PFI.CSC,fld,deci));
      incr := incr + 10
    until incr > 30;
  end; { of axsLBL }

  procedure axsLBL_in_EXP(dvsn: INTEGER; a,z,axs: REAL);
  { used when numbers > 1E+05 or  < 1E-05 are involved }
  begin { axsLBL_in_EXP }
    str(((a+i*(z-a)/dvsn)/PFI.CSC):9, sss);
    if dvsn <= 10 then
      OutTextXY(x1,y1,sss);
    incr := 0;
    repeat
      if (dvsn > (10+incr)) and (dvsn <= (20+incr)) then
        if (i mod ((20+incr) div 10)) = 0 then
          begin
            if i = ii then
              str(((a+i*(z-a)/dvsn)/PFI.CSC):5,sss)
            else
              str(((a+i*(z-a)/dvsn)/PFI.CSC):9,sss);
            OutTextXY(x1,y1,sss)
          end;
      incr := incr + 10
    until incr > 30;
  end; { axsLBL_in_EXP }

begin { BackGround }
  with PFI do
  begin
    XMAX := GetMaxX;
    YMAX := GetMaxY;

    { Axes: line drawing  ... }
    SetColor(AC);
    SetLineStyle(AS, AP, AT);
    Line(ML, YMAX-MB, XMAX-MR, YMAX-MB); { X axis }
    Line(ML, YMAX-MB, ML, MT); { Y axis }

    { Grids: x-axis ... }
    SetColor(DC);
    SetLineStyle(DS,DP,DT);
    Xlen := XMAX - ML - MR; { Xlen = x-axis line length }
    For i := 1 to XAD Do
    begin
      x1 := ML + Round(XLen*i/XAD);
      y1 := YMAX - MB;
      y2 := MT;
      Line(x1,y1,x1,y2);
      if y1 > YMAX-MB-5 then
        axsMARK(XAD,0,5)
    end;

    { Grids: y-axis  ... }
```

```
    Ylen := YMAX - MT - MB; (* Ylen = y-axis line length *)
    for i := 1 to YAD Do
    begin
      x1 := ML;
      x2 := XMAX - MR;
      y1 := YMAX - MB - Round(Ylen*i/YAD);
      Line(x1,y1,x2,y1);
      if x1 < ML+5 then
        axsMARK(YAD,5,0)
    end;

    { Numbering: x-axis: HorizDir=0; CenterText=1; TopText=2 }
    setNMBRS(XNC,XNF,0,XNS,1,2);
    y1 := YMAX - MB + XNO;
    if Command = 'ERR_ITER' then
      XFD := 0;
    For i := 0 to XAD Do
    begin
      x1 := ML + Round(Xlen*i/XAD);
      axsLBL(XAD,XRL,XRH,0,XFD,0)
    end;

    { Numbering: y-axis: RightText = 2;}
    setNMBRS(YNC,YNF,0,YNS,2,1);
    x1 := ML - YNO;
    {if (Command = 'ERR_ITER') or - rem: optional -
      (Command = 'ERR_FREQ') or
      (Command = 'ERR_DIST') then
    begin
      YFD := 2;
      YRL := 0
    end;}
    ii := YAD div 2 ;
    For i := 0 to YAD do
    begin
      y1 := YMAX - MB - Round(i*YLen/YAD);
      if (abs(YRH) > 1E+05) or (abs(YRL) < 1E-05) then
        axsLBL_in_EXP(YAD,YRL,YRH,1)
      else
        axsLBL(YAD,YRL,YRH,0,YFD,1)
    end;

    { Axes Labeling ... }
    setLABELS(XLC,XLF,0,XLS,1,2);
    OutTextXY(ML + XLEN DIV 2, YMAX - MB + XLO,XL);
    setLABELS(YLC,YLF,1,YLS,2,1);
    OutTextXY(ML - YLO, MT + Ylen DIV 2,YL);

    { Graph Title ... }
    SetColor(TIC);
    SetTextStyle(TIF,HorizDir,TIS);
    SetTextJustify(CenterText,BottomText);
    OutTextXY(ML + XLEN DIV 2, MT - TIO,Title);
    SetViewPort(ML, MT, XMAX-MR, YMAX-MB, true)
  end
end; { of BackGround }

procedure the_MEAT_of_GRAPHER;
(*** Performs the basic plotting task by drawing a line b/n
    the points (xa1,ya1) and (xa2,ya2) ***)
var
  x1, y1, x2, y2: INTEGER;
begin { the_MEAT_of_GRAPHER }
  { x1/y1/x2/y2 are obtained after shifting and scaling using
  xa1/ya1/xa2/ya2, the plot margins, XMAX and YMAX.}
  with PFI do
  begin
    if (XRH-XRL) = 0 then
      x1 := Round((XMAX-ML-MR)*(xa1-XRL)/0.00001)
    else
      x1 := Round((XMAX-ML-MR)*(xa1-XRL)/(XRH-XRL));
    if (XRH-XRL) = 0 then
      x2 := Round((XMAX-ML-MR)*(xa2-XRL)/0.00001)
    else
      x2 := Round((XMAX-ML-MR)*(xa2-XRL)/(XRH-XRL));
    if (YRH-YRL) = 0 then
      y1 := YMAX-MT-MB-Round((YMAX-MT-MB)*
        (CSC*ya1-YRL)/0.00001)
    else
      y1 := YMAX - MT - MB -
        Round((YMAX-MT-MB)*(CSC*ya1-YRL)/(YRH-YRL));
    if (YRH-YRL) = 0 then
      y2 := YMAX - MT - MB -
        Round((YMAX-MT-MB)*(CSC*ya2-YRL)/0.00001)
    else
      y2 := YMAX - MT - MB -
        Round((YMAX-MT-MB)*(CSC*ya2-YRL)/(YRH-YRL));
    if (hold = '0') and (yT = yF) then
    begin
      Line(x1,y1,x2,y1);
      Line(x2,y1,x2,y2)
    end
    else if Command = 'ERR_DIST' then
    begin
      circle(x1,y1,2);
      circle(x2,y2,2);
      Line(x1, y1, x1, YMAX - MB);
      Line(x2, y2, x2, YMAX - MB)
    end
    else
      Line(x1,y1,x2,y2)
  end
End; { of the_MEAT_of_GRAPHER }

procedure StartPlot;
(*** This procedure is called by the main program ***)
var kk    : INTEGER;
  XRLL  : STRING;
  aa    : REAL;
  chh,ch : CHAR;

  procedure MinMax(message: STRING; bb: INTEGER);
  var i: INTEGER;
  (***
      Shows default values for x-y divisions, min-max values,
      curve scale and then offers possibility of modification
  ***)
  begin
    TEXTcolor(LightCyan);
    write(message);
    TEXTcolor(White);
    with PFI do
    case bb of
      1: writeln(XRL:0:2);
```

```
    2: writeln(XRH:0:2);
    3: writeln(YRL:0:2);
    4: writeln(YRH:0:2);
    5: writeln(XAD);
    6: writeln(YAD);
    7: writeln(CSC:0:2)
  end;
  if bb in [1..4] then
    GotoXY(19,whereY-1);
  if bb = 5 then
    GotoXY(31,whereY-1);
  if bb = 6 then
    GotoXY(50,whereY-1);
  if bb = 7 then
    GotoXY(34,whereY-1);
  ch := '£';
  chh := '$';
  repeat
    ch := readkey;
    if ch = ' ' then
      gotoXY(whereX+3, whereY);
    write('                ');
    GotoXY(whereX-17, whereY);
    if ch in ['0'..'9','-'] then
    begin
      kk := 0;
      repeat
        XRLL := '';
        chh := ch;
        repeat
          if kk = 0 then
            write(chh);
          kk := 0;
          XRLL := XRLL + chh;
          chh := readkey
        until chh=#13;
        val(XRLL,aa,kk);
        if kk < > 0 then
        begin
          write(#7);
          gotoxy( whereX - length(XRLL), whereY);
          for i := 1 to length(XRLL) do
          write(' ');
          gotoxy( whereX - length(XRLL) + 1, whereY)
        end;
      until kk = 0
    end;
    if (ch < > ' ') and (chh < > #13) then
    begin
      TEXTcolor(12);
      GotoXY(1,whereY+1);
      write('Error: RETYPE or use SPACE BAR for none.');
      if bb in [1..4] then
        GotoXY(19,whereY-1);
      if bb = 5 then
        GotoXY(31,whereY-1);
      if bb = 6 then
        GotoXY(50,whereY-1);
      if bb = 7 then
        GotoXY(34,whereY-1)
    end
  until (ch = ' ') or (chh = #13);
  GotoXY(1,whereY+1);
  initVideo;
  writeln(' ( RETYPE or use SPACE BAR for none )
                                                  ');
  GotoXY(19,whereY-2)
end; { of MinMax }

procedure CALLHELP;
(*** Generates and shows contents of a "help" file ***)
var i: INTEGER;
begin { CALLHELP }
  Open(fVAR1,'CALLHELP.');
  for i := 1 to 75 do
  write(chr(196));
  writeln;
  writeln(fVAR1,'COMMAND   FUNCTION   DESCRIPTION');
  write(fvAR1,'  S        SPC_GAIN');
  writeln(fvAR1,'     specified gain;');
  write(fvAR1,'  I        INI_GAIN');
  writeln(fvAR1,'     initial gain;');
  write(fvAR1,'  O        OPT_GAIN');
  writeln(fvAR1,'     final or optimum gain;');
  write(fvAR1,'  M        IMG_PART');
  writeln(fvAR1,'     imaginary part of initial phase;');
  write(fvAR1,'  R        REA_PART');
  writeln(fvAR1,'     real part of initial phase;');
  write(fvAR1,'  P        PHAS_INI');
  writeln(fvAR1,'     initial phase;');
  write(fvAR1,'  T        ERR_ITER');
  writeln(fvAR1,'     error function versus iteration;');
  write(fvAR1,'  D        ERR_DIST');
  writeln(fvAR1,'     distribution of discrete error values;');
  write(fvAR1,'  E        ERR_FREQ');
  writeln(fvAR1,'     error function versus frequency;');
  write(fvAR1,'  N        INI_SPCG');
  write(fvAR1,'     specified & initial gains');
  writeln(fVAR1,' as frequency functions;');
  write(fvAR1,'  G        OPT_SPCG');
  write(fvAR1,'     specified & optimium gains');
  writeln(fVAR1,' as frequency functions;');
  write(fVAR1,'  F        FIS_GAIN');
  writeln(fVAR1,'     final, initial, and specified gains;');
  write(fvAR1,'  H        CALLHELP     help on plotting');
  writeln(fVAR1,' commands and GRAPHER unit;');
  write(fvAR1,'  L        LOOK    ');
  writeln(fvAR1,'     shows contents of a file;');
  write(fvAR1,'  Q        QUIT    ');
  writeln(fVAR1,'     quits a plotting session.');
  writeln(fVAR1);
  write(fVAR1,'- At the LOOK prompt, you may type any of the');
  writeln(fVAR1,' first 13 names under "FUNCTION".');
  write(fVAR1,'- A plot is first-order-hold if a ');
  writeln(fVAR1,'one-to-one function. Else, zero-order-hold.');
  write(fVAR1,'- TRL, LOD, SYM, BCL, CIR, SPC & OPT input');
  writeln(fVAR1,' file extensions may also be LOOKed.');
  write(fVAR1,'                                    ');
  writeln(fVAR1,'- Press any key to continue - ');
  Close(fVAR1);
  XRead(fVAR1,'CALLHELP.');
  while not EOF(fVAR1) do
  begin
    Readln(fVAR1,s);
```

```
        writeln(s)
      end;
      GotoXY(38,whereY-1);
    end; { of CALLHELP }

begin { StartPlot }
  Look := false;
  XRead(fVAR1, 'INI_SPCG.');
  readln(fVAR1, InFile);
  close(fVAR1);
  Get_Command;
  if Command = 'CALLHELP' then
    CALLHELP
  else if Command = 'LOOK' then
    seek_FILE
  else
  begin
    if Command = 'QUIT' then
      exit;
    with PFI do
    begin
      if Command='TestGraf' then
      begin
        XL := 'abscissa test values';
        YL := 'ordinate test values';
        Title := 'Testing Capabilities of the Grapher Unit'
      end
      else
      if (Command = 'REA_PART') or
        (Command = 'IMG_PART') or
        (Command = 'PHAS_INI') then
      begin
        XL := 'Frequency, Hertz';
        YL := 'Phase, degrees'
      end
      else if Command = 'ERR_ITER' then
      begin
        XL := 'Number of iterations';
        YL := 'Error'
      end
      else if (Command = 'ERR_FREQ') or
           (Command = 'ERR_DIST') then
      begin
        XL := 'Frequency, Hz';
        YL := 'Error'
      end
      else
      begin
        XL := 'Frequency, Hertz';
        YL := 'Gain, dB'
      end;

      if Command = 'SPC_GAIN' then
        Title := 'Plot of Specified Gain';

      if Command = 'INI_GAIN' then
        Title := 'Plot of Initial Gain';
      if Command = 'OPT_GAIN' then
        Title := 'Final Gain';

      if Command = 'FIS_GAIN' then
        Title := 'Initial - Final - Specified Gains';

      if Command = 'ERR_ITER' then
        Title := 'Error Values at Different Iterations';

      if Command = 'ERR_FREQ' then
        Title := 'Frequency Vs Error values';

      if Command = 'ERR_DIST' then
        Title := 'Distibution of Individual Errors';

      if Command = 'REA_PART' then
        Title := 'Real Part of Initial Gain';

      if Command = 'IMG_PART' then
        Title := 'Imaginary Part of Initial Gain';

      if Command = 'PHAS_INI' then
        Title := 'Initial Phase';

      if Command = 'INI_SPCG' then
        Title := 'Specified and Initial Gains';

      if Command = 'OPT_SPCG' then
        Title := 'Specified and Optimum Gains';
      Title := InFile + ': ' + Title
    end;

    grapherCONST; { Get the graph formatting constants }
    kk := 0;
    GetReady; { Initialize Graphics }
    setBKColor(1);
    OutTextXY(GetmaxX div 2 - 12*8,GetmaxY-40,
      ' * Command Accepted * ');
    OutTextXY(GetmaxX div 2 - 12*8,GetmaxY-20,
      ' * please wait * ');

    with PFI do  { search for extreme x-y values in data file. }
    begin
      yI := 1;   { Initial value of y }
      if Command='TestGraf' then
        yF := 6 { Final value of y }
      else if Command = 'FIS_GAIN' then
        yF := 3
      else if (command = 'INI_SPCG') or
        (Command = 'OPT_SPCG') then
        yF := 2
      else
        yF := 1;
      XRead(tempoC,Command);

      if LastCommand <> Command then
      for yT := yI to yF do
      begin
        Reset(tempoC);

        if Command = 'INI_SPCG' then
          readln(tempoC, InFile);
        { take (xa1,ya1) as 1st data point }
        Read(tempoC,xa1);
        for i := 1 to yT do
        Read(tempoC,ya1);
        Readln(tempoC);
```

```
      { compare first data point with maxima and minima... }
      if kk=0 then
      begin
        XRL := xa1;
        XRH := xa1;
        YRL := ya1;
        YRH := ya1
      end
      else
        begin
          if xa1 >XRL then
            if xa1 >XRH then
              XRH := xa1;
          if xa1 < XRH then
            if xa1 <=XRL then
              XRL := xa1;
          if ya1 > YRL then
            if ya1 >YRH then
              YRH := ya1;
          if ya1 < YRH then
            if ya1 <= YRL then
              YRL := ya1
        end;

      { continue reading and comparing ... }
      while not EOF(tempoC) Do
      begin
        read(tempoC,xa2);
        for i  :=  1 to yT do
          read(tempoC,ya2);
        readln(tempoC);
        if xa2 > XRL then
          if xa2 > XRH then
            XRH := xa2;
        if xa2 < XRH then
          if xa2 <= XRL then
            XRL := xa2;
        if ya2 > YRL then
          if ya2 > YRH then
            YRH := ya2;
        if ya2 < YRH then
          if ya2 <= YRL then
            YRL := ya2;
        xa1 := xa2;
        ya1 := ya2  { Last=1st point for next comparison }
      end; { of current data column }
      kk := kk+1
    end
end; { of the With statement }
Close(tempoC);

textmode(LastMode);  { exit graphics and go to text mode }
initVideo;
clrscr;
with PFI do
begin
  for i := 1 to hi(windmax)+1 do
  writeln;
  CSC := 1;
  XAD := 10;
  if (Command = 'ERR_ITER') or
    (Command = 'ERR_DIST') or
    (Command = 'ERR_FREQ') then
      XAD := Trunc(PFI.XRH);
  YAD := 10;
  if (Command = 'ERR_ITER') or
    (Command = 'ERR_DIST') or
    (Command = 'ERR_FREQ') then
      YAD := 50;
  writeln(' ( RETYPE or use SPACE BAR for none )');
  GotoXY(1,whereY-2);

  MinMax('Number of divisions: x-axis = ', 5);
  if ch <> ' ' then
   if chh = #13 then
     XAD := Round(aa);

  GotoXY(whereX+22, whereY);
  MinMax('y-axis = ', 6);
  if ch <> ' ' then
   if chh = #13 then
     YAD := Round(aa);
  repeat
    if (not (PFI.XAD in [1..50]))
       or (not (PFI.YAD in [1..50])) then
    begin
      GotoXY(1, whereY);
      TextColor(LightRed);
      write(#7'Bad value: valid range is 1 - 50 inclusive.');
      writeln('                              ');
      write('number of divisions: (x-axis,y-axis) = ');
      TextColor(White);
      readln(XAD,YAD);
      TextColor(LightGray)
    end
    else
      i := 123
  until i=123;
  XFW := 0; { x-numbering width    }
  XFD := 2; { x-numbering decimals }
  YFW := 0; { y-numbering width    }
  YFD := 2  { y-numbering decimals }
end;
GotoXY(1, whereY + 1);
write('                                        ');
for i := 1 to hi(windmax)-2
do writeln(' ');
writeln(' ( RETYPE or use SPACE BAR for none )');
GotoXY(1,whereY-2);
MinMax('x-values: Xmin =  ',1);
if ch <> ' ' then
  if chh = #13 then
    PFI.XRL := aa;
GotoXY(11,whereY);
for i := 1 to 50 do
  write(' ');
GotoXY(11,whereY);
MinMax('Xmax =  ',2);
if ch <> ' ' then
  if chh = #13 then
    PFI.XRH := aa;
gotoXY(1,whereY);
for i := 1 to 50 do
  write(' ');
```

```
    GotoXY(1,whereY);
    MinMax('y-values: Ymin =  ', 3);
    if ch <> ' ' then
      if chh = #13 then
        PFI.YRL := aa;
    GotoXY(11,whereY);
    for i := 1 to 50 do
    write(' ');
    GotoXY(11,whereY);
    MinMax('Ymax =  ', 4);
    if ch <> ' ' then
      if chh = #13 then
        PFI.YRH := aa;
    GotoXY(1,whereY);
    MinMax('Curve scale, 1 = default value ? ', 7);
    if ch <> ' ' then
      if chh = #13 then
          PFI.CSC := aa;
    GetReady;     (* return to graphics mode ********)
    setcolor(15); (* use white color for foreground *)
    BackGround;   (* draw the graph background ******)
  end
end; { of StartPlot }

procedure FinishPlot;
(*** This procedure is also called by the main program ***)
var
  i,j,jj: INTEGER; ch: CHAR;
  procedure NumberOfData;
  { Counts the number of data lines for a given plot }
  begin
    XRead(tempoC,Command);
    readln(tempoC);
    while not EOF(tempoC) do
    begin
      j := j + 1;
      readln(tempoC)
    end;
    j := j + 1;
    Close(tempoC);
    j := j div 4
  end; { of NumberOfData }

begin { FinishPlot }
  if Command = 'QUIT' then
    exit
  else if not Look then
  begin
    jj := 0;
    j := 0;
    grapherCONST;
    with PFI do { plot contents from data file }
    if Command <> 'CALLHELP' then
    begin
      { Get name of data file & number of y columns }
      if j = 0 then
        NumberOfData;
      yI := 1;
      XRead(tempoC,Command);
      for yT := yI to yF do
      begin
        CP := SolidFill; { curve pattern   }
CT := NormWidth; { curve thickness }
{ curve style:CS; curve colour:CC  }
{ curve name size: CNS }
case yT of
  1: begin
       CS := solidLn;
       CC := 14;
       CNS := 6
     end;
  2: begin
       CS := dottedLn;
       CC := 13;
       CNS := 5
     end;
  3: begin
       CS := dashedLn ;
       CC := 12;
       CNS := 7
     end;
  4: CC := 10;
  5: CC := 15;
end;
if yT > 3 then
  CNS := 7; { curve name size: CNS }
SetLineStyle(CS,CP,CT);
if yT = 4 then
  SetLineStyle(UserBitLn,$3333, CT);
if yT >= 5 then
  SetLineStyle(UserBitLn,$AAAA, CT);
SetColor(CC);

CND := HorizDir;    { curve name direction: CND }
CNF := SmallFont;   { CNF means curve name font }
CNC := CC; { curve name color  = curve color CC }

Reset(tempoC);

if Command = 'INI_SPCG' then
  readln(tempoC, InFile);

{ take (xa1,ya1) as 1st data point }
Read(tempoC,xa1);
for i := 1 to yT do
read(tempoC,ya1);
readln(tempoC);

{ get and plot the rest of the points }
while not EOF(tempoC) do { Read remaining data set }
begin
  read(tempoC,xa2);
  for i  :=  1 to yT do
  read(tempoC,ya2);
  readln(tempoC);
  the_MEAT_of_GRAPHER; { join (xa1,ya1) (xa2,ya2) }
  xa1 := xa2;
  ya1 := ya2; { Last = 1st for next plotting }
  jj := jj+1;
  if Command = 'TestGraf' then
    str(yT,CN)
  else if command = 'INI_SPCG' then
  begin
    case yT of
```

```
                1: CN := 'INI_GAIN';
                2: CN := 'SPC_GAIN'
              end
            end
            else if Command = 'OPT_SPCG' then
            begin
              case yT of
                1: CN := 'OPT_GAIN';
                2: CN := 'SPC_GAIN'
              end
            end
            else if Command = 'FIS_GAIN' then
            begin
              case yT of
                1: CN := 'OPT_GAIN';
                2: CN := 'INI_GAIN';
                3: CN := 'SPC_GAIN'
              end
            end
            else
              CN := Command;
            if yT < 4 then
            begin
              if jj = j*yT then
              begin
                CNx := xa1;
                CNy := ya1;
                plot_ID; { write curve name }
              end
            end
            else
              if jj=j*(YT-3) then
                begin
                  CNx := xa1;
                  CNy := ya1-0.0125*ya1;
                  plot_ID  { write the curve name }
                end
          end;  { of current data column }
          delay(500);
          jj := 0
        end   { of the If beginning }
      end;   { of the With statement }
      if Command < > 'CALLHELP' then
        Close(tempoC);

      If (PFI.XRL-PFI.XRH) = 0 then
         OutTextXY(-80 + GetMaxX div 2, GetmaxY div 2,
           'Invalid x-axis values');
      If (PFI.YRL-PFI.YRH) = 0 then
         OutTextXY(-80 + GetMaxX div 2, 20 + GetMaxY div 2,
           'Invalid y-axis values');
      ch := readkey  { display results until a key is pressed }
    end
  end; { of FinishPlot }

BEGIN { PROGRAM GRAPHER }
  initVideo;
  clrscr;
  GotoXY(1,hi(windmax));
  LastCommand := '';
  GRAPHERmessage;
  Repeat
    ch := '$';
    repeat
      StartPlot;
      FinishPlot;
      Textmode(LastMode);
      InitVideo;
      clrscr;
      GotoXY(1,hi(windmax));
      write('Exit, Y/N ? ');
      repeat
        ch := UpCase(readkey)
      until ch in ['Y', 'N'];
      writeln(ch);
      if ch = 'N' then
        LastCommand := Command;
      for i := 1 to 15 do
      writeln
    until ch = 'Y';
    if ch = 'Y' then
      Command := 'Quit'
  Until Command = 'Quit';
END. { of the whole PROGRAM GRAPHER. }
```

```
PROGRAM FINAL;
USES CRT, GRAPH;
var
   d,m,j: Integer;

BEGIN
  DetectGraph(d,m);
  Initgraph(d,m,'c:\TP5');
  Cleardevice;
  j := GetGraphMode;
  SetUserCharSize(2, 1, 2, 1);
  SetTextStyle(1, 0, UserCharSize);
  if j = 2 then
    SetCOLOR(2);
  OutTextXY(GetMaxX div 2 - 270, GetMaxY DIV 2 - 52,
    'Exiting ...');
  if j = 2 then
    SetColor(12);
  Line(GetMaxX div 2 - 270, GetMaxY DIV 2 - 47,
      GetMaxX div 2 - 270 + 370, GetMaxY DIV 2- 47);
  Line(GetMaxX div 2 - 270, GetMaxY DIV 2 + 22,
      GetMaxX div 2 - 270 + 370, GetMaxY DIV 2 + 22);
  Line(GetMaxX div 2 - 270, GetMaxY DIV 2 - 50,
      GetMaxX div 2 - 270 + 370, GetMaxY DIV 2 - 50);
  Line(GetMaxX div 2 - 270, GetMaxY DIV 2 + 25,
      GetMaxX div 2 - 270 + 370, GetMaxY DIV 2 + 25);
  delay(500);
  CloseGraph
END.
```

References and Bibliography

[1] Desoer, Charles A., "Optimization in Network Analysis", *IEEE Transactions on Circuit Theory,* vol. CT-12, pp. 28-36, March 1965.

[2] Kuo, Franklin F., and Magnuson, Jr., Waldo G., (editors), *COMPUTER ORIENTED CIRCUIT DESIGN*: Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.

[3] Temes, Gabor C., and Calahan, Donald A., "Computer-Aided Network Optimization, the State of the Art," *Proc. IEEE*, vol. 55, pp. 1832- 1863, November 1967.

[4] Rohrer, Ronald Alan, "Fully Automated Network Design by the Digital Computer: Preliminary Considerations", *Proc. IEEE*, vol. 55, pp. 1929-1939, November 1967.

[5] Director, Stephen W., "Survey of Circuit Oriented Optimization Techniques," *IEEE Transactions on Circuit Theory,* vol. CT-18, , pp. 3-9, January 1971.

[6] Kuo, F.F., "Network Analysis by Digital Computer," *Proc. IEEE*, vol. 54, pp. 820-829, June 1966.

[7] Bandler, John W., "Optimization Methods for computer-Aided Design," *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-17, pp. 533-552, August 1969.

[8] Calahan, Donald A., *COMPUTER-AIDED NETWORK DESIGN*: McGraw-Hill, Inc., New York, 1972.

[9] Spence, R., *COMPUTER-AIDED CIRCUIT DESIGN*, European Conference on Circuit Design and Theory (editors), pp.221-267, Lausanne, 1978.

[10] Brayton, R.K., "Optimization in CACD," *ibid.*, pp. 7 - 63, Lausanne, 1978.

[11] Seshu, Sundram, and Reed, Myril B., *LINEAR GRAPHS AND ELECTRICAL NETWORKS*, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1961.

[12] Maxwell, Lee M, and Reed, Myril B., *THE THEORY OF GRAPHS: A BASIS FOR NETWORK THEORY*: Pergamon Press, New York, 1971.

[13] Swamy, M.N.S, and Thulasiraman, K., *GRAPHS, NETWORKS, AND ALGORITHMS*: John Wiley and sons, Inc., New York, 1981.

[14] Desoer, Charles A., and Kuh, E. S., *BASIC CIRCUIT THEORY*: McGraw-Hill Book Company, New York, 1969.

[15] Guillemin, Ernst A., *INTRODUCTORY CIRCUIT THEORY*: John Wiley and Sons, Inc., New York, 1953.

[16] Aatre, Vasudev K., *NETWORK THEORY AND FILTER DESIGN*: Wiley Eastern Limited, New Delhi, 1980.

[17] Adby, P.R., *APPLIED CIRCUIT THEORY: MATRIX AND COMPUTER METHODS*: Ellis Horwood Limited, Chichester, 1980.

[18] Adby, P.R., "Tree Enumeration from the Incidence Matrix," *International Journal of Electrical Engineering Education*, vol. 24, pp.183-187, Manchester V.P. 1987.

[19] Chen, Wai-Kai, "Topological Analysis for Active Networks," *IEEE Transactions on Circuit Theory*, Vol. CT-12, No.1, pp. 85-91, March 1967.

[20] Balabanian, Norman, and Bickart, Theodore A., *ELECTRICAL NETWORK THEORY*: John Wiley & Sons, Inc., New York, 1969.

[21] Reed, Myril B., *FOUNDATION FOR ELECTRIC NETWORK THEORY*: Prentice-Hall Inc., Englewood Cliffs, New Jersy, 1961.

[22] Nagel, Laurence W., *SPICE2: A COMPUTER PROGRAM TO SIMULATE SEMICONDUCTOR CIRCUITS*: Memorandum No. UCB/ERL M520, College of Engineering, University of California, Berkley, 1975.

[23] Vlach, J., Singhal, K., Vlach, M., Chada, R., and Barby, J., "Computer Methods in the Analysis and Design of Networks," *IEEE 1983 International symposium on Circuits and Systems*, Vol. 1, pp. 418 - 425.

[24] Lee, K.J., and Park, S.B., "The Reduced Modified Nodal Approach to Network Analysis," *IEEE 1983 International symposium on Circuits and Systems*, Vol. 2, pp. 637 - 640.

[25] Hatchel, G.D., Brayton, R.K., Gustavson, F.G., "The Sparse Tableau Approach to Network Analysis and Design," *IEEE Transaction on Circuit Theory(Special Issue on Computer-Aided Circuit Design)*, Vol. CT-8, pp. 101 - 113, January 1971.

[26] Weeks, W.T., Jimenez, A.J., Mahoney, G.W., Mehta, D., QassemZadeh, H., "Algorithms for ASTAP - A Network Analysis Program," *IEEE Transaction on Circuit Theory*, vol. CT-20, No. 6, November 1973.

[27] Temes, Gabor C., and LaPatra, Jack W., *INTRODUCTION TO CIRCUIT DESIGN AND SYNTHESIS*: McGraw-Hill Book Company, New York, 1977.

[28] Director, Stephan .W., *CIRCUIT THEORY: A COMPUTATIONAL APPROACH*: Wiley, New York, 1975.

[29] Gupta, K.C., Garg, Ramesh, Chadha, Rakesh, *COMPUTER-AIDED DESIGN OF MICROWAVE CIRCUITS*: Artech House , Inc., Dedham, Massachusetts 02026, 1981.

[30] Calahan, D. A., "Computer Design of Linear Frequency Selective Networks," *Proc. IEEE*, vol. 53, pp.1701-1706, November 1965.

[31] Director, Stephan W., and Rohrer, Ronald A., "Automated Network Design- the Frequency Domain Case," *IEEE Transaction on Circuit Theory*, vol. CT-16, pp.330-337, August 1969.

[32] Director, Stephan W., "LU Factorization in Network Sensitivity Calculations," *IEEE Transaction on Circuit Theory*, vol. CT-18, pp.184-185, January 1971.

[33] Parker, S. R., "Sensitivity: Old Questions Some New Answers," *IEEE Transaction on Circuit Theory*, vol. CT-18, pp. 27-34, January 1971.

[34] Director, Stephen W., and Rohrer, Ronald R.,"The Generalized Adjoint Network and Network Sensitivities,": *IEEE Transaction on Circuit Theory*, vol. CT-16, pp. 330-336, August 1969.

[35] Temes, Gabor C., and Zai, D. Y. F., "Least pth Approximation,": *IEEE Transaction on Circuit Theory*, vol. CT-16, pp. 235-237, May 1969.

[36] Wing, Omar, and Behar, Jaime V., "Circuit Design by Minimization using the Hessian Matrix,": *IEEE Transaction on Circuits and Systems*, vol. CAS-21, No. 5, pp. 643-649, September 1974.

[37] Gyurcsik, R. S., Mayaram, K., Yee, T., Ma, F., and Pederson, Donald O., "Language Selection for Circuit Simulation," *IEEE 1984 International symposium on Circuits and Systems*, vol. 2, pp. 527 - 529, 1984.

[38] Vlach, M., Vlach, J., Singhal, K., Chadha, R., and Christen, E., "WATSCAD - A Program for the Analysis and Design of Switched Capacitor Netwworks," *IEEE 1985 International symposium on Circuits and Systems*, vol. 3, pp. 1177 - 1178, 1985.

[39] Christen, E., Vlach, J., Singhal, K., and Vlach, M., "A New Program for Analysis and Optimization of Passive and Active RC Networks," *IEEE 1986 International symposium on Circuits and Systems*, vol. 3, pp. 972 - 975, 1986.

[40] Chadha, Rakesh, Singhal, Kishore, and Vlach, Jiri, "WATOPT - A New Optimizer for Circuit Applications," *IEEE 1983 International symposium on Circuits and Systems*, vol. 3, pp. 1046 - 10495, 1983.

[41] Dale, Nell, and Lilly, Susan C., *PASCAL plus DATA STRUCTURES, ALGORITHMS, and ADVANCED PROGRAMMING*: Heath and Company, Lexington, Massachsetts, D. C., 1985.

[42] Agnew, David, "Efficient use of the Hessian Matrix for Circuit Optimization," *IEEE Transactions on Circuits and Systems*, vol. CAS 25, No. 8, August 1978.

[43] Cuthbert, Jr., Thomas R., *OPTIMIZATION USING PERSONAL COMPUTERS*: John Wiley and Sons, Inc., New York, 1987.

[44] Bown, G. C. S., and Geiger, G. V., "Design and Optimization of Circuits by Computer," *Proceedings of IEEE*, vol. 118, May 1971, pp. 649 - 661.

[45] Gill, Philip E., Murray, Walter, Wright, Margaret H., *PRACTICAL OPTIMIZATION*: Academic Press, Inc., London, 1981.

[46] Adby, P. R., and Dempster, M. A. H., *INTRODUCTION TO OPTIMIZATION METHODS*: Chapman and Hall Ltd., London, 1974.

[47] Beightler, Charles S., Philips, Don J., Wilde, Douglass J., *FOUNDATIONS OF OPTIMIZATION*: Prentice-Hall, Inc., Englewood Cliffs, N. J., 1979.

[48] Girma Mullissa, *INTRODUCTION TO OPTIMIZATION*: Addis Ababa University, Addis Ababa, 1979.

[49] Gottfreid, Byron S., and Weisman, Joel, *INTRODUCTION TO OPTIMIZATION*: Prentice-Hall, Inc., Englewood cliffs, New Jersey, 1973.

[50] Lasdon, Leon S., *OPTIMIZATION THEORY FOR LARGE SYSTEMS*: McMillan Publishing Co., Inc., New York, 1970.

[51] Aaron, M. Robert, "The Use of Least Squares for System Design," *IRE Transactions on Circuit Theory*, vol CT-3, December 1956, pp. 224 - 231.

[52] Fletcher, R., *PRACTICAL METHODS OF OPTIMIZATION*: John Wiley and Sons, Chichester, 1980.

[53] Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B., *COMPUTER METHODS FOR MATHEMATICAL COMPUTATIONS*: Prentice-Hall, Inc., Englewood Cliffs, New Jersy, 1977.

[54] Walsh, G. R., *METHODS OF OPTIMIZATION*: John Wiley and Sons Ltd., Chichester, 1979.

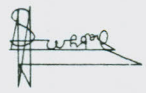
[55] Faulkenberry, Luces M., *AN INTRODUCTION TO OPERATIONAL AMPLIFIERS*: John Wiley and Sons, New York, 1977.

[56] Sieppel, Robert G., *OPERATIONAL AMPLIFIERS*: Reston Publishing Company, Inc., Reston, Virginia, 1983.

[57] Guillemin, Ernst A., *SYNTHESIS OF PASSIVE NETWORKS*: John Wiley and Sons, Inc., New York, 1957.

Signed Declaration

I hereby declare that this thesis is my work and all sources of material used for it have been duly acknowledged.



Solomon Zewde,

Addis Ababa
June 1993