A NOVEL APPROACH TO CALCULATING RELATIVE SCATTERING PARAMETER SENSITIVITY IN COMPUTER–AIDED DESIGN PROGRAMS

by

Dane E. Blackburn

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

William A. Davis, Chairman	Charles W. Bostian
John C. McKeeman	Vatché Vorperian

June, 1988

Blacksburg, Virginia

A NOVEL APPROACH TO CALCULATING RELATIVE SCATTERING PARAMETER SENSITIVITY IN COMPUTER–AIDED DESIGN PROGRAMS

by

Dane E. Blackburn

Committee Chairman: W. A. Davis

Electrical Engineering

(ABSTRACT)

Relative sensitivity is a measure of the percentage change in a system parameter caused by a percentage change in a component parameter. The adjoint network method has previously been used by Monaco and Tiberio in the computation of relative scattering parameter sensitivity. A new approach is presented in this work which defines a bilinear equation and three constants that relate any component scattering parameter to any system scattering parameter. A computer–aided design program which implements this relative sensitivity in analysis and optimization is presented. Circuit analysis examples demonstrating sensitivity analysis and optimization are included. As a background for this work, computer–aided design concepts, such as network modeling, objective functions, Rosenbrock's optimization method, and the adjoint network method for estimating partial derivatives, are also presented.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. William Davis for his encouragement and patience, and for introducing me to the elegance of S–parameters. Also, I want to thank Charles Bostian, John McKeeman, and Vatche Vorperian for their advice, and for being on my committee.

There are many people who have made my six year stay here at Virginia Tech rewarding and enjoyable, and I want to name a few of them. First, thanks to the ladies of the third floor for putting up with me and always doing their best, namely Loretta, Pat, Lorri, Alice, and Robin. Thanks to the Fiber Optics Group and the Satellite Communications Group for allowing me in their midst. The following people mean an awful lot to me: my daredevil brother Rick, my old buddies Jeff, Darrell, Dan, and Kevin, Backseat Drivers Mike, Ken, Paul, and Bruce, and my beer–golfing buddies Dave, Dave, and Ko. Most of all I want to thank my mother for her immeasurable understanding and support.

TABLE OF CONTENTS

1. INTRODUCTION

This thesis presents a new method for calculating network sensitivity. An equation is derived that relates a system scattering parameter to any individual component scattering parameter. This equation is then used to derive an expression for relative sensitivity in terms of scattering parameters. Calculating the coefficients of the equation are very time consuming if computer analysis is not employed; therefore, computer–aided design (CAD) concepts are discussed. To allow for comparison of methods, we will discuss an established method for computing sensitivity in terms of scattering parameters. Since it is often desireable to optimize a circuit by minimizing a particular sensitivity, some popular optimization methods are also discussed. Although the computer–aided design concepts covered in this thesis apply to any frequency range, we will concentrate on the problems faced by microwave circuit designers.

At microwave frequencies, sensitivity becomes an important issue. Sensitivity is a measurement of a percentage change in a system response caused by a percentage change in a component parameter. For instance, we may wish to find the sensitivity of the overall gain of an amplifier to the length of a transmission line in the amplifier. We might find that to achieve the desired gain, close tolerances must be imposed on the line, or that the line length need not be closely monitored. Sensitivity analysis becomes very important in mass production of microwave circuits. The engineer must know how precise the automatic equipment should be in order to produce an adequate circuit. In any case, computer–aided design programs help save the engineer time and money by allowing him or her to predict

how a circuit will perform before it is actually produced.

Since the 1960's, many computer–aided design programs have been developed to aid the microwave circuit designer. Some well known programs include SUPER–COMPACT (Compact Software, Inc., 1988), and TouchStone (EEsof, 1986). Examples of other programs include: PACMO (Fernandes, 1983)— a specialized program for designing microwave filters, couplers, and power dividers, MCAP (Gupta, 1981)— a general microwave circuit analysis program that computes the overall scattering parameters for a multiport network, and CADMIC (Sanchez–Sinencio, 1974)— a more versatile program that includes frequency domain analysis, and optimization of transducer power gain, reflection coefficient, noise figure, and/or sensitivity. In general, a good computer–aided design program should accept circuit topology information, and provide circuit analysis and optimization capabilities.

Microwave computer–aided design programs are similar to general network computer–aided design programs, but allow for microwave components and compensate for high frequency effects. At least three types of components are needed in microwave circuit development: 1) lumped elements (resistors, capacitors, and inductors) 2) distributed elements (such as transmission lines, or waveguides) and 3) black box elements (such as transistors described only by their two–port parameters). Circuit models are used by some programs to describe active devices; however, models of transistors at high frequencies are not always adequate. In this work, we assume that the measured device two–port scattering parameters are available for the frequencies of interest. Basic discussion and examples of microwave computer–aided design have been presented by Bradley (1983).

In addition to basic circuit analysis, computer–aided design programs allow the user to optimize a circuit in order to achieve a desired overall response. For

example, the designer may wish to maximize the forward power gain, or minimize a particular sensitivity. Optimization is usually accomplished through minimization of a user–defined objective function. For instance, minimizing a function equal to the inverse of the forward gain will have the effect of maximizing the forward gain. Optimization routines vary circuit component values within limits, evaluate the objective function, and halt when a minimum is found in the objective function.

Many of the methods used in minimizing the objective function were described by Bandler (1969). Bandler (1970) has also described how to use the popular adjoint network method (Director, 1969) in minimization routines. Rosenbrock's method (Rosenbrock, 1960) has been used in the design of broadband amplifiers (Trick, 1970) and has been incorporated into the program developed for this thesis. Other optimization methods have been discussed by Bandler (1985, 1969) and Charalambous (1985). These methods provide improvements on older methods in terms of computation time. Also, Bandler has used the adjoint network concept in computer analysis of cascaded networks with any possible even number of response ports (Bandler, 1980). A survey of optimization techniques used in frequency domain analysis has been presented by Monaco (1974). More recently, an overview of optimization methods used in microwave circuit design has been presented by Gupta (1981).

Using the adjoint network, Bandler has described a procedure for computing first order sensitivities (Bandler, 1970). First order sensitivity expressions have been derived (Bandler, 1970; Sanchez–Sinencio, 1974). These are expressions for the partial derivative of a system response with respect to a network parameter given in terms of responses computed using the adjoint network method. Also, the adjoint network procedure has been extended for use in the computation of second order sensitivities (Richards, 1969; Monaco, 1974). Second order sensitivities are

system response partials found with respect to two independent network parameters. Sensitivities have generally been expressions of a change in a response voltage or current with respect to a change in some internal network parameter; however, sensitivities have also been defined in terms of scattering parameters (Monaco, 1970; Iuculano, 1971). Monaco and Tiberio have presented methods for finding first and second order sensitivities using scattering parameters in computations (Monaco, 1974).

This thesis introduces a unique method for computing sensitivity. Three constants and one equation are found that relate a system scattering parameter to any particular component (internal) scattering parameter. The derivation of this equation is found in Chapter 4. The developed program allows for the calculation of sensitivity of a system scattering parameter to an internal device scattering parameter. The program also allows the user to minimize the magnitude of this relative sensitivity. Thus, the program enables a circuit designer to incorporate sensitivity in addition to gain and mismatch considerations into his design.

Chapter 2 reviews scattering parameters and computer–aided design concepts. Equations for finding the resultant scattering parameters for various two–port combinations are developed, and several circuit analysis examples are included. Computer based optimization methods are introduced. Chapter 3 covers the objective function used in optimization, and Rosenbrock's minimization procedure. Examples of optimizied circuits are presented at the end of the chapter.

Monaco and Tiberio's adjoint network approach to scattering parameter sensitivity (Monaco, 1970) is discussed in Chapter 4. The new approach to defining scattering parameter sensitivity is presented along with circuit design examples including sensitivity analysis. In Chapter 5, possible improvements to the program are suggested along with a summary of this work.

An updated version of a computer–aided design program developed by W. F. Duke (1977) is included in Appendix C. This program (SOPT) utilizes the optimization and sensitivity concepts discussed in Chapters 2 through 4.

2. COMPUTER–AIDED DESIGN CONCEPTS

In this thesis, scattering parameters (S–parameters) are used in the modeling, optimization, and sensitivity analysis of networks; therefore, a review of scattering parameters is presented in this chapter. Circuit modeling techniques involving combinations of one– and two–ports are discussed along with basic optimization concepts.

2.1 Scattering Parameters

Scattering parameters provide a convenient description for two–port networks (Montgomery, 1948; Kurokawa, 1965). These parameters are particularly useful in describing microwave circuits, since they are readily measured using traveling waves at microwave frequencies, and are directly related to quantities of interest, such as power reflection and power gain. Also, signal flow graph concepts are easily applied to S–parameter analysis. A signal flow graph representation of a two–port is shown in Fig. 2.1–1. We will be referring to the normalized power wave variables a_n and b_n defined by

$$a_n = \frac{V_{in}}{\sqrt{Z_{on}}} \qquad (2.1\text{–}1a)$$

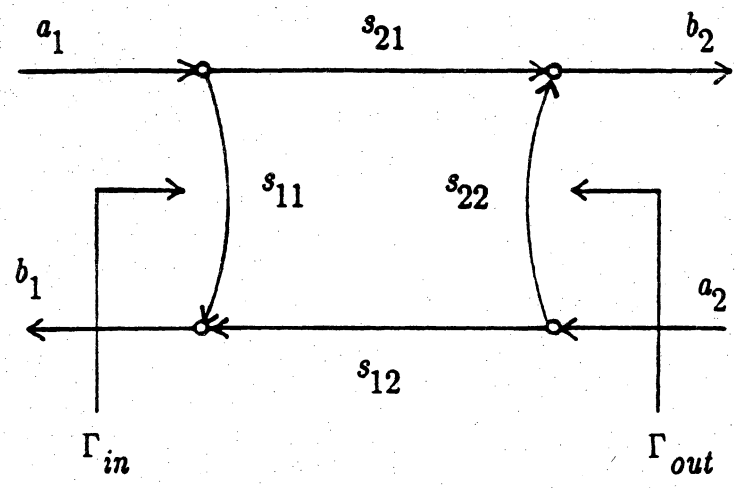


Figure 2.1–1 Two–port flow diagram.

and

$$b_n = \frac{V_{rn}}{\sqrt{Z_{on}}} , \qquad (2.1\text{–}1b)$$

where a_n and b_n are the variables at port n, V_{in} and V_{rn} are the incident and reflected voltage waves at port n, and Z_{on} is the reference impedance of port n. If it is assumed that V_{in} is an rms value, the incident power at port n is given by

$$P_{inc} = \left| a_n \right|^2 . \qquad (2.1\text{–}2)$$

The S–parameters for a two–port network (see Fig. 2.1–1) are then given by the equations

$$b_1 = s_{11}a_1 + s_{12}a_2 \qquad (2.1\text{–}3a)$$

$$b_2 = s_{21}a_1 + s_{22}a_2 , \qquad (2.1\text{–}3b)$$

where $b_{1(2)}$ is the reflected power wave at port 1(2), and $a_{1(2)}$ is the incident power wave at port 1(2). The parameter s_{11} is simply the input reflection coefficient Γ_{in} obtained when the output port is terminated with its reference impedance. Thus s_{11} is given by

$$s_{11} = \left. \frac{b_1}{a_1} \right|_{(a_2=0)} = \Gamma_{in} = \frac{Z - Z_{o1}}{Z + Z_{o1}} , \qquad (2.1\text{–}4)$$

where Z is the complex impedance seen looking into port 1, and Z_{o1} is the reference

impedance of port 1. Note that $a_n = 0$ is achieved by terminating port n with its reference impedance. Likewise, s_{22} is the output reflection coefficient obtained when the input port is terminated with its reference impedance. The forward transmission parameter s_{21} is given by the equation

$$s_{21} = \left.\frac{b_2}{a_1}\right|_{(a_2=0)} = \frac{2\sqrt{Z_{o1}}\ V_2}{\sqrt{Z_{o2}}\ E_1}, \qquad (2.1\text{–}5)$$

where V_2 is the total voltage at port 2, E_1 is a voltage applied to port 1 (see Fig. 2.1–2a), and $a_2 = 0$ is obtained by terminating port 2 with its reference impedance Z_{o2}. Similarly, s_{12} is given by

$$s_{12} = \left.\frac{b_1}{a_2}\right|_{(a_1=0)} = \frac{2\sqrt{Z_{o2}}\ V_1}{\sqrt{Z_{o1}}\ E_2}, \qquad (2.1\text{–}6)$$

where V_1 is the total voltage at port 1, E_2 is a voltage applied to port 2 (see Fig. 2.1–2b), and $a_1 = 0$ is obtained by terminating port 1 with its reference impedance Z_{o1}.

In general, S–parameters are complex numbers expressed in magnitude and phase, and defined with respect to reference impedances. The ports of a microwave network usually have a reference impedance of 50 ohms. Obviously, $Z_{o1} = Z_{o2}$ will simplify the s_{21} and s_{12} equations presented above. From this point on, we will assume that we are dealing with networks with only one reference impedance Z_{ref}.

In addition to being a convenient description of a two–port, S–parameters

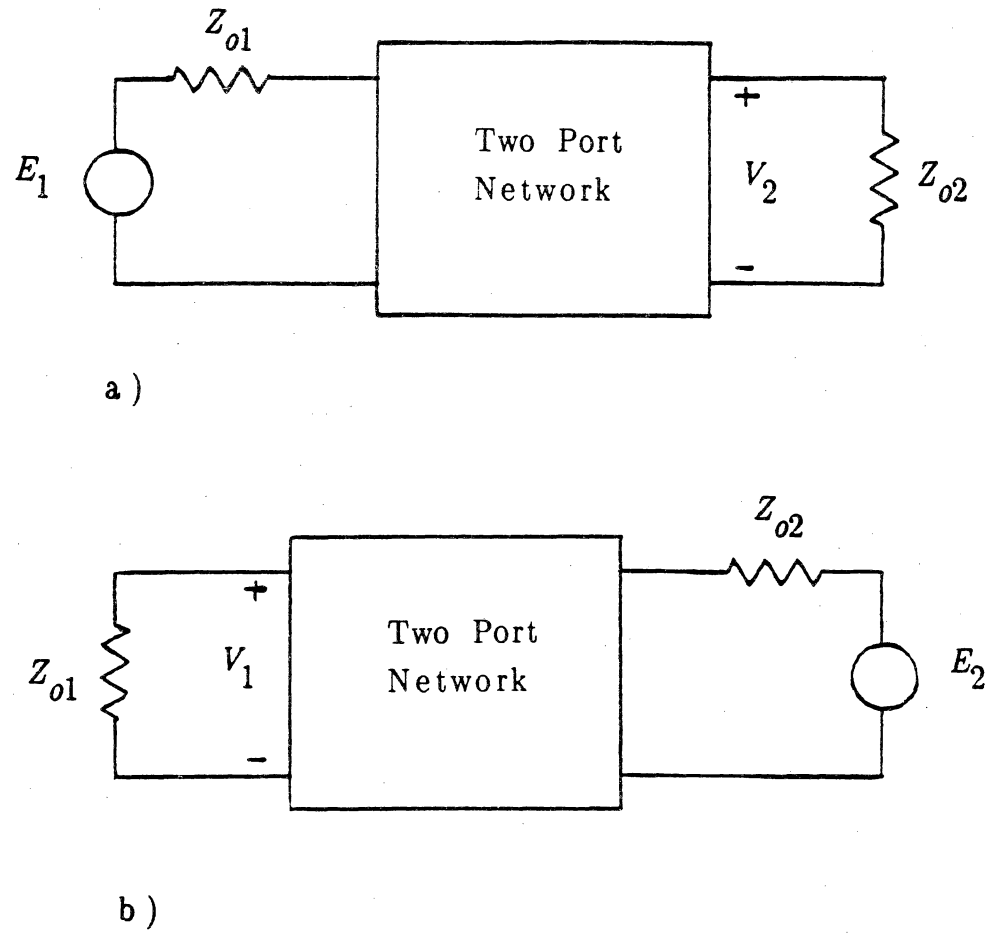


Figure 2.1–2 Test set–ups for finding (a) s_{21} and (b) s_{12}.

provide useful information. For example, $|s_{11}|^2$ gives the ratio of the power reflected from port 1 to the power available at port 1. Also, $|s_{21}|^2$ gives the ratio of the power delivered at port 2 to the power available at port 1, commonly referred to as transducer power gain.

Besides finding power ratios, S–parameters are also used in the equations and procedures of microwave amplifier design. Gonzalez (1984) and Ha (1981) have extensively discussed S–parameters and their use in microwave amplifier design. In this thesis, we will be utilizing computer–aided design techniques rather than the analytical techniques of Gonzalez and Ha. The optimization examples in Chapters 3 and 4 will involve designing microwave amplifiers with the aid of the computer program developed for this thesis.

2.2 Network Modeling

A network with lumped parameter components, transmission lines, and two–port devices may be thought of as a combination of two–ports. Many microwave networks such as amplifiers or filters are two–ports; therefore, it is desirable to compute the overall scattering parameters of the network when the component values and network topology are known. The first step in computing the overall scattering parameters of a network is to determine the parameters associated with each component. Components that we consider include: resistors, capacitors, inductors, transmission lines, and other specified devices. For the specified devices, we must provide the two–port parameters at the frequencies of interest. For the other components, the scattering parameters are determined from theoretical models.

To calculate the parameters associated with a resistor, capacitor, or inductor,

we must specify whether the component is used as a shunt element or a series element (see Fig. 2.2–1). The S–parameters for a shunt element are given by

$$s_{11} = s_{22} = \frac{-1}{2z + 1} \tag{2.2–1a}$$

$$s_{12} = s_{21} = \frac{2z}{2z + 1} \tag{2.2–1b}$$

and the S–parameters for a series element are given by

$$s_{11} = s_{22} = \frac{z}{z + 2} \tag{2.2–2a}$$

$$s_{12} = s_{21} = \frac{2}{z + 2}\ , \tag{2.2–2b}$$

where z is the normalized impedance $z = Z / Z_{ref}$, Z is the complex impedance of the element, and Z_{ref} is the reference impedance of the network.

It can be shown that the S–parameters for a lossless transmission line are given by (Grivet, 1976)

$$s_{ii} = \frac{j(Z^2 - Z_{ref}^2)\sin(\beta l)}{2ZZ_{ref}\cos(\beta l) + j(Z^2 + Z_{ref}^2)\sin(\beta l)} \tag{2.2–3a}$$

$$s_{ik} = \frac{2ZZ_{ref}}{2ZZ_{ref}\cos(\beta l) + j(Z^2 + Z_{ref}^2)\sin(\beta l)}\ , \tag{2.2–3b}$$

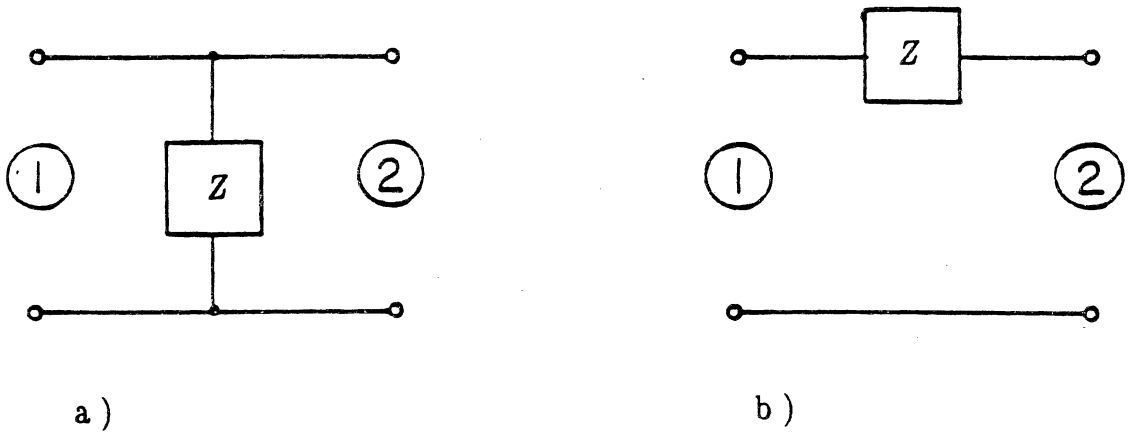


Figure 2.2–1 (a) Shunt and (b) series connections for a passive component.

where Z is the characteristic impedance of the line, Z_{ref} is the reference impedance of the network, β is the propagation constant, and l is the length of the line. The propagation constant β is given by

$$\beta = \frac{2\pi f}{v} , \qquad (2.2\text{–}4)$$

where f is the frequency of operation, and v is the velocity of a wave on the transmission line. The form of (2.2–3) is also applicable to single mode waveguides and related guiding structures with v denoting the mode phase velocity. Although lossy lines are not considered in this work, the form of (2.2–3) could be applicable to lossy lines by replacing β with the complex wave constant γ , and the sin and cos functions with sinh and cosh functions (Ha, 1981).

Once their scattering parameters are known, any pair of two–ports may be combined to form a single two–port. We may combine a pair of two–ports in one of seven combination formats. The seven formats are named and demonstrated in Fig. 2.2–2 . For the Z, Y, H, and G combination formats, the S–parameter matrices of the modules involved are first converted to normalized parameter matrices of the required format type. The parameter matrices are then added, and the resulting format parameter matrix is converted back to an S–parameter matrix. For example, suppose we wish to combine two modules in the series connection format of Fig. 2.2–2a. First, the S–parameters of each module would be converted to z–parameters. Then the z–parameters would be added, and the resulting z–parameters converted to S–parameters. Conversion relations between the S, z, y, h, and g parameters are shown in Table 2.2–1 .

A common combination format is the cascade connection of a pair of

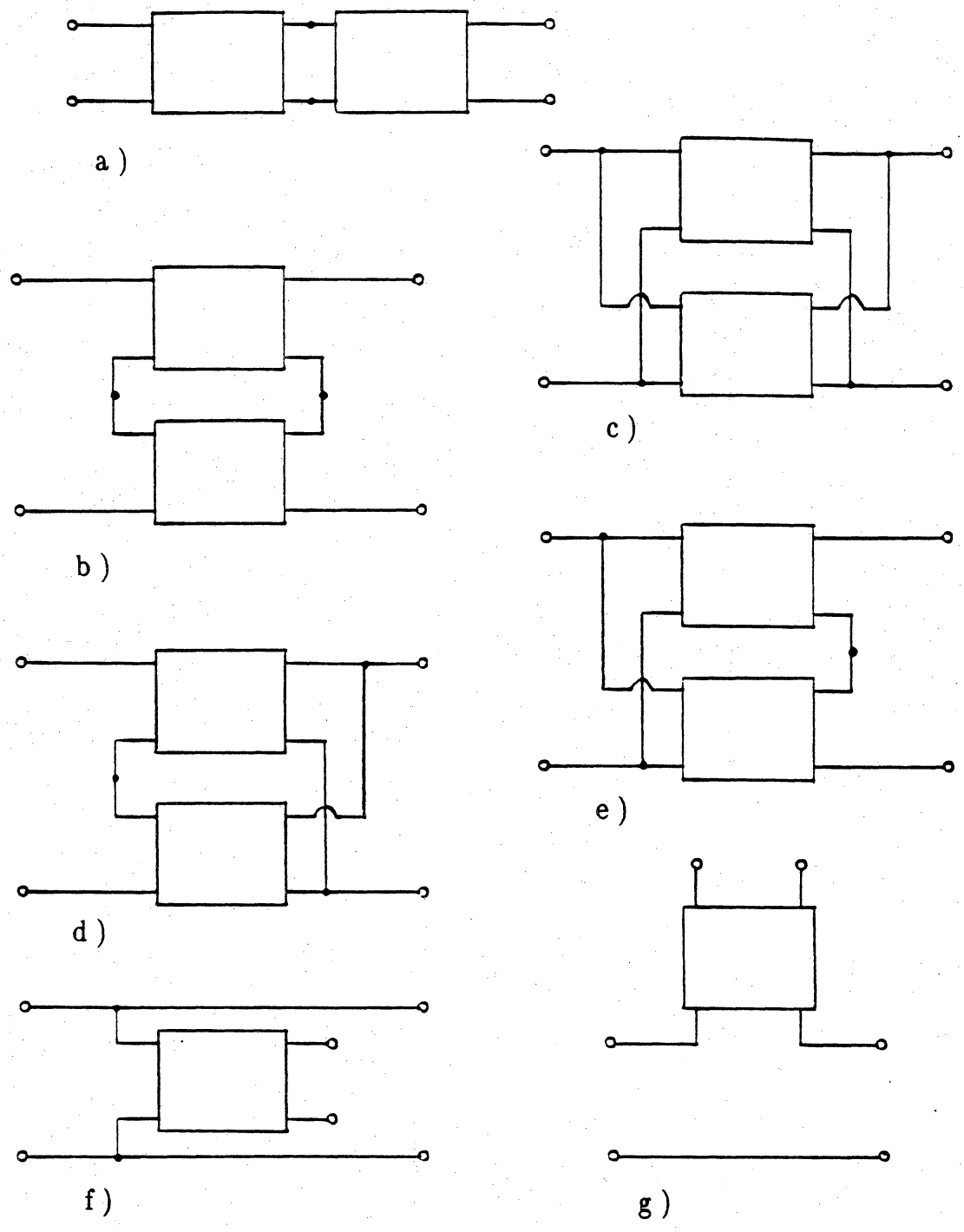


Figure 2.2–2 Two–port module combinations (a) Cascade, (b) Z (series), (c) Y (shunt), (d) H (series–shunt), (e) G (shunt–series), (f) two–port connected as a shunt element, and (g) two–port connected as a series element.

Table 2.2–1 Conversion relations between the S, z, y, h, and g parameters.

S ⇒ z

$$z_{11} = \frac{(1 + s_{11})(1 - s_{22}) + s_{12}s_{21}}{\Delta_5}$$

$$z_{12} = \frac{2s_{12}}{\Delta_5}$$

$$z_{21} = \frac{2s_{21}}{\Delta_5}$$

$$z_{22} = \frac{(1 - s_{11})(1 + s_{22}) + s_{12}s_{21}}{\Delta_5}$$

S ⇒ y

$$y_{11} = \frac{(1 - s_{11})(1 + s_{22}) + s_{12}s_{21}}{\Delta_6}$$

$$y_{12} = \frac{-2s_{12}}{\Delta_6}$$

$$y_{21} = \frac{-2s_{21}}{\Delta_6}$$

$$y_{22} = \frac{(1 + s_{11})(1 - s_{22}) + s_{12}s_{21}}{\Delta_6}$$

S ⇒ h

$$h_{11} = \frac{(1 + s_{11})(1 + s_{22}) - s_{12}s_{21}}{\Delta_7}$$

$$h_{12} = \frac{2s_{12}}{\Delta_7}$$

$$h_{21} = \frac{-2s_{21}}{\Delta_7}$$

$$h_{22} = \frac{(1 - s_{11})(1 - s_{22}) - s_{12}s_{21}}{\Delta_7}$$

S ⇒ g

$$g_{11} = \frac{(1 - s_{11})(1 - s_{22}) - s_{12}s_{21}}{\Delta_8}$$

$$g_{12} = \frac{-2s_{12}}{\Delta_8}$$

$$g_{21} = \frac{2s_{21}}{\Delta_8}$$

$$g_{22} = \frac{(1 + s_{11})(1 + s_{22}) - s_{12}s_{21}}{\Delta_8}$$

Table 2.2–1 (Continued)

z ⇒ S

$$s_{11} = \frac{(z_{11} - 1)(z_{22} + 1) - z_{12}z_{21}}{\Delta_1}$$

$$s_{12} = \frac{2z_{12}}{\Delta_1}$$

$$s_{21} = \frac{2z_{21}}{\Delta_1}$$

$$s_{22} = \frac{(z_{11} + 1)(z_{22} - 1) - z_{12}z_{21}}{\Delta_1}$$

y ⇒ S

$$s_{11} = \frac{(1 - y_{11})(1 + y_{22}) + y_{12}y_{21}}{\Delta_2}$$

$$s_{12} = \frac{-2y_{12}}{\Delta_2}$$

$$s_{21} = \frac{-2y_{21}}{\Delta_2}$$

$$s_{22} = \frac{(1 + y_{11})(1 - y_{22}) + y_{12}y_{21}}{\Delta_2}$$

h ⇒ S

$$s_{11} = \frac{(h_{11} - 1)(h_{22} + 1) - h_{12}h_{21}}{\Delta_3}$$

$$s_{12} = \frac{2h_{12}}{\Delta_3}$$

$$s_{21} = \frac{-2h_{21}}{\Delta_3}$$

$$s_{22} = \frac{(1 + h_{11})(1 - h_{22}) + h_{12}h_{21}}{\Delta_3}$$

g ⇒ S

$$s_{11} = \frac{(1 - g_{11})(1 + g_{22}) + g_{12}g_{21}}{\Delta_4}$$

$$s_{12} = \frac{-2g_{12}}{\Delta_4}$$

$$s_{21} = \frac{2g_{21}}{\Delta_4}$$

$$s_{22} = \frac{(g_{11} + 1)(g_{22} - 1) - g_{12}g_{21}}{\Delta_4}$$

where

$$\Delta_1 = (z_{11} + 1)(z_{22} + 1) - z_{12}z_{21}$$
$$\Delta_2 = (1 + y_{11})(1 + y_{22}) - y_{12}y_{21}$$
$$\Delta_3 = (h_{11} + 1)(h_{22} + 1) - h_{12}h_{21}$$
$$\Delta_4 = (1 + g_{11})(1 + g_{22}) - g_{12}g_{21}$$

$$\Delta_5 = (1 - s_{11})(1 - s_{22}) - s_{12}s_{21}$$
$$\Delta_6 = (1 + s_{11})(1 + s_{22}) - s_{12}s_{21}$$
$$\Delta_7 = (1 - s_{11})(1 + s_{22}) + s_{12}s_{21}$$
$$\Delta_8 = (1 + s_{11})(1 - s_{22}) + s_{12}s_{21}$$

and $z_{11} = Z_{11} / Z_{ref}$, $z_{12} = Z_{12} / Z_{ref}$, etc..

two–port modules. In computer–aided design programs, cascade connections are often handled by first converting to T, or transmission parameters, multiplying the module matrices, and then converting the resulting T matrix to an S–parameter matrix. However, a more efficient method in terms of computing time is to use the S–parameter equations for cascade connection. We can use the signal flow graph of Fig. 2.2–3 to derive the equations needed to calculate the overall S–parameters of this connection. The overall s_{11} is simply equal to the reflection coefficient Γ_{in}, with $s_{11}^{[2]}$ considered a load reflection on module 1. For a typical two–port,

$$\Gamma_{in} = \frac{s_{11} - \Gamma_L \Delta_s}{1 - s_{22}\Gamma_L} , \tag{2.2–5}$$

where Δ_s is the determinant

$$\Delta_s = s_{11}s_{22} - s_{12}s_{21} . \tag{2.2–6}$$

Therefore, with $\Gamma_L = s_{11}^{[2]}$,

$$s_{11} = \frac{s_{11}^{[1]} - s_{11}^{[2]} \Delta_s^{[1]}}{1 - s_{22}^{[1]} s_{11}^{[2]}} . \tag{2.2–7}$$

The overall s_{12} is given by the path gain from a_2 to b_1, divided by the feedback term $1 - s_{22}^{[1]} s_{11}^{[2]}$ as

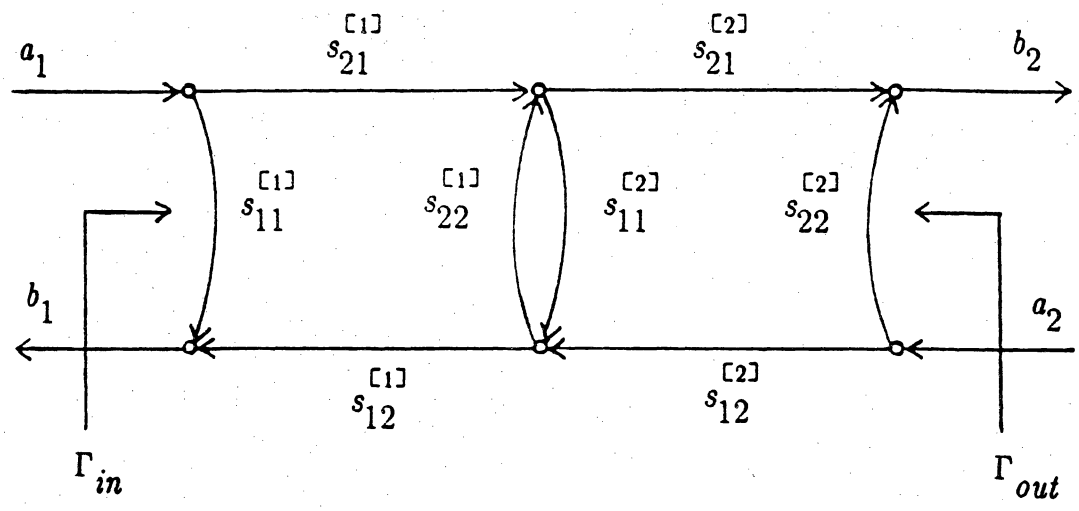


Figure 2.2–3 Signal flow graph for cascaded two–ports.

$$s_{12} = \frac{s_{12}^{[1]}\, s_{12}^{[2]}}{1 - s_{22}^{[1]}\, s_{11}^{[2]}} \,. \tag{2.2–8}$$

Similarly, s_{21} and s_{22} are given by

$$s_{21} = \frac{s_{12}^{[2]}\, s_{12}^{[1]}}{1 - s_{22}^{[1]}\, s_{11}^{[2]}} \tag{2.2–9}$$

and

$$s_{22} = \frac{s_{11}^{[2]} - s_{11}^{[1]}\, \Delta_s^{[2]}}{1 - s_{22}^{[1]}\, s_{11}^{[2]}} \,. \tag{2.2–10}$$

Finally, we might use a two–port module as either a shunt element or a series element, as in the modeling of a stub bias network. These two cases are demonstrated as combination formats f and g in Fig. 2.2–2 . In both cases, the right end of the two–port is left open, and the left end is treated as a one–port. In Fig. 2.2–4, the signal flow graph of a two–port element connected in shunt is displayed, with the parameters A_{11} , A_{12} , A_{21} , and A_{22} representing the S–parameters of the two–port element. The unity gain arrow represents the open–circuited right end of the two–port element. With the open circuit Γ equal to unity, (2.2–5) tells us that the input reflection coefficient is

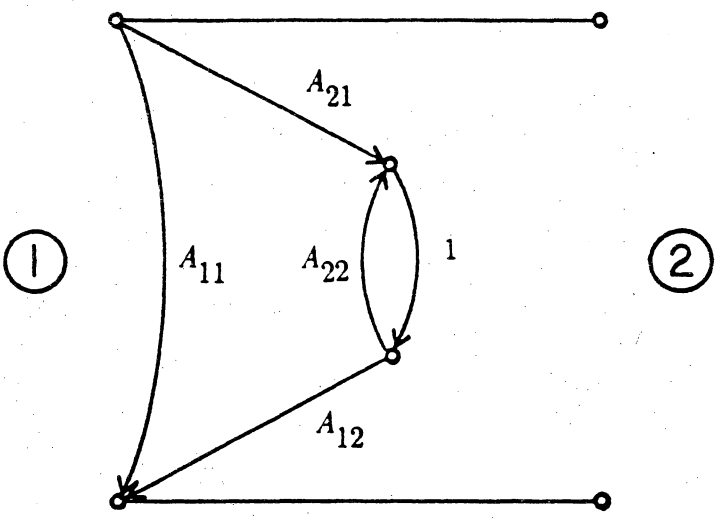


Figure 2.2–4 Flow diagram for two–port used as a shunt element.

$$\Gamma_{in} = \frac{A_{11} - \Delta_A}{1 - A_{22}} \quad . \tag{2.2–11}$$

Rearranging equation (2.1–4), we know that

$$z = \frac{1 + \Gamma_{in}}{1 - \Gamma_{in}} \quad , \tag{2.2–12}$$

where z is the normalized impedance $z = Z / Z_{ref}$. Substituting (2.2–11) into (2.2–12), we find that

$$z = \frac{1 - A_{22} + A_{11} - \Delta_A}{1 - A_{22} - A_{11} + \Delta_A} \quad . \tag{2.2–13}$$

Using (2.2–1), we obtain for the shunt connection the following expression for s_{11} and s_{22} :

$$s_{11} = s_{22} = \frac{-1 + A_{22} + A_{11} - \Delta_A}{3 - 3A_{22} + A_{11} - \Delta_A} \quad . \tag{2.2–14}$$

Also from (2.2–1), we obtain the following expression for s_{12} and s_{21} :

$$s_{12} = s_{21} = \frac{2(\ 1 - A_{22} + A_{11} - \Delta_A\ )}{3 - 3A_{22} + A_{11} - \Delta_A} \quad . \tag{2.2–15}$$

Using a similar procedure and (2.2–2), we find the S–parameters of a two–port element connected in series (see Fig. 2.2–5) to be

$$s_{11} = s_{22} = \frac{1 - A_{22} + A_{11} - \Delta_A}{3 - 3A_{22} - A_{11} + \Delta_A} \tag{2.2–16}$$

and

$$s_{12} = s_{21} = \frac{2(1 - A_{22} - A_{11} + \Delta_A)}{3 - 3A_{22} - A_{11} + \Delta_A} . \tag{2.2–17}$$

These equations have been utilized in the calculation of the overall scattering parameters of a network.

Example 2.1

The passive bandpass filter of Fig. 2.2–6 was analyzed. Values for the computed forward transmission parameter s_{21} are tabulated in Table 2.2–2 and plotted in Fig. 2.2–7 . These results are easily verified by using the center frequency and bandwidth equations for a series RLC circuit. These equations are given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}} \tag{2.2–18}$$

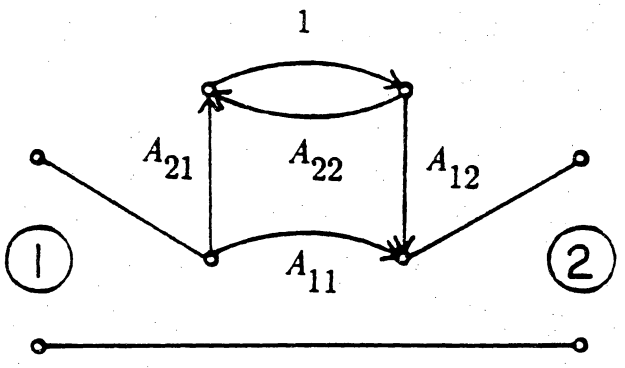


Figure 2.2–5 Flow diagram for two–port used as a series element.

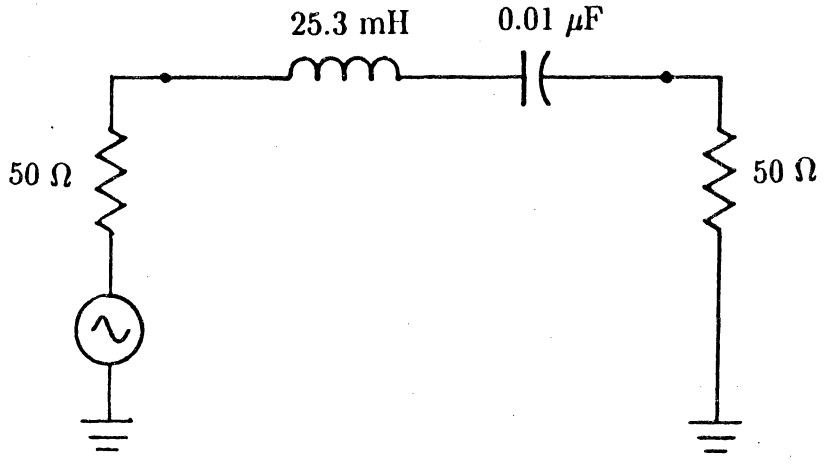


Figure 2.2–6 Passive bandpass filter.

Table 2.2–2 Tabulated s_{21} vs. f data for bandpass filter of Figure 2.2–6.

f (Hz)	s_{21} (dB)
1000	-43.95
5000	-27.57
9685	-3.17
10000	-0.0016
10314	-2.86
20000	-27.55
100000	-43.94

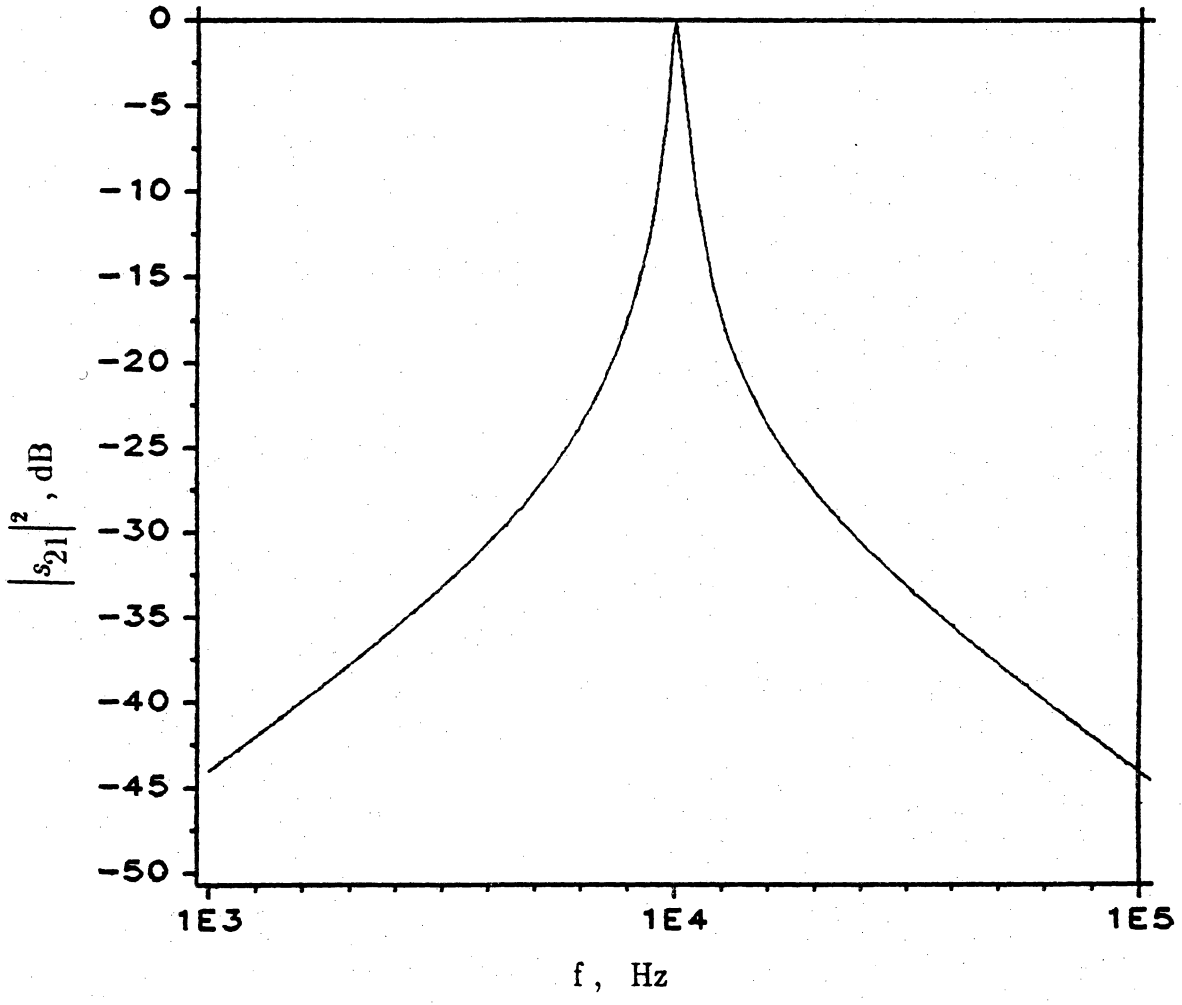


Figure 2.2–7 Plot of s_{21} vs. f data for bandpass filter of Figure 2.2–6.

and

$$BW = R / (2\pi L) \ , \qquad (2.2\text{–}19)$$

where f_o is the center frequency in Hertz, BW is the bandwidth, and R is given by $2Z_{ref}$ (accounting for source loading). If Z_{ref} is 50 ohms, (2.2–18) gives a center frequency of 10 kHz and (2.2–19) gives a bandwith of 629 Hz. These values are equivalent to those estimated by using Fig. 2.2–7.

Example 2.2

The circuit of Fig. 2.2–8 was considered for the analysis of an amplifier. This circuit is a design given in the Hewlett–Packard Application Note 95 (Froehner, 1968). The active device was a 2N3570 bipolar–junction transistor with the following S–parameters at 500 MHz:

$$s_{11} = 0.385 \angle -55^\circ$$
$$s_{12} = 0.045 \angle 90^\circ$$
$$s_{21} = 2.700 \angle 78^\circ$$
$$s_{22} = 0.890 \angle -26.5^\circ$$

The Smith chart design by Froehner was for 12 dB gain, and resulted in the following discrete component values (note that P is used to denote a network parameter):

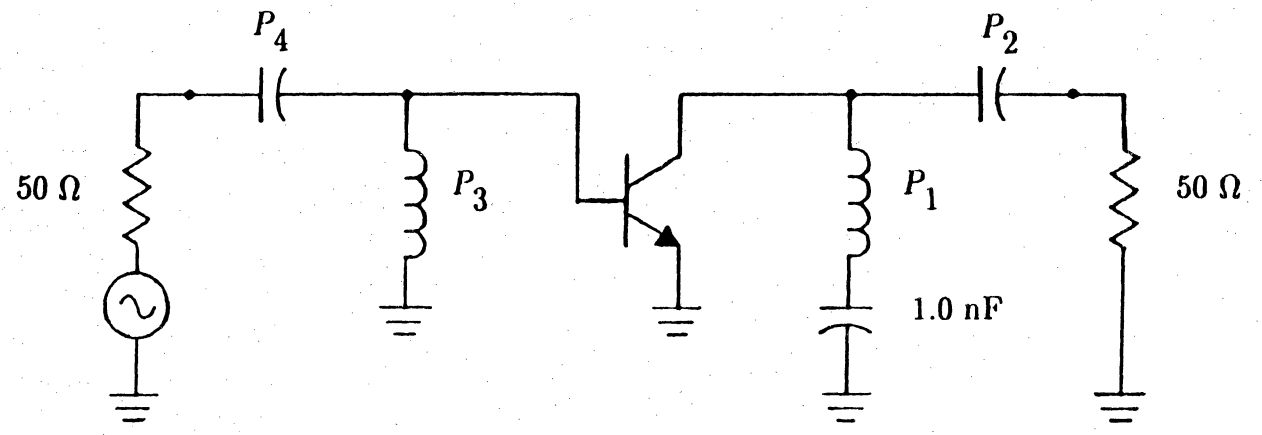


Figure 2.2–8 Amplifier with discrete components.

$$P_1 = 23.0 \text{ nH}$$
$$P_2 = 6.38 \text{ pF}$$
$$P_3 = 17.0 \text{ nH}$$
$$P_4 = 7.66 \text{ pF} \quad .$$

When these values were used in computer analysis, 11.964 dB gain was obtained. This result provides an additional example for validation of the computer code.

2.3 Optimization

Often we desire the computer–aided design program to specify component values needed to obtain a desired overall circuit response. Therefore, the program requires an optimization procedure which can obtain the desired response by variation of initial component values within specified limits. Fig. 2.3–1 demonstrates the flow of the basic optimization process. The desired response may be obtained by finding the minimum of a user specified objective function.

Several methods for minimizing the objective function are discussed by Bandler (1969). Versions of the same methods described by Bandler in 1969 are still used in optimization procedures today.

There are two basic types of multidimensional search strategies: direct and gradient. Direct search strategies do not employ explicit evaluations or estimations of partial derivatives of the objective function (Bandler, 1969). A few common search algorithms include Rosenbrock's method, the steepest descent method, Newton's method, and the Fletcher–Powell algorithm (Fletcher, 1963). While Rosenbrock's method is a direct search strategy, the other three methods mentioned

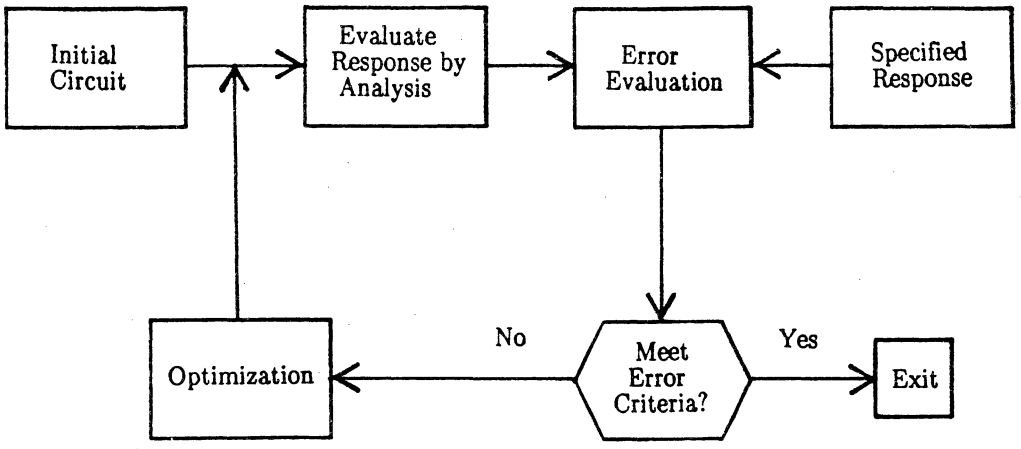


Figure 2.3–1 Flow chart of an optimization process (Ha, 1981).

are gradient search strategies. Forms of the steepest descent method and Newton's method are integrated into the Fletcher–Powell algorithm. Rosenbrock's method and the objective function are discussed in detail in Chapter 3.

3. OPTIMIZATION

When optimizing circuits, a computer–aided design program might utilize a function minimization procedure. Initial circuit parameter values, as well as the maximum and minimum limits on the parameter values, are passed to the procedure. The objective function is then calculated outside of the procedure in an optimization loop (see Appendix B). The objective function is defined by the user, and is evaluated after a full circuit analysis. If the objective function depends on frequency (as in filter design), the circuit must be analyzed once for each frequency of interest. The optimization changes the parameter values, within limits, to minimize the objective function. The optimization process is continued until either a minimum function value is obtained, or a maximum allowed number of function evaluations is reached.

One way of deciding that a minimum function value has been obtained is to analyze the function difference between successive attempts to find a better minimum. We decide that a minimum has been reached if the change is less than a certain value, the final function accuracy. Another method is to analyze circuit parameter differences between successive optimization steps. If all differences fall below a certain parameter accuracy, the optimization is considered complete. This parameter accuracy is usually given as a percentage of the parameter value range.

Before considering specific design examples, we discuss the objective function and the minimization procedure utilized by the developed program.

3.1 The Objective Function

To optimize a circuit by computer, the user must first define an objective function which, when minimized, will achieve desired responses. The objective function consists of several terms, with each term corresponding to a particular circuit response aspect. Every term is multiplied by a frequency dependent weighting coefficient so that the user, by entering positive real numbers for the coefficients, may choose which response elements will be minimized. If more importance is to be placed on one desired response element over another, the weighting coefficient for the more important response term is made larger. If the designer does not care about some responses, the weighting coefficients for those response terms will be set to zero.

A common response of interest is maximum forward power gain in an amplifier design. Maximum forward gain at a single frequency implies that the input and output impedances of the active device used in the amplifier (usually a transistor) are conjugate matched to the source and load impedances. Minimizing the inverse of $\left| s_{21} \right|^2$ has the effect of maximizing the forward power gain, or transducer gain, of the amplifier. In filter design, it may be desirable to maximize forward power gain over the frequencies in the pass band.

Besides maximizing gain, it may be desirable to minimize the sensitivity of a system response S–parameter to a possibly varying component parameter. For example, the overall s_{21} of an amplifier is usually a critical specification; however, the s_{21} of the transistor may differ from device to device. Optimization may be used to create a circuit which minimizes the sensitivity of the overall s_{21} to the transistor s_{21}. If a percentage change in the system parameter divided by a percentage change in the component parameter is defined as σ, the magnitude $\left| \sigma \right|$

is included in the objective function for minimizing relative sensitivity.

To minimize the input and output power reflections, we include the term $\left| \left| s_{11} \right|^2 + \left| s_{22} \right|^2 \right|$. Minimizing this term implies maximizing the power entering the two–port from the source, and the power delivered to the load. This is equivalent to making the input and output impedances equal to the source and load impedances. The computer–aided design program can match the input and output impedances of a two–port to any specified source and load impedances or reflection coefficients by appropriate normalization of the S–parameters; therefore, the designer can minimize input and output reflection to make the two–port efficiently transfer power even if the stages before and after the two–port do not have output and input impedances equal to the standard reference impedance.

Often the circuit designer desires to specify the forward power gain of a two–port. This may be the case when system stability or power budget constraints have dictated the gain of an amplifier, or when a flat frequency behavior is required. Achieving a desired gain may be accomplished by minimizing either one or both of the terms $\left| \left| s_{21} \right|^2 - G \right|^q$ and $\left| 10log \left| s_{21} \right|^2 - G_{dB} \right|$. Minimizing the first term will achieve a gain of G , with the exponent q used to emphasize the differences between the actual and desired gains, thus allowing for quicker initial convergence. The second term is used to achieve a logarithmic gain of G_{dB} , where G_{dB} is the desired gain in decibels.

In filter design, both the desired gain in the pass band and the maximum allowable gain in the reject band are typically specified. This may be accomplished by including the term $max[\, 0 \,,\, (\, 10log \left| s_{21} \right|^2 - G_{dB} \,) \,]$ in the objective function. The max operator implies that the maximum of the two values within the brackets will be included in the objective function, so that optimization will attempt to minimize the actual gain when it is greater than G_{dB} . If the actual gain is less

than G_{dB} , the term is ignored and no optimization is attempted since the filter specification is met. Note that an emphasis of the desired pass band performance versus reject band gain requirements may be enforced by appropriate choices for the weighting coefficients of the two previously mentioned terms.

To achieve an objective function with the terms mentioned above, we define the objective function E by

$$
\begin{aligned}
E = & \sum_{i} \frac{W_1(i)}{\left| s_{21} \right|^2} \\
& + W_2(i) \left| \sigma \right| \\
& + W_3(i) \left| \left| s_{11} \right|^2 + \left| s_{22} \right|^2 \right| \\
& + W_4(i) \left| \left| s_{21} \right|^2 - G \right|^q \\
& + W_5(i) \left| 10log \left| s_{21} \right|^2 - G_{dB} \right| \\
& + W_6(i)\, max[\, 0 \,,\, (\, 10log \left| s_{21} \right|^2 - G_{dB}\,)\,]
\end{aligned}
\qquad (3.1\text{–}1)
$$

where i is the frequency index. Note that E is a positive real number, and that the weighting coefficients, W_1 through W_6 , are positive real numbers that depend on frequency. Now that an objective function has been defined, computer–aided optimization can be achieved with the utilization of a function minimization routine.

3.2 Rosenbrock's Method (as modified by Palmer)

The function minimization routine used in this thesis is based on Rosenbrock's method (1960) as improved by Palmer (1969) and Swann (1964). A minimum is searched for in N directions, where N is the number of circuit components being used in optimization. Once a minimum is found in one direction, the routine begins from that point to search in the next variable direction. While searching in a particular direction, the minimum is estimated by fitting three points to a quadratic.

Two different search formats may be used: a quadratic fit about the minimum or a quadratic fit about the first three points. If the quadratic fit about the minimum is used, the function is computed during a search in one direction until the function begins to increase. Then the last three points computed are used for the quadratic fit. If the first three points format is used, only the first three points computed are used for the quadratic fit. In either case, if the function is found to be nondecreasing, the point with smallest function value will be declared the minimum.

After all N directions have been searched, a conjugate gradient type approach is used to decide the next direction of search. That is, the next direction is given by adding the N distance vectors found in the previous search. Rosenbrock's rotating coordinates method is then used to find $(N-1)$ remaining orthogonal directions of search.

This routine also allows for random directions of the search vector. After a minimum is found using the procedure outlined above, attempts to find a better minimum is made by searching in random directions. This is done by replacing the N component vectors by N random distances, and repeating the above search

technique. This random process may be repeated a specified number of times.

This minimization procedure may be better understood if we look at a two–dimensional case. Suppose we are optimizing a circuit by varying a resistance R and a capacitance C. Also, suppose we have set limits R_{min}, R_{max}, C_{min}, and C_{max} on these parameters. The minimization search will begin at the initial values R_o and C_o specified by the user (see Fig. 3.2–1). The search continues along R until a minimum is found at (C_o, R_1). Then C is varied until a minimum is found at (C_1, R_1). The next direction of search is given by

$$\vec{d}_1 = \vec{R}\frac{\Delta R}{R_o} + \vec{C}\frac{\Delta C}{C_o}, \qquad (3.2\text{–}1)$$

where $\vec{R}$ and $\vec{C}$ are resistance and capacitance vector directions, $\Delta R = R_1 - R_o$, and $\Delta C = C_1 - C_o$. If a minimum is not found along $\vec{d}_1$, the orthogonal direction $\vec{d}_2$ will be searched. Or if the best minimum along $\vec{d}_1$ is found at (C_2, R_2), the direction $\vec{d}_3$ will be searched. If a parameter boundary (a maximum or minimum component limit) is reached during the search, the search continues along that boundary.

In the following optimization examples, a parameter accuracy of 0.001 and the first three points search format were employed.

Example 3.1

The circuit of Fig. 3.2–2 was optimized for maximum gain. The amplifier was operated at 750 MHz with the 2N3570 transistor scattering parameters

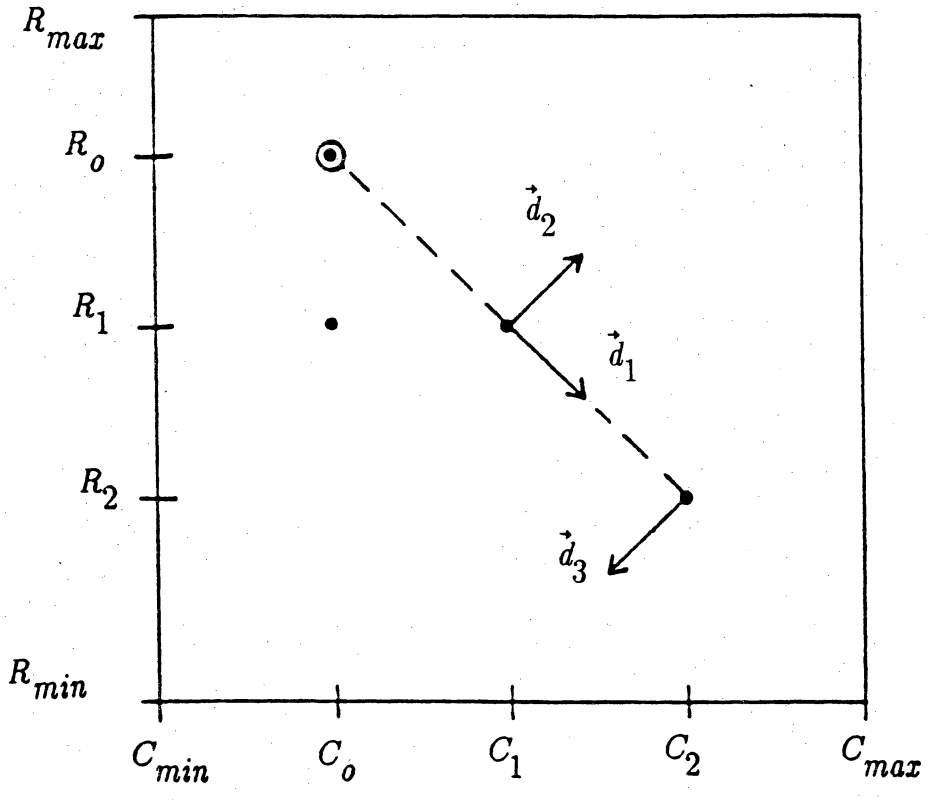


Figure 3.2–1 Two–dimensional optimization search.

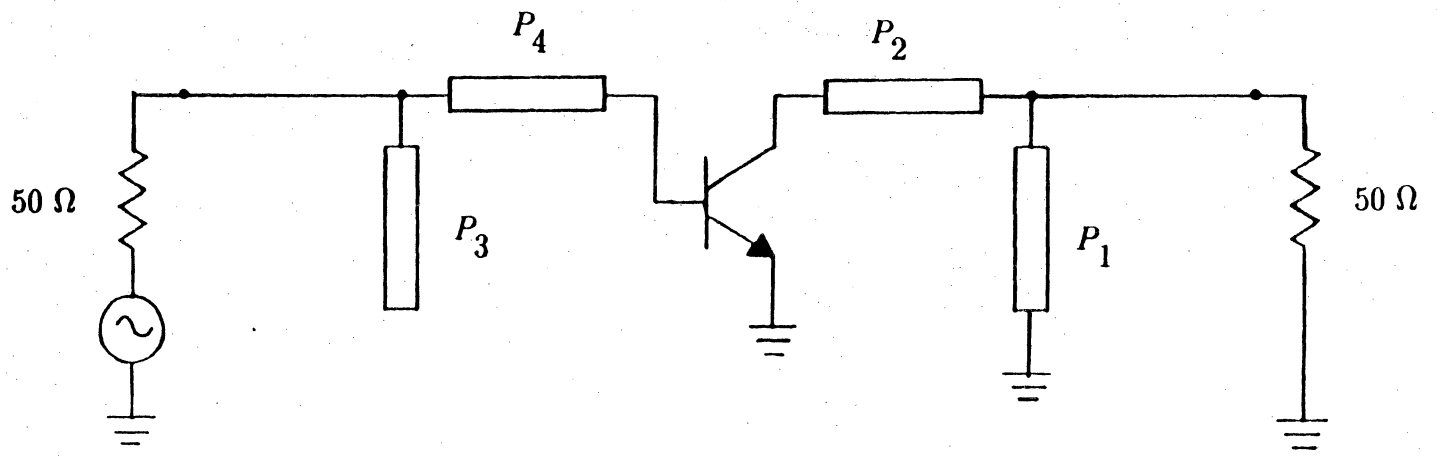


Figure 3.2–2 Amplifier with transmission lines.

$$s_{11} = 0.277 \angle -59^{\circ}$$
$$s_{12} = 0.078 \angle 93^{\circ}$$
$$s_{21} = 1.920 \angle 64^{\circ}$$
$$s_{22} = 0.848 \angle -31^{\circ} .$$

All the 50 ohm strip lines were initially specified to have lengths of 3 cm, and minimum and maximum limits of 0.6 cm and 6.0 cm. With the W_1 coefficient of equation (3.1–1) set to 10, a maximum gain of 12.807 dB was reached with computer optimization. The resulting parameters were

$$P_1 = 7.26 \text{ mm}$$
$$P_2 = 4.98 \text{ cm}$$
$$P_3 = 5.02 \text{ cm}$$
$$P_4 = 3.42 \text{ cm} .$$

Froehner's maximum gain circuit design (Froehner, 1968), resulted in the following parameters :

$$P_1 = 7.15 \text{ mm}$$
$$P_2 = 4.97 \text{ cm}$$
$$P_3 = 5.05 \text{ cm}$$
$$P_4 = 3.42 \text{ cm} .$$

When the above values were used in a computer analysis, a gain of 12.788 dB was achieved. Design equations show that the maximum gain available is 12.807 dB

(Froehner, 1968) ; therefore, the computer–aided optimization result was slightly closer to the theoretical result than the result obtained from Froehner's design, as would be expected.

Example 3.2

To further demonstrate optimization capabilities, the circuit of Example 2.2 was again considered. The circuit was optimized to obtain a gain of 10 dB rather than the 12 dB of Example 2.2. The frequency of operation was 500 MHz and the device S–parameters were the same as those given in Example 2.2 . The capacitors were given initial values of 10 pF, and minimum and maximum limits of 1.0 pF and 100 pF. The inductors were given initial values of 1.0 μH, and minimum and maximum limits of 0.001 μH and 10 μH. With the objective function coefficient W_5 set to 10, a gain of 10 dB was obtained with optimization. The resulting parameters were

$$P_1 = 29.2 \text{ nH}$$
$$P_2 = 18.2 \text{ pF}$$
$$P_3 = 17.6 \text{ nH}$$
$$P_4 = 7.54 \text{ pF} \ .$$

It should be noted that this was only one solution that could be used to achieve the specified gain. Several choices of the circuit values would satisfy this simple constraint. Thus, in a pratical sense, the designer would probably include additional optimization constraints such as tighter limits on component values, or minimization of input and output power reflection.

In Chapter 4, we will look at additional examples of optimization, emphasizing minimization of relative parameter sensitivity.

4. SENSITIVITY

In this chapter we will discuss Monaco and Tiberio's adjoint network approach to computing relative S–parameter sensitivity. Their approach is discussed so that it may be theoretically compared to a new bilinear method which is presented along with with some examples of computer–aided analysis and optimization involving sensitivity aspects.

4.1 The Adjoint Network Method

Since around 1970, the adjoint network method has been widely used in computer–aided design programs in the estimation of partial derivatives. These partial derivatives are used in gradient search optimization strategies or in the computation of relative sensitivity. The formulation of the adjoint network method (Director, 1969) followed from Tellegen's theorem (Tellegen, 1952; Penfield, 1970). Basically, the network of interest must be analyzed in two forms: the original form and the adjoint network form. Both the original and the adjoint networks have the same topology; however, the scattering matrix for a component in the adjoint network is equal to the transpose of the scattering matrix for that component in the original network. From the results of both analyses, equations may be formulated to estimate partial derivatives of network responses with respect to network parameters.

Using adjoint network concepts, formulations have been made for finding sensitivities in terms of scattering parameters (Monaco, 1970; Iuculano, 1971). Monaco and Iuculano have discussed how to compute the following sensitivities :

$$\textit{Magnitude Sensitivity} = \frac{p}{|s_{kj}|}\frac{\partial |s_{kj}|}{\partial p}$$

$$\textit{Group Delay} = \tau_{kj} = \frac{\partial \phi_{kj}}{\partial \omega}$$

$$\textit{Group Delay Sensitivity} = \frac{p}{\tau_{kj}}\frac{\partial \tau_{kj}}{\partial p}\ , \qquad (4.1\text{–}1)$$

where p is a network parameter, s_{kj} represents an arbitrary response S–parameter with magnitude $|S_{kj}|$ and phase ϕ_{kj}, and ω is the radian frequency. In the following formulation, the hth component is an internal component of interest with scattering parameters S^h depending on the network parameter p.

In formulating an equation for the estimation of the partial $\partial s_{kj}/\partial p$, Monaco begins by stating that the incident wave vector $\boldsymbol{a}^h$, relative to the hth–component ports may be determined in terms of matrices S^h and S^w relatto the component h itself, and to the subnetwork w containing the remaining components. The matrices S^h and S^w are divided into submatrices which separthe variables relative to the external ports from those relative to the connected ports. These submatrices are given by

$$\begin{bmatrix} \boldsymbol{b}_p^h \\ \boldsymbol{b}_c^h \end{bmatrix} = \begin{bmatrix} S_{pp}^h & S_{pc}^h \\ S_{cp}^h & S_{cc}^h \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_p^h \\ \boldsymbol{a}_c^h \end{bmatrix} \qquad (4.1\text{–}2)$$

and

$$\begin{bmatrix} \boldsymbol{b}_p^w \\ \boldsymbol{b}_c^w \end{bmatrix} = \begin{bmatrix} S_{pp}^w & S_{pc}^w \\ S_{cp}^w & S_{cc}^w \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_p^w \\ \boldsymbol{a}_c^w \end{bmatrix} , \tag{4.1–3}$$

where $\boldsymbol{a}_c^h$, $\boldsymbol{b}_c^h$, and $\boldsymbol{a}_p^h$, $\boldsymbol{b}_p^h$, are the incident and reflected power wave vectors relative to the ports of component h connected and nonconnected to the subnetwork w respectively; similarly, $\boldsymbol{a}_c^w$, $\boldsymbol{b}_c^w$, and $\boldsymbol{a}_p^w$,$\boldsymbol{b}_p^w$, are the vectors relative to subnetwork w connected and nonconnected to component h respectively (see Fig. 4.1–1). Similar matrix equations exist for the adjoint network, in which the adjoint submatrices are simply the transposes of those for the original network.

Since component h is connected to the subnetwork w , the following must hold true:

$$\boldsymbol{b}_c^h = \boldsymbol{a}_c^w \qquad \boldsymbol{a}_c^h = \boldsymbol{b}_c^w . \tag{4.1–4}$$

Assuming only the nth component has its parameters depending on p , so that

$$\frac{\partial S^h}{\partial p} = 0 \quad \text{for } h \neq n ,$$

Monaco forms the generalized equation

$$\frac{\partial s_{kj}}{\partial p} = \tilde{\mathcal{K}}_k^n \frac{\partial S^n}{\partial p} A_j^n \tag{4.1–5}$$

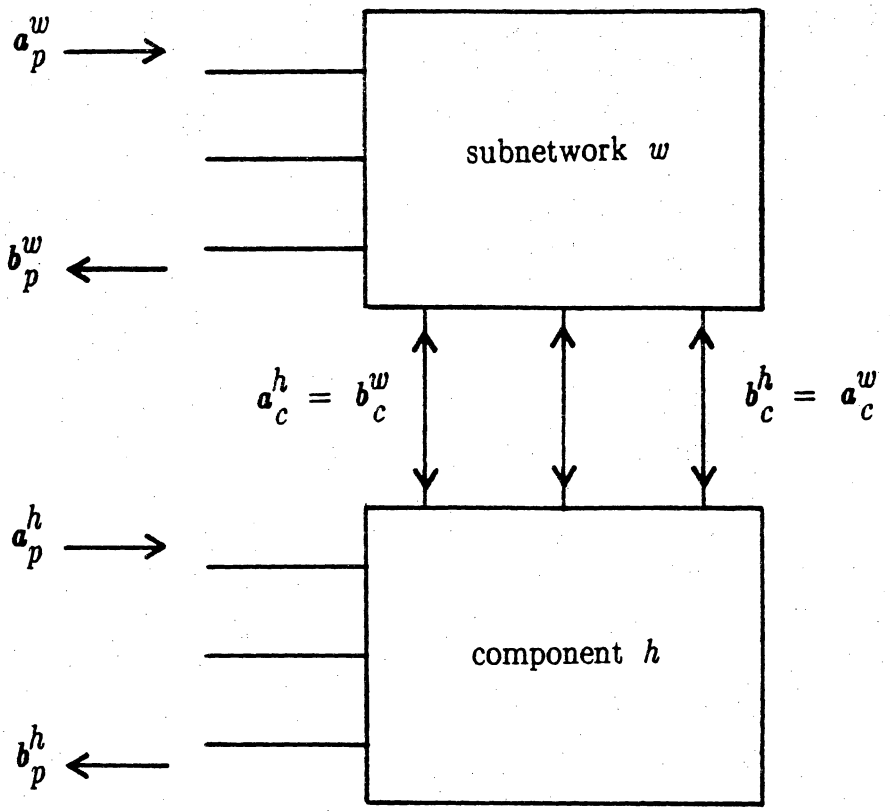


Figure 4.1–1 Diagram of a component h isolated from the subnetwork w.

where $\mathscr{A}_k^n$ is the kth column of the matrix

$$\mathscr{A}^n = \begin{bmatrix} 0 & E_p^n \\ N\tilde{S}_{pc}^w & N\tilde{S}_{cc}^w\tilde{S}_{pc}^n \end{bmatrix} , \tag{4.1–6}$$

A_j^n is the jth column of the matrix

$$A^n = \begin{bmatrix} 0 & E_p^n \\ MS_{cp}^w & MS_{cc}^wS_{cp}^n \end{bmatrix} , \tag{4.1–7}$$

where

$$N = (E_c - \tilde{S}_{cc}^w\tilde{S}_{cc}^n)^{-1} ,$$

$$M = (E_c - S_{cc}^wS_{cc}^n)^{-1} ,$$

E_c is a unity matrix of order c, and E_p^n is a unity matrix of order equal to the number of nth–component external ports. Note that the ˜ accent designates the matrix transpose operation.

The form of (4.1–5) may also be obtained by solving for the overall S in terms of S^w and S^n. The resulting equation may then be differentiated with respect to p using differential identities for matrices including the parametric derivative of a matrix inverse. This result is identical to (4.1–5).

If the network parameter p is the component S–parameter s_i, equation (4.1–5) simplifies to

$$\frac{\partial s_{kj}}{\partial s_i} = \tilde{\mathcal{A}}_k^n \frac{\partial S^n}{\partial s_i} A_j^n \,. \tag{4.1–8}$$

The matrix form which may be obtained for the overall S may be expressed in an identical bilinear matrix form to be presented later in section 4.3. The fundamental difference in the two methods thus becomes the process used to determine the terms in the equation. Solving for equation (4.1–8) requires that the submatrices of equations (4.1–2) and (4.1–3) must be known for both the original and adjoint networks. A new method of calculating sensitivity involves a bilinear relation between any system S–parameter to any component S–parameter. Three complex constants must be found to describe the bilinear transformation; therefore, the network must be analyzed three times to form the independent equations needed to find these constants. Although the formulation of Monaco and Tiberio (1970) requires only two complete analyses of the network, the new formulation is worth discussion. The following is an approach that requires no knowledge of submatrices, and is easily implemented in a circuit analysis/optimization program.

4.2 Derivation of a New Bilinear Relative Sensitivity Equation

In designing a circuit we may desire to find the dependence of any system (overall) scattering parameter on any component(internal) scattering parameter. Specifically, we would like to find the relative sensitivity σ given by

$$\sigma = \frac{\partial s \;/\; s}{\partial s_i \;/\; s_i} \,, \tag{4.2–1}$$

where s is the system S–parameter of interest and s_i is the internal S–parameter.

To determine the relative sensitivity we need a relationship between s and s_i. This relationship may be derived using the signal flow graph of Fig. 4.2–1. In the figure, the ports designated by 1 and 2 represent the external ports of a two–port linear network. Port i represents an internal port where s_i is the S–parameter of interest. We know that the system S–parameters are given by the equations

$$b_1 = s_{11}\, a_1 + s_{12}\, a_2 \tag{4.2–2a}$$

$$b_2 = s_{21}\, a_1 + s_{22}\, a_2 \tag{4.2–2b}$$

where $b_{1(2)}$ is the reflected power wave at port 1(2), and $a_{1(2)}$ is the incident power wave at port 1(2). Following the signal flow graph of Fig. 4.2–1, we can write

$$b_1 = s_{11_w} a_1 + s_{12_w} a_2 + \frac{s_{1i}\, s_i\, s_{i1}\, a_1}{1 - s_{ii}\, s_i} + \frac{s_{1i}\, s_i\, s_{i2}\, a_2}{1 - s_{ii}\, s_i}$$

$$= a_1 \frac{s_{11_w} + \left(s_{1i}\, s_{i1} - s_{11_w}\, s_{ii} \right) s_i}{1 - s_{ii}\, s_i}$$

$$+ a_2 \frac{s_{12_w} + \left(s_{1i}\, s_{i2} - s_{12_w}\, s_{ii} \right) s_i}{1 - s_{ii}\, s_i} . \tag{4.2–3}$$

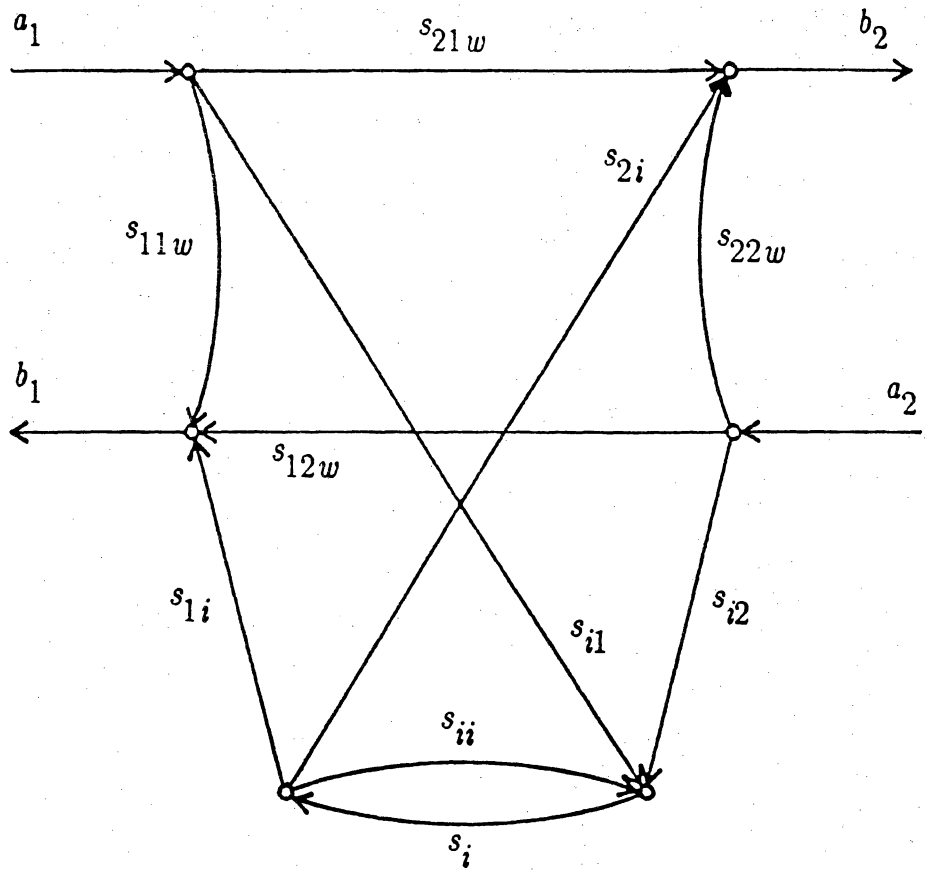


Figure 4.2–1 Flow diagram for two–port including internal port.

Note that the denominator term $1 - s_{ii}s_i$ is a result of the feedback loop of s_i and s_{ii} at port i.

This approach may be used for any set of input and output ports j and k giving system parameters of the form

$$s_{jk} = \frac{s_{jk_w} + \left(s_{ji}s_{ik} - s_{jk_w}s_{ii} \right) s_i}{1 - s_{ii}s_i} \qquad (4.2\text{–}4)$$

Since we are typically interested in how one system S–parameter s varies with the internal S–parameter s_i, we may simply rewrite (4.2–4) as

$$s = \frac{a + b\,s_i}{1 + d\,s_i}, \qquad (4.2\text{–}5)$$

where a, b, and d are complex constants. The constants a and b should not be confused with the power waves a and b. Differentiating (4.2–5) we find that

$$\frac{\partial s}{\partial s_i} = \frac{b - d\,s}{1 + d\,s_i}. \qquad (4.2\text{–}6)$$

Note that we could have solved for $\partial s/\partial s_i$ in terms of s_i only; however, in our optimization work, both s_i and the corresponding s are available. Finally, the expression for sensitivity is found by substituting (4.2–6) into (4.2–1) to obtain

$$\sigma = \frac{b - d\,s}{1 + d\,s_i}\,\frac{s_i}{s}\,, \tag{4.2–7}$$

or in terms of s_i only

$$\sigma = \frac{(\,b - a\,d\,)\,s_i}{(\,1 + d\,s_i\,)\,(\,a + b\,s_i\,)}\,. \tag{4.2–8}$$

Calculation of sensitivity is easily implemented in a computer–aided design program. To find the complex constants a , b , and d , we evaluate the circuit of interest three times, each time with a different internal S–parameter s_i . The results may be used to form three independent equations in terms of the internal S–parameters and the corresponding system S–parameters given by

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} 1 & s_{i1} & -s_1 s_{i1} \\ 1 & s_{i2} & -s_2 s_{i2} \\ 1 & s_{i3} & -s_3 s_{i3} \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix}\,, \tag{4.2–9}$$

which is easily solved for the coefficients a , b , and d . Computational procedures for bilinear sensitivity analysis are outlined in Appendix A.

4.3 Generalization of the Bilinear Relative Sensitivity Equation

Equation 4.2–5 can be generalized to a matrix equation giving the relation between the system scattering matrix and any component scattering matrix. We know that

$$\boldsymbol{b} = \boldsymbol{S}\,\boldsymbol{a} \qquad \text{and} \qquad \boldsymbol{b}_i = \boldsymbol{S}_i\boldsymbol{a}_i \ , \tag{4.3–1}$$

where $\boldsymbol{S}$ is the system scattering matrix, and $\boldsymbol{S}_i$ is the component scattering matrix. The component is connected to the rest of the network such that

$$\boldsymbol{b}_i = \boldsymbol{a}_w \qquad \text{and} \qquad \boldsymbol{a}_i = \boldsymbol{b}_w \ , \tag{4.3–2}$$

or equivalently

$$\boldsymbol{a}_w = \boldsymbol{S}_i\boldsymbol{b}_w \ , \tag{4.3–3}$$

where the w subscript designates the variables connecting the subnetwork to the full system.

A system super matrix defining the system $\boldsymbol{S}$ after isolation of $\boldsymbol{S}_i$ may be described in partitioned form as

$$\begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b}_w \end{bmatrix} = \begin{bmatrix} \boldsymbol{S}_1 & \boldsymbol{S}_2 \\ \boldsymbol{S}_3 & \boldsymbol{S}_4 \end{bmatrix} \begin{bmatrix} \boldsymbol{a} \\ \boldsymbol{a}_w \end{bmatrix} . \tag{4.3–4}$$

Using (4.3–3) and (4.3–4), we find $\boldsymbol{a}_w$ in terms of $\boldsymbol{a}$ as

$$\boldsymbol{a}_w = \boldsymbol{S}_i\boldsymbol{b}_w = \boldsymbol{S}_i(\boldsymbol{I}_i - \boldsymbol{S}_4\boldsymbol{S}_i)^{-1}\boldsymbol{S}_3\boldsymbol{a} \ , \tag{4.3–5}$$

where $\boldsymbol{I}_i$ is an identity matrix with order equal to the order of the component

matrix S_i. Substituting (4.3–5) into (4.3–4), we obtain

$$b = [\, S_1 + S_2 S_i (I_i - S_4 S_i)^{-1} S_3]\, a = S a \ . \tag{4.3–6}$$

which may be rearranged to obtain the generalized equation

$$S = (A + B S_i)(C + D S_i)^{-1} \ , \tag{4.3–7}$$

where

$$A = S_1 S_3^{-1}$$
$$B = -S_1 S_3^{-1} S_4 + S_2$$
$$C = S_3^{-1}$$

and $$D = -S_3^{-1} S_4 \ .$$

Note that the matrices A, B, C, and D are independent of the subsystem matrix S_i. These matrix equations may be used to define relative sensitivity in a multiparameter form. This multidimensional formulation is presented for completeness. However, consideration of sensitivity is restricted to single parameter cases for which we now consider several examples.

Example 4.1

In order to demonstrate relative sensitivity analysis, we considered the circuit of Fig. 2.2–8, using the device S–parameters given in Example 2.2, with the

following circuit parameters:

$$P_1 = 23.0 \text{ nH}$$
$$P_2 = 6.38 \text{ pF}$$
$$P_3 = 17.0 \text{ nH}$$
$$P_4 = 7.66 \text{ pF} \quad .$$

The relation between the overall s_{21} and the device s_{21i} was found to be

$$s_{21} = \frac{(-0.5 + j1.4)\, s_{21i}}{1 + (9.8\times10^{-3} + j7.07\times10^{-4})\, s_{21i}}$$

giving an overall s_{21} of $-3.94 + j0.45$ for a device s_{21i} of $0.56 + j2.64$. Also, From (4.2–9) the sensitivity for s_{21} with respect to s_{21i} given by

$$\sigma = \frac{\partial s_{21} \,/\, s_{21}}{\partial s_{21i} \,/\, s_{21i}}$$

was found to be

$$\sigma = \frac{1}{1 + (9.8\times10^{-3} + j7.07\times10^{-4})\, s_{21i}}$$

giving a relative sensitivity magnitude of 0.996 for a device s_{21i} of $0.56 + j2.64$. From the form above it is clear that the sensitivity of the system s_{21} to variations

in s_{21i} may be minimized by using a device with a large s_{21}. This is analogous to using a high–β transistor and controlling the gain with feedback or loading.

The analogy between the β (or h_{fe}) and the s_{21} of a BJT, or the g_m and the s_{21} of a FET, is obvious since the device s_{21} is linearly dependent on h_{fe} for the BJT, and on g_m for the FET. Using the simplified small signal models for a BJT and a FET in Fig. 4.2–2, the specific relations are found to be

$$s_{21} = \frac{-2\,h_{fe}\,Z_{ref}}{h_{ie} + Z_{ref}} \tag{4.2–10}$$

and

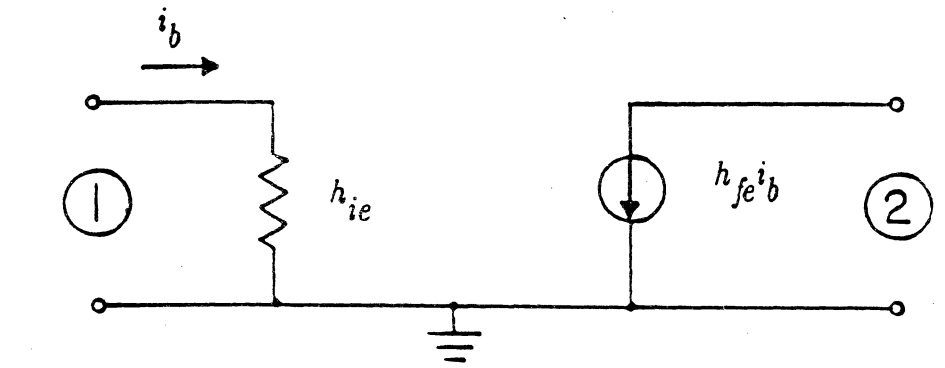
$$s_{21} = \frac{-2\,g_m r_d\,Z_{ref}}{r_d + Z_{ref}} \,. \tag{4.2–11}$$

Therefore, $\partial s_{21i}/s_{21i} = \partial h_{fe}/h_{fe}$ for the BJT, and $\partial s_{21i}/s_{21i} = \partial g_m/g_m$ for the FET, as would be expected.

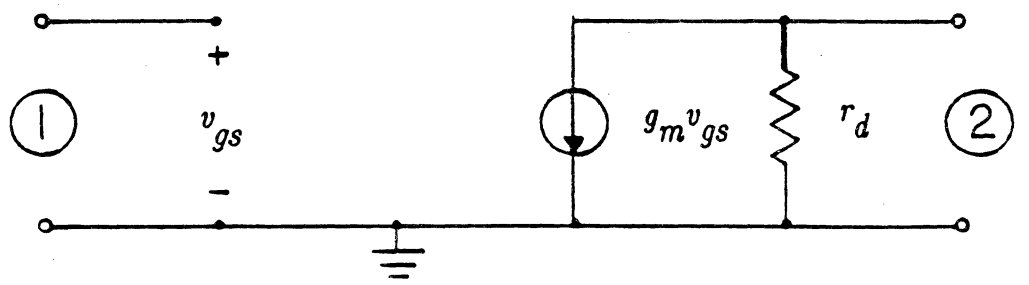
Example 4.2

As another example of sensitivity analysis, we considered the circuit of Fig. 3.2–2, using the device S–parameters given in Example 3.1, with the following circuit parameters:

$$P_1 = 7.15 \text{ mm}$$
$$P_2 = 4.97 \text{ cm}$$



a)



b)

Figure 4.2–2 Small–signal models for (a) a BJT and (b) a FET.

$$P_3 = 5.05 \text{ cm}$$
$$P_4 = 3.42 \text{ cm} \quad .$$

The bilinear relation between s_{11} and s_{21i} was found to be

$$s_{11} = \frac{(-0.62 - j0.4) + (0.33 - j0.19)s_{21i}}{1 + (-.083 + j0.27)s_{21i}} \quad .$$

giving an overall s_{11} of $-0.026 + j0.04$ for a device s_{21} of $0.84 + j1.73$. The relative sensitivity for s_{11} with respect to s_{21i} given by

$$\sigma = \frac{\partial s_{11} / s_{11}}{\partial s_{21i} / s_{21i}} \quad .$$

was found to be $\sigma = 0.041 - j32.3$. The sensitivity magnitude was 32.3.

This magnitude of the relative sensitivity is a measure of how much the input reflection coefficient of the circuit is affected by a change in the s_{21} (or β) of the active device. The next example will demonstrate how we may minimize this sensitivity, to improve input matching, without sacrificing overall power gain. A power gain of 12.79 dB was obtained for this example.

Example 4.3

The circuit of Example 4.2 was considered for optimization of maximum transducer gain while minimizing the the magnitude of the relative sensitivity

$$\sigma = \frac{\partial s_{11} \; / \; s_{11}}{\partial s_{21i} \; / \; s_{21i}} \; .$$

The maximum gain weighting coefficient W_1 was set to 10 and the sensitivity coefficient W_2 was set to 1 in the objective function (3.1–1). A transducer gain of 9.85 dB and a sensitivity magnitude of 0.45 was obtained with optimization compared to a gain of 12.79 dB and a sensitivity magnitude of 32.3 obtained before optimization. The resultant circuit parameters were as follows:

$$P_1 = 1.92 \text{ cm}$$
$$P_2 = 4.06 \text{ cm}$$
$$P_3 = 5.07 \text{ cm}$$
$$P_4 = 4.19 \text{ cm} \; .$$

With a loss in gain of only 2.94 dB, the relative sensitivity magnitude was decreased by about 98.6% of its original value. Therefore, with the circuit parameters given above, a relatively good transducer gain and minimal input reflection should be attainable despite variations in the active device s_{21}.

5. CONCLUSIONS AND RECOMMENDATIONS

In Chapter 4 of this thesis, a new method for finding relative scattering parameter sensitivity, referred to as the bilinear method, has been presented. This method involves a bilinear transformation rather than the adjoint network computations used by Monaco and Tiberio. The adjoint network method requires only two circuit analyses, while the bilinear method requires three; however, the bilinear method does not require the adjoint network computations involving submatrices. For the bilinear method, the original network is analyzed three times for variations in a specific internal scattering parameter. The bilinear transformation method provides a convenient description for sensitivity of a system relative to a scattering parameter of a network element, and may be extended to multiple–parameter sensitivity.

Examples have been presented for the sensitivity of a system S–parameter relative to a single internal device S–parameter. The method has been conceptually extended to the computation of the system sensitivity for an internal sub–system. The examples have been limited to variations in an active device parameter, though the technique is equally valid for the passive components of the system. Additional work should include experimental verification of computer–aided sensitivity analysis and optimization examples.

The program might be modified to allow for the free–format type input described by Jenkins and Fan (1971). Under a free–format, all the nodes in the network would be identified by a number. The user would then add a component by entering the type of component, its initial values and optimization limits, and

the numbers of the nodes between which it is connected. This format would facilitate circuit input description by allowing the user to think of connecting components between nodes rather than trying to visualize the specific connections of two–ports. A graphics routine which would allow the user to see the circuit modeled during initial development would be highly desirable. It would also be desirable to allow the user to specify more than two external ports. The procedures discussed by Gupta (1981) could be implemented to make possible the computation of multiport scattering parameters.

Other suggestions for additions to the program include allowing for the storage and retrieval of device data. This addition provides the user with the option of entering the device scattering parameters or entering the device model number. If the device model number is entered, device S–parameters needed in circuit analysis could be read (interpolated if required) from stored S–parameter data versus frequency. A more advanced computer–aided design program could allow for the computer to interface to S–parameter measurement equipment so that component S–parameters may be accurately measured when they are needed by the computer.

REFERENCES

Bandler, J. W. (1969), "Optimization methods for computer–aided design," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–17**, pp. 533–552.

Bandler, J. W., W. Kellermann, and K. Madsen (1985), "A superlinearly convergent minimax algorithm for microwave circuit design," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–33**, pp. 1519–1528.

Bandler, J. W., and P. A. MacDonald (1969), "Optimization of microwave networks by razor search," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–17**, pp. 552–562.

Bandler, J. W., and M. R. M. Rizk (1980), "Analysis and sensitivity of 2p–port cascaded networks," *IEEE Trans. Microwave Theory Tech. Symposium*, pp. 404–406.

Bandler, J. W., and R. E. Seviora (1970), "Current trends in network optimization," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–18**, pp. 1159–1170.

Bradley, P. and J. Ness (1983), "CAD/CAM Techniques in microwave circuit design," *J. Electr. Electron. Eng. Aust.*, vol. 3, pp. 266–274.

Charalambous, C. (1974), "A unified review of optimization," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–22**, pp. 289–300.

Compact Software , Inc. (1988), COMPACT and SUPER–COMPACT (computer programs), Palo Alto, California.

Director, S. W., and R. A. Rohrer (1969), "The generalized adjoint network and network sensitivities," *IEEE Trans. Circuit Theory*, vol. **CT–16**, pp. 318–323.

Duke, W. F., (1977), "Parameter independent design utilizing scattering parameters," M. S. Thesis, Air Force Institute of Technology.

EEsof (1986), Touchstone, Version 1.45 (computer program), Westlake Village, California.

Fernandes, H., A. J. Giarola, and D. A. Rogus (1983), "PACMO: A comprehensive CAD package for microwave devices," *IEEE Trans. Educ.*, vol. **E–26**, pp. 162–163.

Fletcher, R., and M. J. D. Powell (1963), "A rapidly convergent descent method for minimization," *Comput. J.*, vol. 6, pp. 163–168.

Froehner, W. H. (1968), "Quick amplifier design with scattering parameters," *Hewlett–Packard Application Note 95*, pp. 5.2–5.11 .

Gonzalez, G. (1984), *Microwave Transistor Amplifiers Analysis and Design.* Englewood Cliffs, N. J.: Prentice–Hall, Inc..

Grivet, P. (1976), *Microwave Circuits and Amplifiers.* New York: Academic Press, pp. 328–331.

Gupta, K. C., R. Garg, and R. Chadha (1981), *Computer Aided Design of Microwave Circuits.* Dedham, Mass.: Artech House.

Ha, T. T. (1981), *Solid State Microwave Amplifier Design.* New York: John Wiley & Sons.

Iuculano, G., V. A. Monaco, and P. Tiberio (1971), "Network sesitivities in terms of scattering parameters," *Electron. Lett.*, vol. 7, pp. 53–55.

Jenkins, F. S., S. P. Fan, "TIME— a nonlinear DC and time–domain circuit–simulation program," *IEEE J. Solid–State Circuits* (special issue on computer–aided circuit analysis and device modeling), vol. **SC–6**, pp. 182–188.

Kurokawa, K. (1965), "Power waves and the scattering matrix," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–13**, no. 3, pp. 194–202.

Monaco, V. A., and P. Tiberio (1970), "On linear network scattering matrix sensitivity," *Alta Frequenza*, vol. 39, pp. 193–195.

Monaco, V. A., and P. Tiberio (1974), "Computer–aided analysis of microwave circuits," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–22**(3), pp. 249–263.

Montgomery, C. G., and R. H. Dicke (1948), *Principles of Microwave Circuits.* MIT, Rad. Lab. series no. 8, New York: McGraw–Hill.

Palmer, J. R. (1969), "An improved procedure for orthogonalising the search vectors in Rosenbrock's and Swann's direct search optimisation methods," *Computer J.*, vol. 12, pp. 69–71.

Penfield, P., R. Spence, and S. Duinker (1970), "A generalized form of Tellegen's theorem," *IEEE Trans. Circuit Theory*, vol. **CT–17**, pp. 302–305.

Richards, G. A. (1969), "Second–derivative sensitivity using the concept of the adjoint network," *Electron. Lett.*, vol. 5, pp. 398–399.

Rosenbrock, H. H. (1960), "An automatic method for finding the greatest or least value of a function," *Computer J.*, vol. 4, pp. 175–184.

Sanchez–Sinencio, E. (1974), "CADMIC– Computer–aided design of microwave integrated circuits," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–22**, pp. 309–312.

Swann, W. H. (1964), "Report on the development of a new direct search method of optimisation," Imperial Chemical Industries Ltd., Central Instrument Laboratory Research Note 64/3.

Tellegen, B. D. H. (1952), "A general network theorem, with applications," *Philips Res. Rep.*, n. 7, pp. 259–269.

Trick, T. N., and J. Vlach (1970), "Computer–aided design of broadband amplifiers with complex loads," *IEEE Trans. Microwave Theory Tech.*, vol. **MTT–18**, pp. 541–547.

Appendix A: Sensitivity Calculations

Calculation of sensitivity is easily implemented in a computer–aided design program. To find the complex constants a , b , and d , the circuit of interest can be constructed three times, each time with a different internal S–parameter s_i . This results in three independent pairs consisting of the internal (component) S–parameters and the corresponding external (overall) S–parameters:

$$\begin{aligned} &(s_{ii} , s_{ei}) \\ &(s_{ij} , s_{ej}) \\ &(s_{ik} , s_{ek}) . \end{aligned} \qquad \text{(A–1)}$$

We can arbitrarily set one internal S–parameter s_{ii} to 0, and then (4.2–5) simplifies to $s_{ei} = a$. Therefore, $a = s_{ei}$, and we can now use the known values a , s_{i_1} , s_{e_1} , s_{ik} , and s_{ek} to solve for b and d . Using (4.2–5) we can formulate two equations with two unknowns:

$$\begin{aligned} s_{ej}(1 + d\, s_{ij}) &= a + b\, s_{ij} \\ s_{ek}(1 + d\, s_{ik}) &= a + b\, s_{ik} . \end{aligned} \qquad \text{(A–2)}$$

Multiplying the bottom equation by $-s_{ij} / s_{ik}$, and then adding the two equations leads to a solution for d :

$$d = \frac{a\,(\,s_{ik} - s_{ij}\,) - s_{ej}s_{ik} + s_{ij}s_{ek}}{s_{ik}s_{ij}(\,s_{ej} - s_{ek}\,)} \,. \quad \text{(A–3)}$$

Note that the only internal S–parameter equal to 0 is s_{ii}. However, we must be careful when s_{ej} is equal to s_{ek}. If they are equal, the equation for d becomes nonvalid, and we must assume that the external S–parameter $s_e = s_{ej} = s_{ek} = s_{ei}$. In that case the problem is solved with $a = s_{ei}$ and $b = d = 0$. Once d is known, the first equation in (A–2) can be used to derive a formula for b:

$$b = \frac{s_{ej}\,(\,1 + d\,s_{ij}\,)}{s_{ij}} \,. \quad \text{(A–4)}$$

Now that we know the procedure to find a, b, and d, we can implement sensitivity calculations in a circuit construction routine. Three essential program procedures are needed: one to set the internal S–parameter before each construction, one to save the external S–parameter after each construction, and one to calculate a, b, d, and the relative sensitivity σ after the third construction.

APPENDIX B: USERS GUIDE FOR DEVELOPED PROGRAM

This appendix describes the use of the computer–aided design program developed in order to demonstrate the concepts presented in this thesis. The program was written in Turbo Pascal version 3.0 and, if the command file SOPT.COM is present on the logged disk drive, is activated by entering SOPT at the DOS prompt. The program finds the overall two–port scattering parameters for a network, along with supplying frequency analysis, optimization, and sensitivity analysis capabilities. Basically the user must enter initial data, model the circuit by combining two–port modules, and enter optimization data. If sensitivity analysis is desired, appropriate information is entered with device data during the modeling process. A flowchart for the developed program is shown in Fig. B–1.

The program begins by asking if data is to be read from a file. If so, the user must enter the path, either 'a:' or 'b:', that circuit data is to be read from. The screen will display all of the files on the entered path disk, and the user may then enter which file data is to be read from. Once the file of interest is named, the computer reads the file and displays the circuit modeling information on the screen. The user then has the option to edit the information, enter optimization data, or analyze the circuit as is.

If the user wishes to edit previously recorded data, or if circuit information is being entered for the first time, the computer begins the editing routine. Basic information is entered first, such as the reference impedance, the transmission line velocity, whether the terminations are matched to the reference impedance or not,

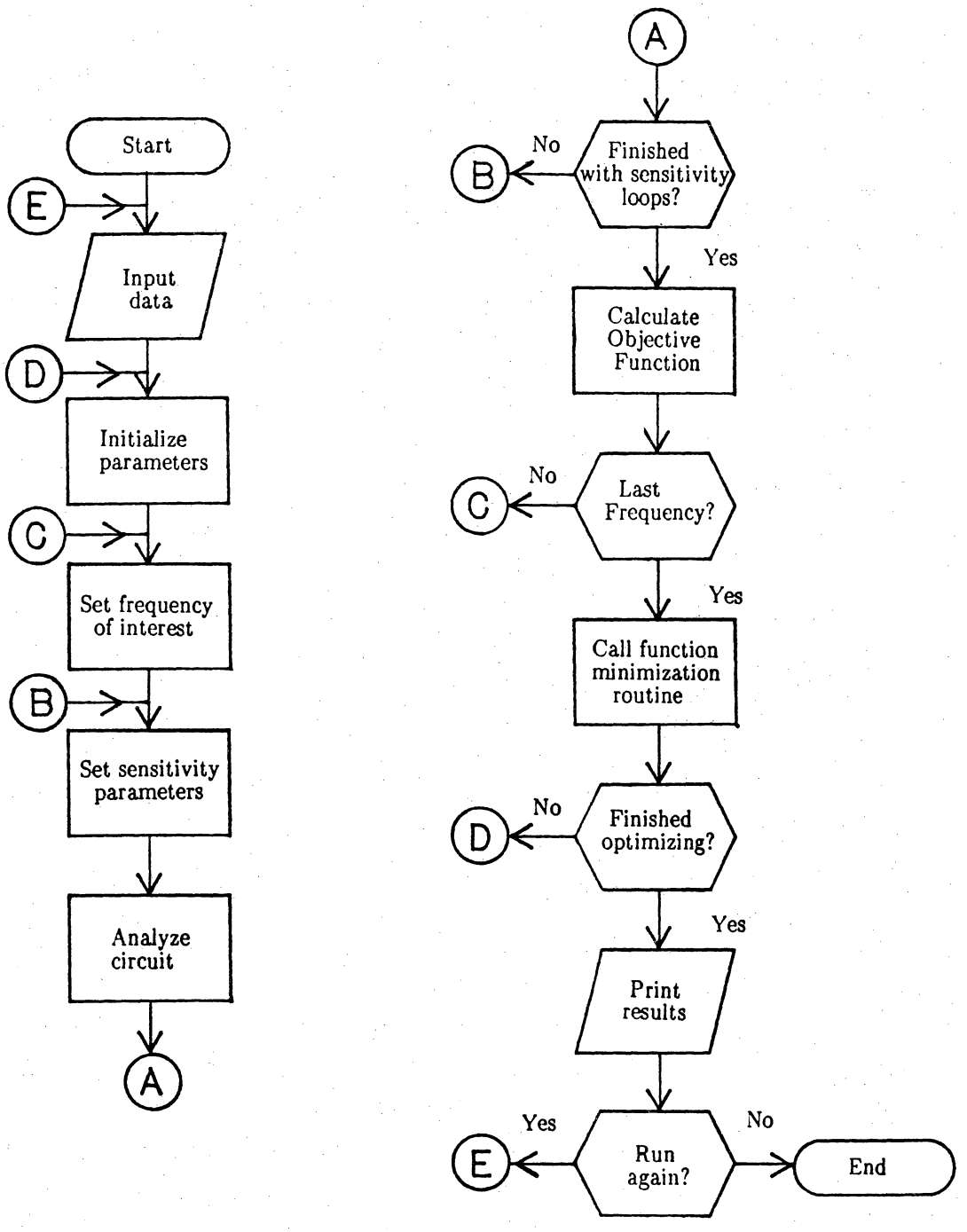


Figure B.1 Flowchart for computer–aided design program including frequency and sensitivity analysis, and optimization capability.

and the number of frequencies the circuit will be analyzed for. If the terminations are not to be matched to the reference impedance, the user must enter the complex source and load reflection coefficients that the final circuit will be matched to. Next the user enters the values for the frequency or frequencies of interest.

The circuit modeling section begins with the following menu screen:

Final S–parameters are given for Module #1

CardNo. = 1

Enter type of Card: D = Device (Input Data)

R = Resistor

C = Capacitor

L = Inductor

X = Transmission Line

A = Module combination

M = Module assignment

Enter Choice or Return if finished:

Note that when a passive component is entered, the user must specify its initial value, its maximum and minimum limits if optimization is desired, and whether it is used as a series element or a shunt element. The bottom of the menu screen displays the S–parameters (real and imaginary parts) of the available "scratch" modules A and B.

Module A =	A and B are open	Module B =
0.000E+00 + j0.000E+00		0.000E+00 + j0.000E+00
0.000E+00 + j0.000E+00		0.000E+00 + j0.000E+00

0.000E+00 + j0.000E+00 0.000E+00 + j0.000E+00

0.000E+00 + j0.000E+00 0.000E+00 + j0.000E+00

The scratch modules are the basic building blocks for the circuit model. After the first component is entered its S–parameters will appear in module A. If analysis is desired for more than one frequency, the S–parameters for the first frequency of interest will appear in the scratch module. The second component's S–parameters will appear in module B. When modules A and B are full, they may be combined to form one two–port by entering 'A' at the menu screen prompt. Note that entering the upper case 'A' or the lower case 'a' will have the same effect. The module combination screen appears as follows:

left module =

right module =

1 – Cascade

2 – Z

3 – Y

4 – H

5 – G

6 – 2–port shunt element, with left end treated as 1–port

7 – 2–port series element

Combination format number =

Thus the two scratch modules, or any two assigned modules, may be combined in any one of the combination formats discussed in Chapter 2 of this thesis. Note that modules may be identified with A, B, or any integer from 1 to 10. Also note that if

a two–port is to be used as a shunt or series element, that module must be entered for both the left and right modules. After the module combination the resultant S–parameters will appear in module A, unless A is full and B is clear before the combination, then the resultant parameters appear in module B.

The module assignment option allows the user to assign the S–parameters of any module to any other module. The user must enter the subject and object modules after entering 'M' from the menu screen. Note that a scratch module will be cleared after assigning its parameters to another module. Also note that the final S–parameters are given for module 1; therefore, the last command of the modeling routine is usually to assign the final circuit parameters in module A or B to module 1.

When a device is entered into the circuit model, the user must assign the device a module number, and provide a set a two–port parameters for the device at all the frequencies of interest. If a device analysis is desired, the following quantities will be computed after the two–port parameters are entered:

$$G1 = \frac{1}{1 - |s_{11}|^2} \qquad G2 = \frac{1}{1 - |s_{22}|^2}$$

$$G0 = |s_{21}|^2 \qquad \det = s_{11}s_{22} - s_{12}s_{21}$$

$$u = \frac{|s_{12}||s_{21}||s_{11}||s_{22}|}{(1 - |s_{11}|^2)(1 - |s_{22}|^2)}$$

$$K = \frac{1 + |\Delta_s|^2 - |s_{22}|^2 - |s_{11}|^2}{2|s_{12}s_{21}|}$$

$$\text{Gumax} = \text{G1G2G3} \qquad \text{Gamax} = \left| \frac{s_{21}}{s_{12}} \right| (K - \sqrt{K^2 - 1}) \ .$$

The source and load reflection coefficients needed to conjugately match the device input and output reflection coefficients are also given. Note that a K less than unity indicates that the device is potentially unstable. The source and load reflection coefficients and Gamax will not be given when this condition is met. Also, if either s_{21} or s_{12} is zero, K will be set to 10000. If the device is to be used in sensitivity calculations, the user must specify the internal (component) parameter and the external (system) parameter desired for the relative sensitivity defined by

$$\sigma = \frac{\partial s_e / s_e}{\partial s_i / s_i} \ ,$$

where s_e is the system parameter of interest and s_i is the component parameter. Entering a blank at the menu screen prompt will exit the circuit modeling routine.

After exiting the modeling routine the screen will display a summary of the circuit modeling information, giving the user the option of editing the information. Whether or not editing is desired, the user will have the option of saving the circuit modeling information to a disk file.

Next the user will be asked if optimization is to be done. If so, the user will be asked if the objective function is frequency dependent, and if it is desired to print optimization status to the screen during the optimization routine. The user must then specify the error function module, i.e. the module whose S–parameters will enter into the objective function calculations. Typically the error function module

is set to 1 in order to optimize the entire circuit. If optimization is desired the objective function will be displayed and the user must enter coefficient values after the following prompts:

For Maximum Gain:

For Sensitivity:

For abs(sqr(S11))+abs(sqr(S22)):

For Flat Gain:

For Log Gain Flatness:

For Minimum Gain:

For Truncated Log Gain Flatness:

The use of the objective function in optimization is explained in detail in Chapter 3 of this thesis. After the coefficients are entered, other optimization information is requested such as the actual gain desired, maximum number of optimization routine calls, number of random steps to confirm the minimum, the quadratic search type, and the relative parameter error. Typical values are 500 for the maximum number of optimization routine calls, 1 to designate the first three points search type, and 0.001 for the relative parameter error. Optimization variables are defined in Chapter 3 of this thesis.

After optimization, or after circuit modeling if no optimization is desired, the S–parameters for all the frequencies of interest will be displayed. If optimization was done, the optimization parameters will be displayed under "PALROS parameters = " and will be given in the order that they were identified as optimization parameters in the circuit modeling routine. Other information includes the final value of the objective function ObjF, the magnitude of the relative

sensitivity |sigma|, and the sensitivity constants a, b, and d. The sensitivity constants are defined in Chapter 4 of this thesis. The values of s_{21} in dB and the frequencies are also given.

When analysis and optimization are complete, the user has the option of printing the final results to a printer (hardcopy) or to a disk. If the results are printed to a disk, a file named "RESULTS" is created on the disk in the logged drive.

Appendix C: Listing for Developed Program

```
Program MicrowaveDesign;

{    ****************************************************************
     *                                                              *
     *                  RF / Microwave CAD Program                  *
     *                                                              *
     *             by:   W. A. Davis and D. Blackburn               *
     *                                                              *
     **************************************************************** }

{    ****************************************************************
     *                                                              *
     *    The following computer aided design program performs the  *
     * functions of calculating the overall S - parameters of a     *
     * network, the relative sensitivity, and the objective         *
     * function used in a direct search optimization.   This        *
     * program is a Pascal translation of an original program devel *
     * oped by Capt. Willaim F. Duke (USAF) and later modified by   *
     * Dr. William Davis.  (See M.S. Thesis, "Parameter Independent *
     * Design Utilizing Scattering Parameters," Air Force Institute *
     * of Technology, June 1977).                                   *
     *                                                              *
     **************************************************************** }

{    ****************************************************************}

{    Declare arrays and variable types                               }
Const
  MaxOptParas = 6;
  MaxFreqs = 10;
  MaxModules = 10;
  MaxCards = 40;
  MaxDevices = 2;
Type
  Str255  =  String[2];
  complex   = record
                re           :  real;
                im           :  real;
              end;
  Arange    = array[1..2,1..2] of complex;
  Brange    = array[1..MaxOptParas] of real;
  Crange    = array[1..MaxFreqs] of real;
  Drange    = array[1..MaxOptParas] of Integer;
  ErrorFunc = record
                InvGain      :  real;
                Sens         :  real;
                RetLoss      :  real;
                FlatGain     :  real;
                Iexp         :  real;
                LogGain      :  real;
                ForGain      :  real;
                TruncLogGain :  real;
                Gain         :  real;
                GaindB       :  real;
              end;
  MType     = Char;
  CardDef   = record
                CardType     :  MType;
                Value        :  array[1..2] of real;
                Shunt        :  Integer;
                CombForm     :  Integer;
              end;  { CardDef record}
  Const
    Unity                    :  Complex = (re : 1.0; im : 0.0);
    MUnity                   :  Complex = (re :-1.0; im : 0.0);
    Zero                     :  Complex = (re:1e-15; im : 0.0);
    czero                    :  Complex = (re : 0.0; im : 0.0);
    RadDegree                :  Real = 0.01745329; {Pi/180.0}
    ZRef                     :  Real = 50.0;
    Vel                      :  Real = 3.0e8;

  Var
    Amod                     :  array[1..2,1..2,-1..MaxModules,1..MaxFreqs] of
                                 complex;
```

```
    Amps                    :  array[1..2,1..2] of complex;
    CSMA                    :  Arange;
    CSMB                    :  Arange;
    Gamma                   :  array[1..MaxFreqs,1..2] of complex;
    G                       :  Arange;
    F                       :  Arange;
    FC                      :  Arange;
    GC                      :  Arange;
    Funct                   :  array[1..MaxFreqs] of ErrorFunc;
    CardNo                  :  Integer;
    sconst,stol             :  complex;
    Module                  :  array[-1..MaxModules,1..MaxFreqs] of Arange;
    Card                    :  array[1..MaxCards] of CardDef;
    Freq                    :  Crange;
    P,Q,Qh,D                :  Brange;
    Ql                      :  Brange;
    Qu                      :  Brange;
    Np                      :  array[1..MaxOptParas,1..2]  of Integer;
    A,B                     :  array[1..MaxOptParas,1..MaxOptParas] of real;
    Gs                      :  Crange;
    Gdb                     :  Crange;
    Ice                     :  Crange;
    ty1,ty2                 :  complex;
    i,j,jopt,K              :  Integer;
    Nfreq                   :  Integer;
    Dependent               :  Boolean;
    Temp                    :  Char;
    Opt                     :  Boolean;
    FreqChk                 :  Boolean;
    Sensitivity             :  Boolean;
    Cards                   :  Integer;
    NCardsConst             :  Integer;
    ID                      :  Integer;
    Term                    :  Char;
    IPrint                  :  Char;
    NoDevice                :  Byte;
    ModNo                   :  array[1..MaxDevices] of Byte;
    OptPara                 :  Integer;
    Kount                   :  Integer;
    Search                  :  Integer;
    Maxcall                 :  Integer;
    Status                  :  Integer;
    CardCt,DevCt            :  Integer;
    IA                      :  Byte;
    NF                      :  Integer;
    Zo,E                    :  real;
    ObjF                    :  real;
    runprog                 :  Boolean;
    S21                     :  array[1..MaxFreqs] of real;
    IE                      :  Integer;
    Dev                     :  Text;

    isens,esens             :  Integer;
    iisens,jisens           :  Integer;
    iesens,jesens           :  Integer;
    SensMod,SensCt          :  Integer;
    Si,Se                   :  array[1..3] of complex;
    asens,bsens,dsens       :  array [1..MaxFreqs] of complex;
    psens,plsens            :  complex;
    Mpsens                  :  array [1..MaxFreqs] of real;
    SensQ,SensQ2            :  Boolean;
    SensQl                  :  array[1..2] of Char;

{ The following variables are defined for the PALROS procedure }
    NumSearch,Iternum       : integer;
    FnCountMax              : integer;
    divd                    : real;
    ItPrint                 : integer;
    Number3Up               :  Boolean;
    PBest,Qdelta,Dist       :  Brange;
    NumRandom,MaxRandom     :  Integer;
    Err,DistMin,Distance    :  Real;
    Fmid,Fold,Fmin,Dnew     :  Real;
    Dmid,Dold,Dmin          :  Real;
    Xn,EE,StepStart,Step    :  Real;
```

```
    KountMax,NSearch          :  Integer;
    It,KK,NN                  :  Integer;

{--------------------ReadReal-----------------------------}

procedure ReadReal(icol:byte ; var Number:real );
  begin
    GotoXY(icol,whereY);  Write(Number);
    GotoXY(icol,whereY);  repeat until keypressed;  clreol;
    Readln(Number);  GotoXY(icol,whereY-1);  Writeln(Number);
  end;

{----------------------Cabs-------------------------------}

Function Cabs( CNum: complex) : real;
{ Complex absolute value. }
 begin
   Cabs := SQRT(SQR(CNum.re) + SQR(CNum.im));
 end;  {Cabs}

{----------------------Alog10-----------------------------}

Function Alog10( Num: real) : real;
{ Log base 10. }
 begin
   If Num<1e-30 then Num:=1e-30;
   Alog10 := ln(Num)/ln(10);
 end;  {Alog10}

{----------------------Cadd-------------------------------}

Procedure Cadd( CNum1,CNum2:complex ; var Sum: complex);
{ This procedure computes the sum of two complex numbers. }
 begin
   Sum.re := CNum1.re + CNum2.re ;
   Sum.im := CNum1.im + CNum2.im ;
 end;  {Cadd}

{----------------------Csub-------------------------------}

Procedure Csub( CNum1,CNum2:complex ; var Diff: complex);
{ This procedure computes the difference of two complex numbers }
 begin
   Diff.re := CNum1.re - CNum2.re ;
   Diff.im := CNum1.im - CNum2.im ;
 end;  {Csub}

{----------------------Cmult------------------------------}

Procedure Cmult( CNum1,CNum2:complex ; var Prod: complex);
{ This procedure computes the product of two complex numbers }
 var
   Temp:  real;
 begin
   Temp := (CNum1.re*CNum2.re)-(CNum1.im*CNum2.im) ;
   Prod.im := (CNum2.re*CNum1.im)+(CNum1.re*CNum2.im) ;
   Prod.re := Temp;
 end;  {Cmult}

{----------------------Cdiv-------------------------------}

Procedure Cdiv( CNum1,CNum2:complex ; var Quot: complex);
{ This procedure computes the quotient of two complex numbers }
 begin
   CNum2.im := -CNum2.im;
   Cmult (CNum1,CNum2,CNum1);
   CNum2.re := sqr(CNum2.re) + sqr(CNum2.im);
   If CNum2.re= 0.0 then CNum2.re:= 1e-15;
   quot.re :=  CNum1.re / CNum2.re;
   quot.im :=  CNum1.im / CNum2.re;
 end;  {Cdiv}

{----------------------CDMA-------------------------------}
```

```
Procedure  CDMA( B:Arange; U,V,W: complex ; var A:Arange);

 var
   I,J: integer;
 Begin
   For I:= 1 to 2 do
     For J:= 1 to 2 do
       Cmult(U,B[I,J],A[I,J]);
   Cadd(A[1,1],V,A[1,1]);
   Cadd(A[2,2],W,A[2,2]);
 end;  {CDMA}

{----------------------CMAS--------------------------------}

Procedure  CMAS( A,B: Arange; M: real; var D: Arange);
{ Compute the sum or difference of two matrices where
     M=+1 is for sum and M=-1 is for difference. }
 var
   I,J: integer;
   MC:  complex;
 Begin
   MC.re := M ; MC.im := 0.0;
   For J:= 1 to 2 do
     For I:= 1 to 2 do
      begin
        Cmult(MC,B[I,J],D[I,J]);
        Cadd(A[I,J],D[I,J],D[I,J]);
      end;
 end;  {CMAS}

{----------------------CMI---------------------------------}

Procedure CMI( A: Arange; var B:Arange; var IE: integer);
{ Compute complex inverse B of matrix A. }
 var
   D,C: complex;
 Begin
   Cmult(A[1,2],A[2,1],D);
   Cmult(A[1,1],A[2,2],C);
   Csub(C,D,C);
   If (abs(C.re) =0.0) and (abs(C.im) = 0.0) then
    begin
      IE := 1;
      Writeln( '  singular matrix ');
    end
   else
    begin
      Cdiv(A[2,2],C,D);
      Cdiv(A[1,1],C,B[2,2]);
      B[1,1] := D;
      C.re := -C.re;
      C.im := -C.im;
      Cdiv(A[1,2],C,B[1,2]);
      Cdiv(A[2,1],C,B[2,1]);
    end;
  end;  {CMI}

{----------------------CMM---------------------------------}

Procedure CMM( A,B: Arange; var C: Arange);
{ Multiply two complex matrices A and B. }
 var
   temp,D,E,F : complex;
 Begin
   Cmult(A[1,2],B[2,1],temp);
   Cmult(A[1,1],B[1,1],D);
   Cadd(temp,D,D);
   Cmult(A[1,2],B[2,2],temp);
   Cmult(A[1,1],B[1,2],E);
   Cadd(temp,E,E);
   Cmult(A[2,2],B[2,1],temp);
   Cmult(A[2,1],B[1,1],F);
   Cadd(temp,F,F);
   Cmult(A[2,2],B[2,2],temp);
   Cmult(A[2,1],B[1,2],C[2,2]);
```

```
   Cadd(C[2,2],temp,C[2,2]);
   C[1,1] := D;
   C[1,2] := E;
   C[2,1] := F;
 end;  {CMM}

{----------------------Xform-------------------------------}

Procedure Xform( var S: Arange; A,B,C,D: complex; var IE: integer);
{ Translate between parameter sets:  H, Y, Z, G to S              }
 var
   S1,S2: Arange;
   Y:     complex;
 Begin
   Cmult(A,S[1,1],S1[1,1]);
   S1[1,1].re:=1.0+S1[1,1].re;
   Cmult(A,S[1,2],S1[1,2]);
   Cmult(B,S[2,1],S1[2,1]);
   Cmult(B,S[2,2],S1[2,2]);
   S1[2,2].re:=1.0+S1[2,2].re;
   Y.re := 2.0;    Y.im := 0.0;
   Csub(Y,S1[1,1],S2[1,1]);
   Csub(Y,S1[2,2],S2[2,2]);
   Y.re := -1.0;
   Cmult(Y,S1[1,2],S2[1,2]);
   Cmult(Y,S1[2,1],S2[2,1]);
   CMI(S1,S1,IE);
   CMM(S2,S1,S);
   Cmult(C,S[1,1],S[1,1]);
   Cmult(C,S[1,2],S[1,2]);
   Cmult(D,S[2,1],S[2,1]);
   Cmult(D,S[2,2],S[2,2]);
 end;  {Xform}

{----------------------GenS--------------------------------}

Procedure GenS( ID2: integer;  Zref,Freq,Zo,E: real;
                var S: Arange;  V: real);

 var
   Po,Temp,Q,Y,P,A,B,C : Complex;
   X,BL,qr             : real;

{- - - - - - - - - - - GenS- - - - - - - - - - - - - - - - }

 procedure GS1;
  begin
    Q.re := 1.0;  Q.im := 0.0;
    If (E=1) then Q.re:= -1.0;
    Cdiv(P,Po,P);
    Y.re := 2.0;  Y.im := 0.0;
    Cadd(P,Y,C);
    Y.re := 1.0;
    Cdiv(Y,C,C);
    Cmult(C,Q,S[1,1]);
    Cmult(S[1,1],P,S[1,1]);
    S[2,2] := S[1,1];
    Y.re := 2.0;
    Cmult(C,Y,S[2,1]);
    S[1,2] := S[2,1];
  end;  {GS1}

{- - - - - - - - - - - GenS- - - - - - - - - - - - - - - - }

 procedure GS2;
  begin
    S[1,1].re := -1.0;
    S[1,1].im := 0;
    S[2,2].re := -1.0;
    S[2,2].im := 0;
    S[1,2].re := 0;
    S[1,2].im := 0;
    S[2,1].re := 0;
    S[2,1].im := 0;
  end;  {GS2}
```

```
{- - - - - - - - - - - - GenS- - - - - - - - - - - - - - - - - - }

 procedure GS3;
  begin
    qr := 1.0;
    if (E=1) then qr := -1.0;
    S[1,1].re := (qr+1.0)/2.0;
    S[1,1].im := 0.0;
    S[2,2] := S[1,1];
    S[1,2].re := (1.0-qr)/2.0;
    S[1,2].im := 0.0;
    S[2,1] := S[1,2];
  end;  {GS3}

{- - - - - - - - - - - - GenS- - - - - - - - - - - - - - - - - - }

 procedure GS4;
  begin
    If (E = 2) then GS1
    else begin
       If (abs(P.re) =0) and (abs(P.im)=0) then
         GS2
       else begin
          Y.re := 1.0;Y.im := 0.0;
          Cdiv(Y,P,P);
          Cdiv(Y,Po,Po);
          GS1;
        end;
     end;
  end;  {GS4}

{- - - - - - - - - - - - GenS- - - - - - - - - - - - - - - - - - }

 procedure GS;
  begin
    X:= 2*Pi*Freq*Zo;
    If (x=0.0) and (ID2 =2) then GS3
    else
     begin
       If (ID2 = 2) then X := -1/X;
       P.re := 0;
       P.im := X;
       GS4;
     end;
  end;  {GS}

{- - - - - - - - - - - - GenS- - - - - - - - - - - - - - - - - - }

 Begin
   Po.re := Zref;
   Po.im := 0;
   Case ID2 of
       1:  begin
             P.re := Zo;
             P.im := 0;
             GS4;
           end;
    2..3:  GS;
       4:  begin
             BL := E*2*Pi*Freq/V;
             A.re := 0;
             A.im := (sqr(Zo)-sqr(Zref))*sin(BL);
             B.re := 2*Zo*Zref*cos(BL);
             B.im := (sqr(Zo)+sqr(Zref))*sin(BL);
             Cdiv(A,B,S[1,1]);
             S[2,2] := S[1,1];
             Y.re := 2*Zo*Zref;Y.im := 0.0;
             Cdiv(Y,B,S[1,2]);
             S[2,1] := S[1,2];
           end;
   end;  {Case}
 end;  {GenS}

{---------------- FILES (as an include file) --------------}
```

```
{$I FILES}

{--------------------NETCO (as an include file)-----------}

{$I NETCO}

{----------------------Anal--------------------------------}

Procedure Anal(D: Arange);

 var
   Det,Am,An,Gms,Gml,Y,Z,T:      complex;
   GaMax,U,G1,G2,G0,GuMax,KStab,
   B1,B2,Sn,Num,K,k2,Q:              real;

{- - - - - - - - - - - Anal- - - - - - - - - - - - - - - - - }

 procedure printvar ;
  begin
    writeln;
    writeln(' G1 = ',G1:9,' dB    G2 = ',G2:9,' dB    G0 = ',G0:9,' dB');
    writeln('   u = ',    u:9,'                    K = ',KStab:9);
    Writeln('   det = ',Det.re:10,'  +j ',Det.im:10);
    writeln('   GuMax = ',gumax:9,' dB           GaMax = ',gamax:9,' dB');
    writeln('Source Gamma = ',gms.re:9,' +j ',gms.im:9);
    writeln('  Load Gamma = ',gml.re:9,' +j ',gml.im:9);
    GotoXY(1,24); write('Press any Key to Continue'); repeat until KeyPressed;
  end;  {printvar}

{- - - - - - - - - - - Anal- - - - - - - - - - - - - - - - - }

 Begin
   G1 := 1/(1-sqr(Cabs(D[1,1])));
   G2 := 1/(1-sqr(Cabs(D[2,2])));
   Cmult(D[1,2],D[1,1],Y);
   Cmult(Y,D[2,1],Y);
   Cmult(Y,D[2,2],Y);
   u := Cabs(Y)*G1*G2;
   G1 := 10*Alog10(G1);
   G2 := 10*Alog10(G2);
   G0 := 20*Alog10(Cabs(D[2,1]));
   GuMax := G0 + G1 + G2;
   Cmult(D[1,1],D[2,2],Y);
   Cmult(D[2,1],D[1,2],Z);
   Csub(Y,Z,Det);
   Cmult(D[1,2],D[2,1],Z);
   if Cabs(Z)<>0.0 then
    KStab := (1. + Sqr(Cabs(Det)) - Sqr(Cabs(D[1,1]))
          - Sqr(Cabs(D[2,2])))/(2.*Cabs(Z))
   else KStab:= 10000.0;
   GaMax  := 0;
   Gms.re := 0;
   Gms.im := 0;
   Gml.re := 0;
   Gml.im := 0;

   If (KStab > 1.0) then
    begin
      B1 := 1 + sqr(Cabs(D[1,1])) -Sqr(Cabs(D[2,2]))
          - Sqr(Cabs(Det));
      B2 := B1 + 2.0*(Sqr(Cabs(D[2,2])) - Sqr(Cabs(D[1,1])));
      If (B1 < 0 ) then k := -1.0 else k := 1.0;
      If (B2 < 0 ) then k2 := -1.0 else k2 := 1.0;
      Sn := Abs(Sqrt(KStab*KStab-1));
      Sn := k*Sn;
      Cmult(D[2,1],D[1,2],Y);
      D[2,2].im := -1*D[2,2].im;
      Cmult(D[2,2],Det,Am);
      Csub(D[1,1],Am,Am);
      D[1,1].im := -1*D[1,1].im;
      D[2,2].im := -1*D[2,2].im;
      Cmult(D[1,1],Det,An);
      Csub(D[2,2],An,An);
      Num := ( B1 - k*sqrt( sqr(B1) - 4.0*sqr(Cabs(Am)) ) ) / 2.0 ;
```

```
      Gms.re := Num;
      Gms.im := 0;
      Cdiv(Gms,Am,Gms);
      Num := ( B2 - K2*sqrt( sqr(B2) - 4.0*sqr(Cabs(An)) ) ) / 2.0 ;
      Gml.re := Num;
      Gml.im := 0;
      Cdiv(Gml,An,Gml);
      Cdiv(D[2,1],D[1,2],Z);
      Y.re := KStab - Sn;
      Cmult(Z,Y,Z);
      GaMax := 10*Alog10(Cabs(Z));
      if Cabs(D[1,2])= 0.0 then
       GaMax := GuMax;
    end;

   printvar;
 end;  {Anal}

{---------------PALROS (as an include file)---------------}

{$I PALROS}

{-----------------------------------------------------------}

procedure PrintResults;
  begin
        For i:= 1 to NCardsConst do
          begin
            With Card[i] do
              begin
                Writeln(Lst,' Card # ',i,' = ');
                Writeln(Lst,CardType,Value[1],Value[2],'  ',Shunt);
              end;
          end;

         For NF:= 1 to Nfreq do
          begin
            Writeln(Lst,' ');
            Writeln(Lst,'  Final S-parameters = ');
            For i:= 1 to 2 do
            For j:= 1 to 2 do
            Writeln(Lst,'S[',i,j,']= ',Amod[i,j,1,NF].re:10,' +j',
             Amod[i,j,1,NF].im:10);
            Writeln(Lst,' ');
            Writeln(Lst,' ObjF= ',ObjF:9,'    | sigma |= ',Mpsens[NF]:9);
            Writeln(Lst,' a= ',asens[NF].re:9,' +j ',asens[NF].im:9);
            Writeln(Lst,' b= ',bsens[NF].re:9,' +j ',bsens[NF].im:9);
            Writeln(Lst,' d= ',dsens[NF].re:9,' +j ',dsens[NF].im:9);
            Writeln(Lst,' ');
            Writeln(Lst,'   S21= ',S21[NF]:10,' dB   F= ',Freq[NF]:9,' Hz');
           end;
  end; {PrintResults}

{----------------------FileResults----------------------------}

procedure FileResults;
  begin
        For i:= 1 to NCardsConst do
          begin
            With Card[i] do
              begin
                Writeln(Dev,' Card # ',i,' = ');
                Writeln(Dev,CardType,Value[1],Value[2],'  ',Shunt);
              end;
          end;
        If OptPara<>0 then
          begin
            Writeln(Dev,' ');
            For i:= 1 to OptPara do
              Writeln(Dev,' P',i,' = ',P[i]:9);
          end;

        For NF:= 1 to Nfreq do
          begin
            Writeln(Dev,' ');
```

```
            Writeln(Dev,'  Final S-parameters = ');
            For i:= 1 to 2 do
            For j:= 1 to 2 do
            Writeln(Dev,'S[',i,j,']= ',Amod[i,j,1,NF].re:10,' +j',
             Amod[i,j,1,NF].im:10);
            Writeln(Dev,' ');
            Writeln(Dev,' ObjF= ',ObjF:9,'    | sigma | = ',Mpsens[NF]:9);
            Writeln(Dev,' a= ',asens[NF].re:9,' +j ',asens[NF].im:9);
            Writeln(Dev,' b= ',bsens[NF].re:9,' +j ',bsens[NF].im:9);
            Writeln(Dev,' d= ',dsens[NF].re:9,' +j ',dsens[NF].im:9);
            Writeln(Dev,' ');
            Writeln(Dev,'   S21= ',S21[NF]:10,' dB   F= ',Freq[NF]:9,' Hz');
          end;
  end; {FileResults}

{-------------------------------------------------------------}

procedure FinalData;
var
 Filename     : String[7];

  begin
    Writeln;
    Write('  Would you like a hardcopy of program results (Y/N)? ');
    Read(KBD,Temp);
    If Temp in ['Y','y'] then
        PrintResults;
    Writeln;
    Writeln;
    Write('  Would you like to write results to a file (Y/N)? ');
    Read(KBD,Temp);
    If Temp in ['Y','y'] then
      begin
        Writeln;
        Writeln('  Disk file is named "RESULTS" '); Writeln;
        Filename:= 'RESULTS';
        Assign(Dev,Filename);
        Rewrite(Dev);
        FileResults;
        Close(Dev);
      end;

    Writeln;  Writeln;
    Write('  Would you like to run the program again (Y/N)? ');
    Read(KBD,Temp);  Writeln;
    If Temp in ['N','n'] then runprog:= False;
  end; {FinalData}

{-------------------------------------------------------------}

procedure ErrorCard;

var
   IRR    : Integer;

  begin
    IRR:= CardCt -1;
    Writeln('  Error card = ',IRR);
    Writeln('  Last S-parameters = ');
    For i:= 1 to 2 do
    For j:= 1 to 2 do
    Writeln('S[',i,j,']= ',Amod[i,j,1,NF].re:9,' + j',Amod[i,j,1,NF].im:9);
    Writeln(' S21 = ',S21[NF]:10,' dB   F= ',Freq[NF]:9,' Hz   NFR= ',Search:9);
    For i:=1 to CardNo do
      begin
        With Card[i] do
          begin
            Writeln(CardType,' ',Value[1]:9,' ',Value[2]:9,' ',Shunt);
          end;
      end;
    Halt;
  end;  {ErrorCard}

{-------------------------------------------------------------}
```

```
procedure IJset;

  var      IJ      : integer;

  begin
   Mpsens[NF]:= 0.0;
   asens[NF]:= czero ;  bsens[NF]:= czero ;  dsens[NF]:= czero;
   SensCt:= 3;
   If SensQ then SensQ2:= True else SensQ2:= False;
   If SensQ2= True then begin
    SensCt:= 0;
    IJ:= 0;
    For i:=1 to 2 do
      For j:=1 to 2 do
        begin
          IJ:= IJ+1;
          If IJ= isens then
            begin
              iisens:= i;
              jisens:= j;
            end;
          If IJ= esens then
            begin
              iesens:= i;
              jesens:= j;
            end;
        end;
      Si[3]:= Module[SensMod,NF,iisens,jisens];
   end; {If SensQ2}
  end;  { IJset }

{-------------------------------------------------------------}

procedure SensSet;

  begin
   If SensQ2= True then
    begin
     SensCt:= SensCt + 1;
     Case SensCt of
      1:  begin
            If (Si[3].re<>0.0) or (Si[3].im<>0.0) then Si[1]:= czero
              else Si[1]:= Unity;
            Module[SensMod,NF,iisens,jisens]:= Si[1];
          end;
      2:  begin
            If (Si[3].re<>0.0) or (Si[3].im<>0.0) then
              begin
                Si[2].re:= Si[3].re/2.0;
                Si[2].im:= Si[3].im/2.0;
              end
              else Si[2]:= MUnity;
            Module[SensMod,NF,iisens,jisens]:= Si[2];
          end;
      3:  Module[SensMod,NF,iisens,jisens]:= Si[3];
     end;
    end; {If SensQ2}
  end; { SensSet }

{-------------------------------------------------------------}

procedure SensSave;
  begin
    If SensQ2=True then
      begin
        Se[SensCt]:= Amod[iesens,jesens,1,NF];
        If SensCt= 3 then SensQ2:= False;
      end;
  end;

{-------------------------------------------------------------}

procedure SensCalc;

var       num,den     : complex;
```

```
  begin

   If SensQ then begin
    If (Si[3].re<>0.0) or (Si[3].im<>0.0) then asens[NF]:= Se[1]
      else asens[NF]:= Se[3];
    If (Si[3].re<>0.0) or (Si[3].im<>0.0) then
      begin
        i:= 2;
        j:= 3;
      end
      else
        begin
          i:= 1;
          j:= 2;
        end;
    Csub(Si[j],Si[i],num);
    Cmult(asens[NF],num,num);
    Cmult(Se[i],Si[j],den);
    Csub(num,den,num);
    Cmult(Si[i],Se[j],den);
    Cadd(num,den,num);
    Csub(Se[i],Se[j],den);
    Cmult(den,Si[i],den);
    Cmult(den,Si[j],den);
    If (den.re=0.0) and (den.im=0.0) then
      begin
        dsens[NF]:= czero;
        Se[i]:= asens[NF];
      end
      else
       Cdiv(num,den,dsens[NF]);

    Cmult(dsens[NF],Si[i],num);
    Cadd(num,Unity,num);
    Cmult(num,Se[i],num);
    Csub(num,asens[NF],num);
    Cdiv(num,Si[i],bsens[NF]);

    Cmult(dsens[NF],Se[3],num);
    Csub(bsens[NF],num,num);
    Cmult(dsens[NF],Si[3],den);
    Cadd(Unity,den,den);
    Cdiv(num,den,plsens);
    Cmult(plsens,Si[3],num);
    If (Se[3].re=0.0) and (Se[3].im=0.0) then Se[3]:= Unity;
    Cdiv(num,Se[3],psens);

    Mpsens[NF]:= Cabs(psens);
   end; { If SensQ }
  end; { SensCalc }

{-------------------------------------------------------------}

procedure StartConst;
 begin
   CardCt:=CardCt+1;
   If IE<>0 then ErrorCard;

   With Card[CardCt] do
     begin
       ID:=5;
       Case Cardtype of
           'R','C','L','X' :    ID:=1;
                     'A'   :    ID:=2;
                     'M'   :    ID:=3;
                     'D'   :    ID:=4;
       end;

     end;
 end;  {StartConst}

{-------------------------------------------------------------}

procedure ComponentCalc;
```

```
var
    ID2     : Integer;

 begin

   With Card[CardCt] do
     begin
       E:= Shunt;
       Case CardType of
         'R': ID2:= 1 ;
         'C': ID2:= 2 ;
         'L': ID2:= 3 ;
         'X': begin
                E:= Value[2];
                ID2:= 4 ;
              end;
       end;
       Zo:= Value[1];
       Case IA of
         0: begin
              GenS(ID2,Zref,Freq[NF],Zo,E,CSMA,Vel);
              IA:=1;
            end;
         1: begin
              GenS(ID2,Zref,Freq[NF],Zo,E,CSMB,Vel);
              IA:=2;
            end;
         2: begin
              ClrScr;
              Writeln('  Need a control card! ');
              ErrorCard;
            end;
       end;
     end;

 end; {ComponentCalc}

{-------------------------------------------------------------}

procedure ModuleCombine;

var
    I1,I2,J1,N,ID2     : Integer;
    CSMC,CSMD          : Arange;

 begin

   With Card[CardCt] do
     begin
       ID2:= CombForm;
       I1:= Trunc(Value[1]);
       I2:= Trunc(Value[2]);
       N:= 1;
       For i:= 1 to 2 do
       For j:= 1 to 2 do
         begin
           CSMC[i,j]:= Amod[i,j,I1,NF];
           CSMD[i,j]:= Amod[i,j,I2,NF];
         end;
       Case ID2 of
         6..7: begin
                 J1:= I1;
                 If J1>1 then J1:=1;
                 Case J1 of
                   -1: begin
                         Netco(IE,ID2,N,CSMB,CSMA,CSMB);
                         IA:= 2;
                       end;
                    0: begin
                         Netco(IE,ID2,N,CSMA,CSMB,CSMA);
                         IA:= 1;
                       end;
                    1: begin
                         Netco(IE,ID2,N,CSMC,CSMA,CSMD);
```

```
                    end;
                end;
              end; {case 6..7}
        1..5: begin
                J1:= 7;
                If (I1>0)  and (I2>0)  then J1:= 1;
                If (I1>0)  and (I2=-1) then J1:= 2;
                If (I1=-1) and (I2>0)  then J1:= 3;
                If (I1>0)  and (I2=0)  then J1:= 4;
                If (I1=0)  and (I2>0)  then J1:= 5;
                If (I1=-1) and (I2=0)  then J1:= 6;
                Case J1 of
                  1: begin
                       Netco(IE,ID2,N,CSMC,CSMD,CSMA);
                       IA:= 1;
                     end;
                  2: begin
                       Netco(IE,ID2,N,CSMC,CSMB,CSMB);
                       IA:= 2;
                     end;
                  3: begin
                       Netco(IE,ID2,N,CSMB,CSMD,CSMB);
                       IA:= 2;
                     end;
                  4: begin
                       IA:= 1;
                       Netco(IE,ID2,N,CSMC,CSMA,CSMA);
                     end;
                  5: begin
                       IA:= 1;
                       Netco(IE,ID2,N,CSMA,CSMD,CSMA);
                     end;
                  6: begin
                       IA:= 1;
                       Netco(IE,ID2,N,CSMB,CSMA,CSMA);
                     end;
                  7: begin
                       IA:= 1;
                       Netco(IE,ID2,N,CSMA,CSMB,CSMA);
                     end;
                end;        {case J1}
              end;       {case 1..5}
      end;          {case ID2}

      For i:= 1 to 2 do
      For j:= 1 to 2 do
        Amod[i,j,I2,NF]:= CSMD[i,j];
  end;        { With Card }

end;   { ModuleCombine }

{-------------------------------------------------------------}

procedure ModuleAssign;

var
    J1,J2,I1,I2    : Integer;

begin
  With Card[CardCt] do
    begin
      J1:= Trunc(Value[1]);
      J2:= Trunc(Value[2]);
      If J1>0 then J1:= 1;
      For I1:= 1 to 2 do
        For I2:= 1 to 2 do
          Case J1 of
            -1: begin
                  Amod[I1,I2,J2,NF]:= CSMB[I1,I2];
                  IA:= 1;
                end;
             0: begin
                  Amod[I1,I2,J2,NF]:= CSMA[I1,I2];
                  IA:= 0;
                end;
```

```
                1: begin
                     Amod[I1,I2,J2,NF]:= Amod[I1,I2,Trunc(Value[1]),NF];
                   end;
              end;
      end;     {With Card}

 end;  { ModuleAssign}

{-------------------------------------------------------------}

procedure ModMatrix;
  begin
    For i:= 1 to 2 do
    For j:= 1 to 2 do
    CSMA[i,j]:= Amod[i,j,1,NF];

    CMM(G,CSMA,CSMB);
    CDMA(CSMB,MUnity,Unity,Unity,CSMB);
    CMI(CSMB,CSMB,IE);
    CMM(CSMB,FC,CSMB);
    CMAS(CSMA,GC,-1.0,CSMA);
    CMM(CSMA,CSMB,CSMA);
    CMI(F,CSMB,IE);
    CMM(CSMB,CSMA,CSMA);

    For i:= 1 to 2 do
    For j:= 1 to 2 do
    Amod[i,j,1,NF]:= CSMA[i,j];

  end;   {ModMatrix}

{-------------------------------------------------------------}

procedure CompleteConst;

var
    J1,I1,I2    : Integer;

 begin
   If Term in ['N','n'] then ModMatrix;

     If Opt then
       begin
         With Card[CardCt] do
           begin
             J1:= Trunc(Value[1]);
             If J1>0 then J1:= 1;
             For I1:= 1 to 2 do
             For I2:= 1 to 2 do
               Case J1 of
                 -1: Amps[I1,I2]:= CSMB[I1,I2];
                  0: Amps[I1,I2]:= CSMA[I1,I2];
                  1: Amps[I1,I2]:= Amod[I1,I2,Trunc(Value[1]),NF];
               end;
           end;
       end;

 end;  { CompleteConst }

{-------------------------------------------------------------}

procedure ErrorSum;

var
    P1,P3,P4,P2    : real;

 begin

   If Opt then
     begin
       P1:=0.0;
       P3:=0.0;
       P4:=0.0;
       If Cabs(Amps[2,1])=0.0 then Amps[2,1]:= Zero;
       With Funct[NF] do
```

```
        begin
           If (InvGain<>0.0) or (Sens<>0.0) then
             P1:= 1.0/Sqr(Cabs(Amps[2,1]));
           P2:= Sqr(Cabs(Amps[2,1]));
           If (TruncLogGain<>0.0) or (LogGain<>0.0) then
             P3:= 10.0*Alog10(P2);
           If P3>GaindB then P4:= P3-GaindB;

           Err:= Err + P1*InvGain
             + FlatGain*(Exp(Ln(Abs(P2-Gain))*Iexp))
             + LogGain* Abs(P3-GaindB)
             + ForGain*P2 + TruncLogGain*P4
             + RetLoss*(Sqr(Cabs(Amps[1,1]))
             + Sqr(Cabs(Amps[2,2])));

           ObjF:= Err + Mpsens[NF]*Sens;
        end;  {With Func}
      end;  { If Opt }

  S21[NF]:= 20.0*Alog10(Cabs(Amod[2,1,1,NF]));

 end;  {ErrorSum}

{-------------------------------------------------------------------}

procedure CheckStatus;
  begin
    If NF=Nfreq then FreqChk:= False;
    If (OptPara=0) or (Status>=4) then Opt:= False;
 end; {CheckStatus}

{-----------------------------------------------------------}

procedure Optimize;

var
    IP,IC2,I2    : Integer;

 begin
   PALROS(Ql,Qu,Kount,Status,Search,OptPara,ObjF,P);
   For IP:= 1 to OptPara do
     begin
       IC2:= Np[IP,1];
       I2:= Np[IP,2];
       With Card[IC2] do
         begin
           Case I2 of
             1: Value[1]:= Exp(P[IP]);
             2: Value[2]:= P[IP];
           end;
         end;
     end;
 end;  { Optimize }

{-----------------------------------------------------------}

procedure OutputP;

var
    IP,IC2,I2    : Integer;

 begin
   If OptPara<>0 then
     begin
       If Status=5 then
         Writeln('  # of Function Requests Exceeded ');
       Writeln;
       Writeln('  PALROS parameters = ');
       For IP:= 1 to OptPara do
         begin
           I2:= Np[IP,2];
           If I2= 1 then P[IP]:= Exp(P[IP]);
         end;

       For i:= 1 to OptPara do
```

```
          begin
            Writeln(' P',i,' = ',P[i]:9);
          end;
        Writeln('   FINAL CONSTRUCTION ');
      end;
 end;  { OutputP }

{-------------------------------------------------------}

procedure OutputCards;
 begin

 For NF:= 1 to Nfreq do
  begin
   Writeln;
   Writeln('  S-params. = ');
   For i:= 1 to 2 do
   For j:= 1 to 2 do
   Writeln('S[',i,j,']= ',Amod[i,j,1,NF].re:9,' +j',Amod[i,j,1,NF].im:9);
   Writeln;
   Writeln(' ObjF= ',ObjF:9,'     | sigma |= ',Mpsens[NF]:9);
   If SensQ then
     begin
      Writeln(' a= ',asens[NF].re:9,' +j ',asens[NF].im:9);
      Writeln(' b= ',bsens[NF].re:9,' +j ',bsens[NF].im:9);
      Writeln(' d= ',dsens[NF].re:9,' +j ',dsens[NF].im:9);
     end;
   Writeln;
   Writeln('   S21= ',S21[NF]:10,' dB   F= ',Freq[NF]:9,' Hz');
  end;
 end; { OutputCards }

{-------------------------------------------------------}

procedure DeviceData;

var
    K2,J1,J,I1   : Integer;

 begin
   DevCt:= DevCt + 1;
   K2:= DevCt;
   J1:= ModNo[K2];
   If J1>1 then J1:=1;
       For J:=1 to 2 do
         For I1:=1 to 2 do
             Case J1 of
               -1: begin
                     CSMB[I1,J]:= Module[ModNo[K2],NF,I1,J];
                     IA:=2;
                   end;
                0: begin
                     CSMA[I1,J]:= Module[ModNo[K2],NF,I1,J];
                     IA:=1;
                   end;
                1: begin
                     Amod[I1,J,ModNo[K2],NF]:= Module[ModNo[K2],NF,I1,J];
                   end;
             end;
 end; { DeviceData}

{-------------------------------------------------------}

procedure TermMods;

var
   Fnum,Fden    : Complex;
   Fd1,Fd2      : real;

 begin
   IA:=0;  NF:=NF+1;
   If (Term in ['N','n']) then
     begin
       for i:=1 to 2 do
         begin
```

```
            G[i,i]:= Gamma[NF,i];
            GC[i,i].re:= G[i,i].re;
            GC[i,i].im:= -G[i,i].im;
            Csub(Unity,GC[i,i],Fnum);
            Csub(Unity,G[i,i],Fden);
            Fd1:= Cabs(Fden);
            Fd2:= Cabs(G[i,i]);
            Fd2:= Fd2*Fd2;
            Fd2:= 1.0 - Fd2;
            Fd2:= sqrt(Fd2);
            Fd2:= Fd1*Fd2;
            Fden.re:= Fd2;
            Fden.im:= 0.0;
            Cdiv(Fnum,Fden,F[i,i]);
            FC[i,i].re:= F[i,i].re;
            FC[i,i].im:= -F[i,i].im;
          end;  {for i}
      end;  {If }
 end;  {TermMods}

{-------------------------------------------------------}

procedure StartOpt;
 begin
   IE:=0;  Err:=0.0;  ObjF:=0.0; NF:=0;
 end;

{-------------------------------------------------------}

procedure InitializeOpt;
 begin
   Status:=-1;
   Writeln;
   write('Maximum Number of Optimization Routine Calls = '); readln(Maxcall);
   If Iprint= 'N' then Maxcall:= -Maxcall;
   write('   Number of Random Steps to Confirm Minimum = '); readln(Kount);
   write('    Quadratic wrt: 0 - Minimum, 1 - 1st 3 pts: '); readln(Search);
   write('                        Relative Parameter Error = '); readln(Err);
   writeln;
   PALROS(Ql,Qu,Kount,Status,Maxcall,OptPara,Err,P);
 end;

{----------------------InputData (as an include file)------}

{$I INDATA}

{----------------------Title-----------------------------}

procedure Title;
begin
  ClrScr;  GotoXY(1,8);
  writeln('    ***************************************************************');
  writeln('    *                                                             *');
  writeln('    *                  RF / Microwave CAD Program                 *');
  writeln('    *                                                             *');
  writeln('    *            by:   W. A. Davis and D. Blackburn               *');
  writeln('    *     Virginia Polytechnic Institute and State University     *');
  writeln('    *                                                             *');
  writeln('    ***************************************************************');
  GotoXY(1,24); write('Press any Key to Continue');  repeat until KeyPressed;
end;

{-------------------------------------------------------}

{----------------------Main------------------------------}

{     Main Program -

        The main program reads the initialization
     data in a prescribed format, as well as the device data, component
     data, and optimization data.  Then, the program will compute the S
     parameters for the two-port elements used to described the network,
     modifying the matrices if non-fifty ohm terminators are being used.
     Next, the device data for the frequency in use is entered and the
     program will costruct the network by the combination of the two-port
```

```
     S-parameter matrices.  Finally, the program will compute the objective
     function for the network and optimize the network using the Palros
     subroutine.
                                                                          }
{-------------------------------------------------------------}

BEGIN
  Title;
  runprog:= True;

  While runprog do
    begin
      InputData;
      If Opt then InitializeOpt;

      repeat
        StartOpt;
        FreqChk:=True;

        while FreqChk do
          begin
            TermMods;
            IJset;

            Repeat
                SensSet;
                CardCt:= 0;  DevCt:= 0;
                for Cards := 1 to NCardsConst do
                 begin
                   StartConst;
                   Case ID of
                     1:  ComponentCalc;
                     2:  ModuleCombine;
                     3:  ModuleAssign;
                     4:  DeviceData;
                   end;  {Case ID}
                 end;  {For Cards}
                CompleteConst;
                SensSave;
            until not(SensQ2);

            SensCalc;
            ErrorSum;
            CheckStatus;
          end;  {while FreqChk}

        if Opt then
          Optimize;
      until not(Opt);


      OutputCards;
      OutputP;
      FinalData;
    end;  { while runprog }

END.  {  MICROWAVE DESIGN  }
```

```
procedure FILES(var FilePath: Str255);
type
  Registers = record
                case integer of
                  0 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : INTEGER);
                  1 : (AL,AH,BL,BH,CL,CH,DL,DH : byte);
                end;
const
  Attribute = $01;  { $1f = all files}
  Blank     = ' ';
  SetDTA    = $1a;
  FindFirst = $4e;
  FindNext  = $4f;
var
  DTA    : Array[0..127] of Byte;
  Reg    : Registers;
  ASCIIZ : String[65];
  Count  : Integer;
{-------------------------------------------------------------}
procedure OneEntry;
var
  i,j : integer;
begin
  MsDos(Reg);
  if Reg.al=0 then
  begin
    if Count MOD 5 = 0 then writeln else write(':');
    Count:=Count + 1;
    j := 30;
    for i:=1 to 12 do
    if DTA[j] = 0 then
      write(Blank)
    else
      begin
      write(Chr(DTA[j]));
      j:=j+1;
      end;
  end; {if}
end; {OneEntry}
{-------------------------------------------------------------}
procedure GetArg;
begin
  write('Path? ');
  readln(ASCIIZ);
  FilePath:=ASCIIZ;
  if (length(ASCIIZ) = 0) or (ASCIIZ[Length(ASCIIZ)] in ['\',':']) then
    ASCIIZ := ASCIIZ + '*.*';
  ASCIIZ := ASCIIZ + Chr(0);
end; {GetArg}
{-------------------------------------------------------------}
procedure Initialize;
begin
  Count:=0;
  GetArg;
  writeln('Directory :  ',ASCIIZ);
  writeln;
  with Reg do
  begin
    ah:=SetDTA;
    ds:=Seg(DTA);
    dx:=Ofs(DTA);
    MsDos(Reg);
    ah:=FindFirst;
    al:=0;
    cx:=Attribute;
    ds:=Seg(ASCIIZ);
    dx:=Ofs(ASCIIZ[1]);
  end;
end; {Initialize}
{-------------------------------------------------------------}
begin
  Initialize;
  OneEntry;
  with Reg do while al=0 do
```

```
  begin
    ah:=FindNext;
    OneEntry;
  end;  {with/while}
  writeln;
end;
```

```
{----------------------NetCo-------------------------------}
Procedure Netco( var IE,ID2,N: integer;  A: Arange; B: Arange;
                 var CSM: Arange);
{                                                          }
 var
   C,P,P1,Z,Y,Q,A1,B1,R,R1:           Complex;
{- - - - - - - - - - - NetCo - - - - - - - - - - - - - - - }
 procedure My600(var A1,B1: complex);
  var
    Y : complex;
    result : Arange;
  begin
    A1.im:=0.0;  B1.im:=0.0;
    XForm(A,A1,B1,Unity,Unity,IE);
    XForm(B,A1,B1,Unity,Unity,IE);
    IE := 2*IE;
    CMAS(A,B,1.0,result);
    IE := 3*IE;
    XForm(result,Unity,Unity,A1,B1,IE);
    CSM:= result;
    IF (IE<>0) then
      writeln( '  IE = ', IE);
  end;  {My600}
{- - - - - - - - - - - NetCo - - - - - - - - - - - - - - - }
 procedure My900( var Q: complex);
  var
    P1, C1, Z : Complex;
  begin
    Cmult(A[1,1],A[2,2],R);
    Cmult(A[2,1],A[1,2],P);
    Csub(R,P,C);
    Csub(C,A[1,1],R);
    Cmult(R,Q,P);
    Csub(Unity,A[2,2],R);
    Cadd(R,P,P1);
    If (P1.re=0) and (P1.im=0)
     then begin
        CSM[1,1] := Q;
        CSM[2,2] := Q;
        CSM[2,1].re := 0;
        CSM[2,1].im := 0;
        CSM[1,2].re := 0;
        CSM[1,2].im := 0;
     end
    else begin
       Cmult(MUnity,P,C);
       Cadd(R,C,Z);
       Cdiv(Z,P1,P);
       Z.re :=2.0;
       Z.im := 0;
       Cadd(P,Z,R);
       Cdiv(Unity,R,C);
       Cmult(C,Q,R);
       Cmult(R,P,CSM[1,1]);
       CSM[2,2] := CSM[1,1];
       Cmult(C,Z,CSM[2,1]);
       CSM[1,2] := CSM[2,1];
     end;
  end;  {My900}
{- - - - - - - - - - - NetCo - - - - - - - - - - - - - - - }
 Begin
   Case ID2 of
      1:  begin
            Cmult(A[2,2],B[1,1],C);
            CSub(Unity,C,R);
            P1.re := abs(R.re);
            P1.im := abs(R.im);
            If (P1.re=0) and (P1.im= 0)
             then begin
                IE:=1;
                writeln('     Oscillating Junction   ');
             end
            else begin
                Cmult(A[2,1],B[1,1],R1);
```

```
                Cmult(A[1,2],R1,CSM[1,1]);
                Cdiv(CSM[1,1],R,R1);
                Cadd(R1,A[1,1],CSM[1,1]);
                Cmult(B[2,1],A[2,2],R1);
                Cmult(R1,B[1,2],CSM[2,2]);
                Cdiv(CSM[2,2],R,R1);
                Cadd(B[2,2],R1,CSM[2,2]);
                Cmult(A[2,1],B[2,1],R1);
                Cdiv(R1,R,CSM[2,1]);
                Cmult(A[1,2],B[1,2],R1);
                Cdiv(R1,R,CSM[1,2]);
              end;
          end;
      2:  begin
            A1 := MUnity;
            B1 := MUnity;
            My600(A1,B1);
          end;
      3:  begin
            A1 := Unity;
            B1 := Unity;
            My600(A1,B1);
          end;
      4:  begin
            A1 := MUnity;
            B1 := Unity;
            My600(A1,B1);
          end;
      5:  begin
            A1 := Unity;
            B1 := MUnity;
            My600(A1,B1);
          end;
      6:  begin
            Q:= MUnity;
            My900(Q);
          end;
      7:  begin
            Q:= Unity;
            My900(Q);
          end;
    end;  {Case}
  end;  {Netco}
```

```
{----------------------PalRos-----------------------------}
PROCEDURE PALROS (Ql,Qu : Brange;  var  Kount,Status : Integer;
                  Search,N : Integer; Fnew : real; var P : Brange);
{    Subroutine for finding the minimum of a function, using Rosenbrock's
     method modified by Palmer.  (See M. S. Thesis, D. Olmstead, U of Ill.
     1972.)  This procedure will "plot"  in memory several points of a
     function, perform a curve fitting operation and find the minimum
     of the curve.  Once a minima is found, the program will proceed
     in random directions to confirm the minimum point.
     ------------------------------------------------------------------
     P = the parameter vector
     Ql and Qu = lower and upper bound vectors for P
     N = number of parameters
     Kount = (1st)   - # random steps to confirm minimum
             (other) - # function requests made
     Fnew  = (1st)   - Parameter error
             (other) - function value
     Search= (1st)   - # function requests allowed
                       Sign = + Print during optimization
                            - No print
             (other) - search type
                       0 = Quadratic about minimum
                       1 = Quadratic about 1st 3 points
     Status should be negative 1 or 0 on the first entry.  It must not be reset
          on remaining calls.
          = 1  Compute Fnew, first request.
          = 2  Compute Fnew, not first request.
          = 3  Compute Fnew, quadratic minimum.
          = 4  All done, minimum in P.
          = 5  # function requests exceeded, P contains next variable set.   }
 const
   E1     :  Real= 50.0;
   E2     :  Real= 5.0;
 var
   U,divd                                    : real;
   Rote                                      : Boolean;
   Numerator,Denominator,Secderivative       : real;
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 procedure Set_up;  {This procedure sets up PALROS for operation.}
   begin
     DistMin := Fnew*Fnew;
     for i:=1 to N do
      begin
        Q[i] := P[i];
        for j:=1 to N do
          A[i,j] := 0.0;
        A[i,i] :=1.0;
        Qh[i] := Qu[i]-Ql[i];
        B[i,1] := sqrt(DistMin)*Qh[i];
      end;
     MaxRandom := Kount ;
     Kount := 1;
     KountMax := Abs(Search);
     ItPrint := Search;
     Xn := N;
     NumRandom := 0;
     EE := 2.0*Xn*DistMin;
     It := 0;
     Status := 1;
     If Search=0 then Status:=5;
     Step := 0.01;
     StepStart := Step;
   end;  {Set_up}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 procedure Init_print;
  const
    imax: integer = 5;
  begin
    writeln('********** PALROS **********');
    if ItPrint>0 then
     begin
       writeln(' Initial Parameter Values');
       for i:=1 to N do
        begin
```

```
          write(Q[i]);
          if i=imax then begin imax:=imax+5; writeln; end;
        end;
      writeln;
      writeln(' Initial Function Value F=',Fnew);
      writeln(' Lower Bounds - Ql');
      for i:=1 to N do
       begin
         write(Ql[i]);
         if i=imax then begin imax:=imax+5; writeln; end;
       end;
      writeln;
      writeln(' Upper Bounds - Qu');
      for i:=1 to N do
       begin
         write(Qu[i]);
         if i=imax then begin imax:=imax+5; writeln; end;
       end;
      writeln;
      writeln(' Final Accuracy of Each Parameter will be:');
      for i:=1 to N do
       begin
         write(B[i,1]);
         if i=imax then begin imax:=imax+5; writeln; end;
       end;
      writeln;
      writeln('-----------------------------');
    end;
  end;  {Init_Print}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - }
 procedure Init_Data;
  begin
    Fmin := Fnew;
    Fmid := Fnew;
    Fold := Fnew;
    Dnew := Step;
    Dmid := 0.0;
    KK := 1;
    NSearch := 1;
    Status := 2;
  end;  {Init_Data}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - }
 procedure StepP;
  begin;
    For i:=1 to N do
     begin
       P[i]:=Q[i]+A[i,jopt]*Dnew*Qh[i];
       if P[i] < Ql[i] then P[i] := Ql[i];
       if P[i] > Qu[i] then P[i] := Qu[i];
     end;
    Kount := Kount + 1;
    NSearch := NSearch + Search;
  end;  {StepP}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - }
 procedure StepQ;
  begin;
    for i:=1 to N do
     begin
       Q[i]:=Q[i]+A[i,jopt]*Dmin*Qh[i];
       if Q[i] < Ql[i] then Q[i] := Ql[i];
       if Q[i] > Qu[i] then Q[i] := Qu[i];
     end;
  end;  {StepQ}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - }
 procedure Iteration;
 {This procedure maintains track of iterations.}
  begin
    Rote:= False;
    IT := IT+1;
    If (NumRandom<2) and (ItPrint>0) then
     begin
       writeln('Iteration No.',IT,'   F=',Fmin,'  # Function calls =',Kount);
       for i:=1 to Trunc(N/5) do
         writeln(' X = ',Q[5*(i-1)+1],Q[5*(i-1)+2],Q[5*(i-1)+3],
                         Q[5*(i-1)+4],Q[5*(i-1)+5]);
```

```
      writeln('--------------------');
    end;
    jopt := 1;
    Init_Data;
    StepP;
  end;  {Iteration}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 procedure Exit;
 {This procedure resets the P vector at the end.}
  begin
    for i := 1 to N do
    P[i] := Q[i];
  end;  {Exit}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 procedure RandomSet;
 {This procedure sets up for random rotation.}
  begin
    If (N > 1) and (NumRandom< MaxRandom) then
      begin
         NumRandom := NumRandom + 1;
         Step := StepStart;
         If NumRandom = 1 then
          begin
            for i:=1 to N do
             begin
               for j:=1 to N do
                 A[i,j] := 0.0;
               A[i,i] := 1.0;
             end;
            if ItPrint>0 then
             writeln(' Axis Reset');
             Iteration;
          end
         else
          begin
            for i:=1 to N do
              D[i] := Random;
            NN:=NumRandom - 1;
            If ItPrint>0 then writeln('Random Rotation ',NN);
          end;
      end
      else
        begin
          Status:= 4;
          Writeln('  PALROS TERMINATED  ');
          Rote:= False;
          Exit;
        end;
  end;  {RandomSet}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 procedure Rotate;
 {This procedure rotates the search vector.}
  var
    M,L :integer;
  begin
    K := N;
    for j:=1 to N do
      if D[j]=0.0 then
        begin
          for i:=1 to N do
            B[i,K] := A[i,j];
            P[K] := 0.0;
            K := K-1;
        end;
    KK := K;
    for L:=1 to KK do
     begin
       K := 1;
       for i:=1 to N do
         if ABS(D[K])<ABS(D[i]) then K := i;
       P[L] := D[K];
       D[K] := 0.0;
       for i:=1 to N do
         B[i,L] := A[i,K];
     end;
```

```
    for K:=1 to N do
     begin
       D[K] := P[K];
       for i:=1 to N do
         A[i,K] := B[i,K];
     end;
    for i:=1 to N do B[i,N] := A[i,N]*D[N];
    P[N] := D[N]*D[N];
    for L:=2 to N do
     begin
       K:=N-L+1;
       M:=K+1;
       P[K] := P[M]+D[K]*D[K];
       for i:=1 to N do
         B[i,K] := B[i,M]+A[i,K]*D[K];
     end;
    for j:=2 to N do
     begin
       K:=N-j+2;
       M:=K-1;
       divd := sqrt(P[K]*P[M]);
       If divd<>0.0 then
          for i:=1 to N do
            A[i,k] := (D[M]*B[i,k]-A[i,M]*P[K])/divd;
     end;
    divd := sqrt(P[1]);
    if divd=0.0 then
      writeln('No variation in any direction!')
    else
    begin
      for i:=1 to N do
        A[i,1] := B[i,1]/divd;
      Iteration;
    end;
  end;  {Rotate}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - }
 procedure QuadReturn;
 {This procedure sets rotations as required.}
  begin
    jopt := jopt+1;
    if jopt <= N then
     begin
       Init_Data;
       StepP;
     end
    else
     begin
       Distance := 0.0;
       if Kount > KountMax then
        begin
          Status := 5;
          Exit;
        end
       else
         begin
            for i := 1 to N do
              Distance := Distance + D[i]*D[i];
              U:= sqrt(Distance);
            if U >= Step then
               if (U > 10.*Step) and (10.*Step <= 0.01) then Step :=10.*Step
            else
             Step := Step / 10.0;
                Rote:= True;
                if Distance> DistMin then
                   NumRandom := 0
                 else
                    RandomSet;
                While Rote do
                  begin
                    If N=1 then Iteration
                     else
                      begin
                        Rotate;
                        If divd=0.0 then RandomSet;
                      end;
```

```
                    end; { While Rote }
          end;
      end;
  end;  {QuadReturn}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 procedure Quad;
 {This procedure estimates the minimum of the function.}
  begin
    Numerator := Fold*(Dmid-Dnew)+Fmid*(Dnew-Dold)+Fnew*(Dold-Dmid);
    Denominator := (Dold-Dmid)*(Dold-Dnew)*(Dmid-Dnew);
    Secderivative := 2.0*Numerator/Denominator;
    if Secderivative>0.0 then
     begin
       D[jopt] := ( Fold*(Dmid*Dmid-Dnew*Dnew)+Fmid*(Dnew*Dnew-Dold*Dold)
              +Fnew*(Dold*Dold-Dmid*Dmid) ) / (Numerator+Numerator);
       Dnew := D[jopt];
       Status := 3;
       StepP;
     end
    else
     begin
       D[jopt] := Dmin; Fnew := Fmin;
       StepQ;
       QuadReturn;
     end;
  end;  {Quad}
{- - - - - - - - - - - PalRos- - - - - - - - - - - - - - - - }
 BEGIN
   CASE Status of
        5 :  Writeln(' # of Function Requests Exceeded ');
        4 :  Writeln(' WE WERE DONE LAST TIME!!');
        3 :  begin
               if Fnew < Fmin then
                begin
                  Fmin := Fnew;
                  for i := 1 to N do
                    Q[i] := P[i];
                end
               else
                begin
                  D[jopt] := Dmin;
                  Fnew := Fmin;
                  StepQ;
                end;
               QuadReturn;
             end;
        2 :  begin
               if Fnew < Fmid then
                begin
                  Dmin := Dnew;
                  Fmin := Fnew;
                  Status := 2;
                  if NSearch <= 2 then
                   begin
                     KK := 0;
                     Dold := Dmid;
                     Dmid := Dnew;
                     Fold := Fmid;
                     Fmid := Fnew;
                     if 0.01>(ABS(Dnew)) then
                       Dnew := Dnew*E1
                     else
                       Dnew := Dnew*E2;
                     StepP;
                   end
                  else
                   Quad;
                end
               else
                begin
                  Dmin := Dmid;
                  Fmin := Fmid;
                  Status := 2;
                  if KK<1 then Quad
                  else
```

```
                    begin
                      KK := -1;
                      Fold := Fnew;
                      Dold := Dnew;
                      Dnew := -Dnew;
                      StepP;
                    end;
                end;
            end;
        1 :  begin
               Init_print;
               jopt := 1;
               Init_Data;
               StepP;
             end;
    -1..0:  Set_up;
  end;   {  End of optimization routine PALROS }
end;
```

```
{----------------------InputData--------------------------}
Procedure InputData;
{ Input program data for processing }
 const
   Pathname       : String[2]= 'A:';
 type
   ParaType = Char;
   ModType = Char;
 var
   Data             :  Text;
   Filename         :  String[6];
   OldNoCards       :  Integer;
   tempr,n          :  real;
   CType,NoEdit     :  Char;
   edit             :  Boolean;
   TempS            :  String[1];
   ModC             :  Char;
   TempQ,TempQ2     :  array[1..MaxCards] of Char;
   PType            :  Paratype;
   MP               :  Char;
   MPb              :  Boolean;
   ModuleNo,IJ,I1   :  Integer;
   Max,Min          :  array[1..MaxOptParas] of real;

{- - - - - - - - - - - InputData - - - - - - - - - - - - - }

 procedure RI(var r,i:real);
  var
    tempr: real;
  begin
    tempr:=r*cos(i*Pi/180.0);
    i:=r*sin(i*Pi/180.0);
    r:=tempr;
  end;  {RI}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - }

 procedure MagP(var r,i:real);
  var
    tempr: real;
   begin
     tempr:= sqrt( sqr(r) + sqr(i) );
     If (r=0.0) and (i=0.0) then i:= 0.0;
     If (r=0.0) and (i<>0.0) then i:= 90.0;
     If r<>0.0 then i:= ArcTan( i/r )*180.0/Pi;
     If r<0.0 then i:= i + 180.0;
     r:= tempr;
   end;  {MagP}

{- - - - - - - - - - - InputData - - - - - - - - - - - - }

procedure MP_ri;
  begin
    for i:=1 to Nfreq do
     begin
      For j:= 1 to 2 do
      For K:= 1 to 2 do
       with Module[ModuleNo,i,j,k] do
           RI(re,im);
     end;
  end;  {MP_ri}

{- - - - - - - - - - - InputData - - - - - - - - - - - - }

procedure ri_MP;
  begin
    for i:=1 to Nfreq do
     begin
      For j:= 1 to 2 do
      For K:= 1 to 2 do
       with Module[ModuleNo,i,j,k] do
           MagP(re,im);
     end;
  end;  {ri_MP}
```

```
{- - - - - - - - - - - - - InputData - - - - - - - - - - - - - }

 procedure Trans(PType:ParaType ; var x:Arange);
  const
    a : Complex = (re:1.0;im:0.0);
    b : Complex = (re:-1.0;im:0.0);
  var
    ie:integer;
  begin
    case PType of
      'H','h':  XForm(x,a,a,b,a,ie);
      'Z','z':  XForm(x,a,a,b,b,ie);
      'Y','y':  XForm(x,a,a,a,a,ie);
      'G','g':  XForm(x,a,a,a,b,ie);
    end;
    if ie=1 then Write('SINGULAR TRANSFORMATION');
  end;  {Trans}

{- - - - - - - - - - - - - InputData - - - - - - - - - - - - - }

procedure Char_real(ModC: ModType; var   n: real);
 var     Result,j : Integer;
  begin
    If ModC in ['A','a'] then n:= 0.0;
    If ModC in ['B','b'] then n:= -1.0;
    If (Not(ModC in ['A','a'])) and (Not(ModC in ['B','b'])) then
      begin
        Val(ModC,Result,j);
        n:= Result;
      end;
  end;

{- - - - - - - - - - - - - InputData - - - - - - - - - - - - - }

procedure real_Char( var  ModC: ModType;  n: real);
 var     Result  : Integer;   TempS  : String[1];
  begin
    If n= -1.0 then ModC:='B';
    If n= 0.0 then ModC:= 'A';
    If (n<>-1.0) and (n<>0.0) then
      begin
        Result:= Trunc(n);
        Str(Result,TempS);
        ModC:= TempS[1];
      end;
  end;

{- - - - - - - - - - - - - InputData - - - - - - - - - - - - - }
 procedure DeviceRead;
  var
    analysis       :  Char;

  begin
    With Card[CardNo] do
      begin
        CardType:= 'D';
        Write('This Device is Module No.  ');
        real_Char(ModC,Value[1]);
        If edit then
         Write(ModC,'  ');
        temp:= ModC;  Readln(ModC);
        If ModC=#26 then ModC:= temp;
        Char_real(ModC,Value[1]);
        ModuleNo:= Trunc(Value[1]);
        NoDevice:=NoDevice+1;
        ModNo[NoDevice]:=ModuleNo;
        Write('Do you wish a device analysis? [Y/N]  ');  Readln(analysis);
        Case Shunt of
          1: PType:=  'S'  ;
          2: PType:=  'H'  ;
          3: PType:=  'Z'  ;
          4: PType:=  'Y'  ;
          5: PType:=  'G'  ;
        end;
```

```
        temp:= PType;
        Write('Input Parameter Type: S,H,Z,Y,G = ');
        If edit then Write(PType);  Readln(PType);
        If PType= #26 then PType:= temp;
        PType:= upcase(PType);
        Case PType of
          'S' : Shunt:=1 ;
          'H' : Shunt:=2 ;
          'Z' : Shunt:=3 ;
          'Y' : Shunt:=4 ;
          'G' : Shunt:=5 ;
        end;
        Value[2]:= 0.0;
      end; { With Card }

      Write('Is the data Mag and Phase? (Y/N): ');
      Readln(MP);
      MPb:=(MP in ['Y','y']);
      If (edit) and (MPb) then ri_MP;

    temp:= SensQl[NoDevice];
    Write('Is this device used in sensitivity calculations? (Y/N) ');
    If edit then Write(SensQl[NoDevice],'  ');
    Readln(SensQl[NoDevice]);
    If SensQl[NoDevice]= #26 then SensQl[NoDevice]:= temp;
    If SensQl[NoDevice] in ['Y','y'] then SensQ:= True else SensQ:= False;
    If SensQ then
      begin
        SensMod:= ModuleNo;
        Writeln;
        Writeln('      S11      1');
        Writeln('      S12      2');
        Writeln('      S21      3');
        Writeln('      S22      4');
        GotoXY(2,19);
        Write(' Enter # for internal (device) parameter ');
        If edit then Write(isens,'  ');
        Readln(isens);
        Write(' Enter # for external (system) parameter ');
        If edit then Write(esens,'  ');
        Readln(esens);
      end;

    for i:=1 to Nfreq do
     begin
       ClrScr; GotoXY(1,5);
       Writeln('Frequency = ',Freq[i],' Hz');

       For j:= 1 to 2 do
       For K:= 1 to 2 do
       with Module[ModuleNo,i,j,k] do begin
         Write(PType,'[',j,',',k,'] = ');
         If edit then Write(re,im,'  '); Readln(re,im);
         If MPb then RI(re,im);
       end;

       Trans(PType,Module[ModuleNo,i]);
     end;   { For 1 to Nfreq }

    if analysis in ['Y','y'] then
      For i:= 1 to NFreq do Anal(Module[ModuleNo,i]);
  end;  {DeviceRead}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - - }

 procedure ResistanceRead;
  begin
    With Card[CardNo] do
      begin
        CardType := 'R';
        Write('Initial Resistance Value = ');
        If edit then Write(Value[1],'  ');
        Readln(Value[1]);
        temp:= TempQ[CardNo];
        Write('Optimize Resistance? [Y/N] ');
```

```
        If edit then Write(TempQ[CardNo],'  ');
        Readln(TempQ[CardNo]);
        If TempQ[CardNo]= #26 then TempQ[CardNo]:= temp;

        if TempQ[CardNo] in ['Y','y'] then
         begin
           Opt:=True; OptPara:=OptPara+1;
           Write('    Max Resistance Value = ');
           If edit then Write(Max[OptPara],'  ');
           Readln(Max[OptPara]);
           Write('    Min Resistance Value = ');
           If edit then Write(Min[OptPara],'  ');
           Readln(Min[OptPara]);
           P[OptPara]:=ln(Value[1]);
           Qu[OptPara]:=ln(Max[OptPara]);
           if Min[OptPara]<=0.0 then Ql[OptPara]:=-20.0
             else Ql[OptPara]:=ln(Min[OptPara]);
           NP[OptPara,1]:=CardNo;
           NP[OptPara,2]:=1;
         end;

        Write('Enter 1 for Shunt, 2 for Series -- ');
        If edit then Write(Shunt,'  ');
        Readln(Shunt);
        Value[2]:= 0.0;
      end;
  end;   {ResistanceRead}

{ - - - - - - - - - - - InputData - - - - - - - - - - - - - }

 procedure CapacitanceRead;
  begin
    With Card[CardNo] do
      begin
        CardType := 'C';
        Write('Initial Capacitance Value = ');
        If edit then Write(Value[1],'  ');
        Readln(Value[1]);
        temp:= TempQ[CardNo];
        Write('Optimize Capacitance? [Y/N] ');
        If edit then Write(TempQ[CardNo],'  ');
        Readln(TempQ[CardNo]);
        If TempQ[CardNo]= #26 then TempQ[CardNo]:= temp;

        if TempQ[CardNo] in ['Y','y'] then
         begin
           Opt:=True; OptPara:=OptPara+1;
           Write('    Max Capacitance Value = ');
           If edit then Write(Max[OptPara],'  ');
           Readln(Max[OptPara]);
           Write('    Min Capacitance Value = ');
           If edit then Write(Min[OptPara],'  ');
           Readln(Min[OptPara]);
           P[OptPara]:=ln(Value[1]);
           Qu[OptPara]:=ln(Max[OptPara]);
           if Min[OptPara]<=0.0 then Ql[OptPara]:=-20.0
             else Ql[OptPara]:=ln(Min[OptPara]);
           NP[OptPara,1]:=CardNo;
           NP[OptPara,2]:=1;
         end;

        Write('Enter 1 for Shunt, 2 for Series -- ');
        If edit then Write(Shunt,'  ');
        Readln(Shunt);
        Value[2]:= 0.0;
      end;
  end;  {CapacitanceRead}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - }

 procedure InductanceRead;
  begin
    With Card[CardNo] do
      begin
        CardType := 'L';
```

```
        Write('Initial Inductance Value = ');
        If edit then Write(Value[1],'  ');
        Readln(Value[1]);
        temp:= TempQ[CardNo];
        Write('Optimize Inductance? [Y/N] ');
        If edit then Write(TempQ[CardNo],'  ');
        Readln(TempQ[CardNo]);
        If TempQ[CardNo]= #26 then TempQ[CardNo]:= temp;

        if TempQ[CardNo] in ['Y','y'] then
         begin
           Opt:=True; OptPara:=OptPara+1;
           Write('    Max Inductance Value = ');
           If edit then Write(Max[OptPara],'  ');
           Readln(Max[OptPara]);
           Write('    Min Inductance Value = ');
           If edit then Write(Min[OptPara],'  ');
           Readln(Min[OptPara]);
           P[OptPara]:=ln(Value[1]);
           Qu[OptPara]:=ln(Max[OptPara]);
           if Min[OptPara]<=0.0 then Ql[OptPara]:=-20.0
             else Ql[OptPara]:=ln(Min[OptPara]);
           NP[OptPara,1]:=CardNo;
           NP[OptPara,2]:=1;
         end;

        Write('Enter 1 for Shunt, 2 for Series -- ');
        If edit then Write(Shunt,'  ');
        Readln(Shunt);
        Value[2]:= 0.0;
       end;
  end; {InductanceRead}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - }

 procedure LineRead;
  begin
    With Card[CardNo] do
     begin
       CardType := 'X';
       Write('Initial Characteristic Impedance Value = ');
       If edit then Write(Value[1],'  ');
       Readln(Value[1]);
       temp:= TempQ[CardNo];
       Write('Optimize Characteristic Impedance? [Y/N] ');
         If edit then Write(TempQ[CardNo],'  ');
         Readln(TempQ[CardNo]);
         If TempQ[CardNo]= #26 then TempQ[CardNo]:= temp;

       if TempQ[CardNo] in ['Y','y'] then
        begin
          Opt:=True; OptPara:=OptPara+1;
          Write('    Max Characteristic Impedance Value = ');
          If edit then Write(Max[OptPara],'  ');
          Readln(Max[OptPara]);
          Write('    Min Characteristic Impedance Value = ');
          If edit then Write(Min[OptPara],'  ');
          Readln(Min[OptPara]);
          P[OptPara]:=ln(Value[1]);
          Qu[OptPara]:=ln(Max[OptPara]);
          if Min[OptPara]<=0.0 then Ql[OptPara]:=-20.0
           else Ql[OptPara]:=ln(Min[OptPara]);
          NP[OptPara,1]:=CardNo;
          NP[OptPara,2]:=1;
        end;

       Write('Initial Length Value(m) = ');
       If edit then Write(Value[2],'  ');
       Readln(Value[2]);
       temp:= TempQ2[CardNo];
       Write('Optimize Length? [Y/N] ');
       If edit then Write(TempQ2[CardNo],'  ');
       Readln(TempQ2[CardNo]);
       If TempQ2[CardNo]= #26 then TempQ2[CardNo]:= temp;
```

```
        if TempQ2[CardNo] in ['Y','y'] then
         begin
           Opt:=True; OptPara:=OptPara+1;
           Write('    Max Length Value(m) = ');
           If edit then Write(Qu[OptPara],'  ');
           Readln(Qu[OptPara]);
           Write('    Min Length Value(m) = ');
           If edit then Write(Ql[OptPara],'  ');
           Readln(Ql[OptPara]);
           P[OptPara]:=Value[2];
           NP[OptPara,1]:=CardNo;
           NP[OptPara,2]:=2;
         end;

        Shunt:= 0;
      end;
  end;  {LineRead}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - - }

 procedure ModuleComb;
  begin
    with Card[CardNo] do
     begin
       CardType := 'A';
       Write(' left module  = ');
       real_Char(ModC,Value[1]);
       If edit then
         Write(ModC,'  ');
       temp:= ModC;  Readln(ModC);
       If ModC=#26 then ModC:= temp;
       Char_real(ModC,Value[1]);

       Write(' right module = ');
       real_Char(ModC,Value[2]);
       If edit then
         Write(ModC,'  ');
       temp:= ModC;  Readln(ModC);
       If ModC=#26 then ModC:= temp;
       Char_real(ModC,Value[2]);
       Writeln;
       Writeln('  1 -  Cascade ');
       Writeln('  2 -  Z       ');
       Writeln('  3 -  Y       ');
       Writeln('  4 -  H       ');
       Writeln('  5 -  G       ');
       Writeln('  6 -  2-port shunt element, with left end treated as 1-port');
       Writeln('  7 -  2-port series element                               ');
       Writeln;
       Write(' Combination format number = ');
       If edit then Write(CombForm,'  ');
       Readln(CombForm);
       Shunt:= CombForm;
     end;
  end; {ModuleComb}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - - }

 procedure ModAssign;
  begin
    with Card[CardNo] do
     begin
       CardType:='M';
       Write(' subject module = ');
       real_Char(ModC,Value[1]);
       If edit then
         Write(ModC,'  ');
       temp:= ModC;  Readln(ModC);
       If ModC=#26 then ModC:= temp;
       Char_real(ModC,Value[1]);

       Write(' object module  = ');
       real_Char(ModC,Value[2]);
       If edit then
         Write(ModC,'  ');
```

```
        temp:= ModC;  Readln(ModC);
        If ModC=#26 then ModC:= temp;
        Char_real(ModC,Value[2]);
        Shunt:= 0;
      end;
  end;  {ModAssign}

{- - - - - - - - - - - - InputData - - - - - - - - - - - - - - - }

procedure FunctWrite;
  begin
    ClrScr;
    Writeln(' Objective Function  ');
    Writeln;
    Writeln('       =  InvGain / sqr| S21 |');
    Writeln;
    Writeln('          +  Sens * Mpsens ');
    Writeln;
    Writeln('          +  RetLoss * | sqr| S11 | + sqr| S22 | | ');
    Writeln;
    Writeln('                                                ( Iexp )');
    Writeln('          +  FlatGain * | sqr| S21 | - G | ');
    Writeln;
    Writeln('          +  LogGain * | 10*log(sqr| S21 |) - GdB | ');
    Writeln;
    Writeln('          +  ForGain * sqr| S21 | ');
    Writeln;
    Writeln('          +  TruncLogGain * max[ 0 , (10*log(sqr| S21 |) - GdB) ]');
    Writeln;  Writeln;
  end; {FunctWrite}

{----------------------  Input Data  -------------------------}

 procedure FunctionRead(var Funct:ErrorFunc);
  begin
    with Funct do begin
       FunctWrite; InvGain:=0; Sens:=0; FlatGain:=0; ForGain:=0;
       LogGain:= 0; TruncLogGain:=0; Gain:=0; GaindB:=0;
       Writeln('Frequency = ',Freq[i],'  Hz');
       Write('For Maximum Gain:                  ');  ReadReal(34,InvGain);
       Write('For Sensitivity:                   ');  ReadReal(34,Sens);
       Write('For | sqr|S11| + sqr|S22| |:       ');  ReadReal(34,RetLoss);
       Write('For Flat Gain:                     ');  ReadReal(34,FlatGain);
       if FlatGain<>0.0 then
        begin
          Iexp:=1;
          Write('   To power of Iexp = :            ');  ReadReal(34,Iexp);
        end
       else Iexp:=1.0;
       Write('For Flat Log Gain:                 ');  ReadReal(34,LogGain);
       Write('For Gain Minimization:             ');  ReadReal(34,ForGain);
       Write('For Truncated Log Gain Flatness: ');  ReadReal(34,TruncLogGain);
       Write('Actual Gain Desired (dB):          ');  ReadReal(34,GaindB);
       Gain:=exp((GaindB/10.0)*ln(10.0));
    end;  {with Funct}
  end;  {FunctionRead}

{--------------------------OutData----------------------------}

procedure OutData;
 var      l   : integer;

  begin
    Assign(Data,Filename);
    Rewrite(Data);
    Writeln(Data,Zref);
    Writeln(Data,Vel);
    Writeln(Data,Term);
    Writeln(Data,Nfreq);
    For i:= 1 to Nfreq do
      Writeln(Data,Freq[i]);
    if NOT(Term in ['Y','y']) then
      begin
        for i:=1 to Nfreq do
          begin
```

```
      Writeln(Data,Gamma[i,1].re);
      Writeln(Data,Gamma[i,1].im);
      Writeln(Data,Gamma[i,2].re);
      Writeln(Data,Gamma[i,2].im);
    end; {for i}
end; {if Term}

 OptPara:= 0; NoDevice:= 0;
 For i:= 1 to NCardsConst do
   With Card[i] do
    begin
     Case CardType of

     'D': begin
            NoDevice:= NoDevice + 1 ;
            Writeln(Data,CardType);
            Writeln(Data,ModNo[NoDevice]);
            Writeln(Data,SensQl[NoDevice]);
            If SensQl[NoDevice] in ['Y','y'] then
              begin
                Writeln(Data,isens);
                Writeln(Data,esens);
              end;

             for j:=1 to Nfreq do
              begin
               For k:= 1 to 2 do
               For l:= 1 to 2 do begin
                  Writeln(Data,Module[ModNo[NoDevice],j,k,l].re);
                  Writeln(Data,Module[ModNo[NoDevice],j,k,l].im);
                end;
              end;
            Writeln(Data,Shunt);
          end;

       'R','C','L': begin
                      Writeln(Data,CardType);
                      Writeln(Data,Value[1]);
                      Writeln(Data,TempQ[i]);
                      If TempQ[i] in ['Y','y'] then
                       begin
                         OptPara:=OptPara + 1;
                         Writeln(Data,Qu[OptPara]);
                         Writeln(Data,Ql[OptPara]);
                       end;
                      Writeln(Data,Shunt);
                     end;

     'X'    : begin
                Writeln(Data,CardType);
                Writeln(Data,Value[1]);
                Writeln(Data,TempQ[i]);
                If TempQ[i] in ['Y','y'] then
                 begin
                   OptPara:=OptPara + 1;
                   Writeln(Data,Qu[OptPara]);
                   Writeln(Data,Ql[OptPara]);
                 end;
                Writeln(Data,Value[2]);
                Writeln(Data,TempQ2[i]);
                If TempQ2[i] in ['Y','y'] then
                 begin
                   OptPara:=OptPara + 1;
                   Writeln(Data,Qu[OptPara]);
                   Writeln(Data,Ql[OptPara]);
                 end;
              end;

     'A'    :  begin
                 Writeln(Data,CardType);
                 Writeln(Data,Value[1]);
                 Writeln(Data,Value[2]);
                 Writeln(Data,CombForm);
               end;
```

```
            'M'    :  begin
                        Writeln(Data,CardType);
                        Writeln(Data,Value[1]);
                        Writeln(Data,Value[2]);
                      end;
           end;  {Case}
         end;
    Writeln(Data,' ');
    Close(Data);
  end;  {OutData}

{--------------------- InPrint ----------------------------}
procedure InPrint;
  begin
   ClrScr;
   Writeln('Reference Impedance (real):           ',Zref);
   Writeln('Transmission Line Velocity(m/s):      ',Vel);
   Writeln('Terminations matched to Zref? (Y/N)   ',Term);
   Writeln('The number of Freqs. are  ( <= ',MaxFreqs,' ) : ',Nfreq);
   for i:=1 to Nfreq do
      Writeln('  Frequency (Hz) No.',i,'= ',Freq[i]);

   if NOT(Term in ['Y','y']) then
    begin
      for i:=1 to Nfreq do
        begin
          Writeln(Freq[i],'  Input Reflection Coefficient (Mag, Phase) = ',
           Gamma[i,1].re,Gamma[i,1].im);
          Writeln(Freq[i],' Output Reflection Coefficient (Mag, Phase) = ',
           Gamma[i,2].re,Gamma[i,2].im);
        end;
    end;

    For i:= 1 to NoDevice do
     begin
       Writeln('This Device is Module No. ',ModNo[i],'  ');
       Write('Is this device used in sensitivity calculations? (Y/N) ');
       Writeln(SensQ1[i],'  ');
       If SensQ1[i] in ['Y','y'] then
         begin
           Writeln;
           Write(' # for internal (device) parameter ');
           Writeln(isens);
           Write(' # for external (system) parameter ');
           Writeln(esens);  Writeln;
         end;
       PType:= 'S';

       ri_MP;
       for i:=1 to Nfreq do
        begin
          Writeln('Frequency = ',Freq[i]:10,' Hz');
          For j:= 1 to 2 do
            For k:= 1 to 2 do begin
             with Module[ModuleNo,i,j,k] do begin
               Write(PType,'[',j,',',K,'] =  '); Writeln(re:9,'  ',im:9);
             end;
           end;
          Writeln;
        end;
      MP_ri;
     end;

   For i:= 1 to NCardsConst do
     begin
       With Card[i] do
       Writeln(i,'  ',CardType,Value[1],Value[2],'  ',Shunt);
     end;
 end; { InPrint }

{-------------------- In_data ----------------------------}

procedure In_data;
 Begin
   Assign(Data,Filename);
```

```
Reset(Data);
Readln(Data,Zref);
Readln(Data,Vel);
Readln(Data,Term);
Readln(Data,Nfreq);
for i:=1 to Nfreq do
   Readln(Data,Freq[i]);

if NOT(Term in ['Y','y']) then
 begin
   for i:=1 to Nfreq do
    begin
      Readln(Data,Gamma[i,1].re);
      Readln(Data,Gamma[i,1].im);
      Readln(Data,Gamma[i,2].re);
      Readln(Data,Gamma[i,2].im);
    end; {for i}
 end; {if Term}

 CardNo:=0; NoDevice:=0; OptPara:=0;
 Opt:=False;
 repeat
   NCardsConst:= CardNo;
   CardNo:=CardNo+1;
   Readln(Data,TempS);
   CType:=TempS[1];
   Case CType of

        'D'    : begin
                   With Card[CardNo] do
                     begin
                       CardType := CType;
                       Readln(Data,ModuleNo);
                       Value[1]:= ModuleNo;
                       Value[2]:= 0.0;
                       NoDevice:=NoDevice+1;
                       ModNo[NoDevice]:=ModuleNo;
                       Read(Data,SensQl[NoDevice]);
                       If SensQl[NoDevice] in ['Y','y'] then
                         begin
                           SensQ:= True;
                           SensMod:= ModuleNo;
                           Readln(Data,isens);
                           Readln(Data,esens);
                         end;

                       for i:=1 to Nfreq do
                        begin
                            For j:= 1 to 2 do
                            For k:= 1 to 2 do begin
                              Readln(Data,Module[ModuleNo,i,j,k].re);
                              Readln(Data,Module[ModuleNo,i,j,k].im);
                            end;
                        end;

                       Readln(Data,Shunt);
                     end;  { With Card }
                end;

             'R','C','L'   : begin
                               With Card[CardNo] do
                                 begin
                                  CardType := CType;
                                  Readln(Data,Value[1]);
                                  Value[2]:= 0.0;
                                  Readln(Data,TempQ[CardNo]);
                                  if TempQ[CardNo] in ['Y','y'] then
                                   begin
                                    Opt:=True; OptPara:=OptPara+1;
                                    Readln(Data,Qu[OptPara]);
                                    Readln(Data,Ql[OptPara]);
                                    Max[OptPara]:= exp(Qu[OptPara]);
                                    Min[OptPara]:= exp(Ql[OptPara]);
                                    P[OptPara]:=ln(Value[1]);
                                    NP[OptPara,1]:=CardNo;
```

```
                                        NP[OptPara,2]:=1;
                                      end;
                                    Readln(Data,Shunt);
                                  end;
                              end;

          'X'    :  begin
                      With Card[CardNo] do
                       begin
                         CardType := CType;
                         Readln(Data,Value[1]);
                         Readln(Data,TempQ[CardNo]);
                         if TempQ[CardNo] in ['Y','y'] then
                          begin
                            Opt:=True; OptPara:=OptPara+1;
                            Readln(Data,Qu[OptPara]);
                            Readln(Data,Ql[OptPara]);
                            Max[OptPara]:= exp(Qu[OptPara]);
                            Min[OptPara]:= exp(Ql[OptPara]);
                            P[OptPara]:=ln(Value[1]);
                            NP[OptPara,1]:=CardNo;
                            NP[OptPara,2]:=1;
                          end;
                         Readln(Data,Value[2]);
                         Readln(Data,TempQ2[CardNo]);
                         if TempQ2[CardNo] in ['Y','y'] then
                          begin
                            Opt:=True; OptPara:=OptPara+1;
                            Readln(Data,Qu[OptPara]);
                            Readln(Data,Ql[OptPara]);
                            P[OptPara]:=Value[2];
                            NP[OptPara,1]:=CardNo;
                            NP[OptPara,2]:=2;
                          end;
                          Shunt:= 0;
                       end;
                     end;

          'A'    :  begin
                      with Card[CardNo] do
                       begin
                         CardType := CType;
                         Readln(Data,Value[1]);
                         Readln(Data,Value[2]);
                         Readln(Data,CombForm);
                         Shunt:= CombForm;
                       end;
                     end;

          'M'    :   begin
                       with Card[CardNo] do
                        begin
                          CardType:=CType;
                          Readln(Data,Value[1]);
                          Readln(Data,Value[2]);
                          Shunt:= 0;
                        end;
                      end;

      end;  {Case}

    until CType=' ';
    Close(Data);
    MPb:= False;   OldNoCards:= NCardsConst;
 end;  {In_data}

{- - - - - - - - - - - InputData - - - - - - - - - - - - - }

procedure ReadFile;
   begin
    Files(Pathname);
    Writeln;
    Write('Enter name of file you wish to read from -- ');
    Readln(Filename);
    In_Data;
```

```
    InPrint;
    GotoXY(1,25);
    Write('Would you like to edit information? (Y/N) ');
    Read(KBD,Temp);  Writeln;
    If Temp in ['Y','y'] then
      begin
        edit:= True;
        NoEdit:= 'N';
      end;
   end;  {ReadFile}

{- - - - - - - - - - - - InputData - - - - - - - - - - - - - - }

procedure ClearA;
  begin
    For i:= 1 to 2 do
    For j:= 1 to 2 do
      CSMA[i,j]:= czero;
  end;

procedure ClearB;
  begin
    For i:= 1 to 2 do
    For j:= 1 to 2 do
      CSMB[i,j]:= czero;
    end;

{- - - - - - - - - - - - InputData - - - - - - - - - - - - - - }

Begin
 SensQ:= False;
 ClrScr; GotoXY(1,5);
 Write('Would you like to read data from a file (Y/N)? ');
 Read(KBD,Temp);  Writeln;
 NoEdit:= Temp;
 edit:= False;
 Zref:= 50;  Vel:= 3e8;  Term:= 'Y';  Nfreq:= 1;
 If Temp in ['Y','y'] then ReadFile;

 While NoEdit in ['N','n'] do
  begin
   ClrScr;
   Write('Enter Reference Impedance (real):           ');
   ReadReal(45,Zref);
   Write('Enter Transmission Line Velocity(m/s):      ');
   ReadReal(45,Vel);
   temp:= Term;
   Write('Are the terminations matched to Zref? (Y/N) ',Term,'  ');
   Readln(Term);
   If Term= #26 then Term:= temp;
   Write('The number of Freqs. are  ( <= ',MaxFreqs,' ) :       ',Nfreq,'  ');
   Readln(Nfreq);

   for i:=1 to Nfreq do
    begin
      ClrScr; GotoXY(1,5);
      Write('Frequency (Hz) No.',i,'= ',Freq[i],'  ');  Read(Freq[i]);
    end;

   if NOT(Term in ['Y','y']) then
    begin
      ClrScr;
      for i:=1 to Nfreq do
       begin
         Writeln(Freq[i],'   Input Reflection Coefficient (Mag, Phase) = ');
         Write(Gamma[i,1].re,Gamma[i,1].im,'  ');
         Readln(Gamma[i,1].re,Gamma[i,1].im);
         tempr:=Gamma[i,1].re*cos(Gamma[i,1].im*RadDegree);
         Gamma[i,1].im:=Gamma[i,1].re*sin(Gamma[i,1].im*RadDegree);
         Gamma[i,1].re:=tempr;
         Writeln(Freq[i],'  Output Reflection Coefficient (Mag, Phase) = ');
         Write(Gamma[i,2].re,Gamma[i,2].im,'  ');
         Readln(Gamma[i,2].re,Gamma[i,2].im);
         tempr:=Gamma[i,2].re*cos(Gamma[i,2].im*RadDegree);
         Gamma[i,2].im:=Gamma[i,2].re*sin(Gamma[i,2].im*RadDegree);
```

```
    Gamma[i,2].re:=tempr;
  end; {for i}
end; {if Term}

CardNo:=0; NoDevice:=0; OptPara:=0;
Opt:=False;
CardCt:= 0;   NF:=1;
IA:= 0;   IE:=0;
DevCt:= 0;
repeat
  NCardsConst:= CardNo;
  CardNo:=CardNo+1;
  ClrScr;

     {              Added to construct during input          }

     {              Print CSMA and CSMB                      }

  GotoXY(28,19);
  Case IA of
    0:  begin
          Write('A and B are open');
          ClearA;  ClearB;
        end;
    1:  begin
          Write('A has been filled');
          ClearB;
        end;
    2:  Write('A and B are full ');
  end;  {Case IA}

  GotoXY(2,19);
  Write('Module A =');
  GotoXY(52,19);
  Writeln('Module B =');
  For i:= 1 to 2 do
  For j:= 1 to 2 do begin
    Write(CSMA[i,j].re:9,' + j',CSMA[i,j].im:9);
    Write('                    ');
    Writeln(CSMB[i,j].re:9,' + j',CSMB[i,j].im:9);
  end;


  GotoXY(15,3); Writeln('Final S-parameters are given for Module #1');
  GotoXY(15,5); Writeln('CardNo. = ',CardNo);
  Writeln('Enter type of Card: D = Device (Input Data)');
  Writeln('                    R = Resistor');
  Writeln('                    C = Capacitor');
  Writeln('                    L = Inductor');
  Writeln('                    X = Transmission Line');
  Writeln('                    A = Module combination');
  Writeln('                    M = Module assignment');
  Write('Enter Choice or Return if finished: ');
  With Card[CardNo] do
    TempS:= CardType;
  If CardNo> OldNoCards then TempS:=' ';
  If edit then Write(TempS,'   ');
  temp:= TempS;     Readln(TempS);
  If TempS= #26 then TempS:= temp;

  CType:=TempS[1]; ClrScr; GotoXY(1,5);
  Case CType of
       'D','d':  DeviceRead;
       'R','r':  ResistanceRead;
       'C','c':  CapacitanceRead;
       'L','l':  InductanceRead;
       'X','x':  LineRead;
       'A','a':  ModuleComb;
       'M','m':  ModAssign;
  end;  {Case}

    StartConst;
    Case ID of
      1:  ComponentCalc;
      2:  ModuleCombine;
```

```
          3:  ModuleAssign;
          4:  DeviceData;
        end;  {Case ID}

    until (CType=' ') or (CType< #65) ;

    InPrint;
    GotoXY(1,25);
    Write('Would you like to edit information? (Y/N) ');
    Read(KBD,Temp);   Writeln;
    If Temp in ['Y','y'] then
      Edit:= True
       else
        NoEdit:= 'Y';

    Write('Would you like to save entered data in a file (Y/N)? ');
    Read(KBD,Temp); Writeln;
    If Temp in ['Y','y'] then
      begin
        Files(Pathname);
        Writeln;
        Write('Enter name of file you wish to write to -- ');
        Readln(Filename);
        OutData;
      end;
    OldNoCards:= NCardsConst;
  end;    {While NOT NoEdit}

    if Opt then
     begin
       Write('Is Optimization to be done? (Y/N): ');
       Read(KBD,Temp);  Writeln;
       Opt:=Opt and (Temp in ['Y','y']);
     end;

    if Opt then
     begin
       Write('Is Objective Function frequency dependent? (Y/N): ');
       Read(KBD,Temp); Writeln;
       Dependent:=(Temp in ['Y','y']);
       Write('Is Status to be printed to screen during Optimization? (Y/N): ');
       Read(KBD,Temp);  Writeln;
       if (Temp in ['Y','y']) then Iprint:='Y' else Iprint:='N';
       With Card[CardNo] do
        begin
          CardType:='F';
          Write('Error Function Module No. = '); Readln(Value[1]);
        end;
         if dependent then
         for i:=1 to Nfreq do FunctionRead(Funct[i])
       else
        begin
          i:= 1;  FunctionRead(Funct[i]);
          if Nfreq>1 then
            for i:=2 to Nfreq do Funct[i]:=Funct[1];
        end;
     end;
end;  {INDATA}
```

The vita has been removed from the scanned document