COMPUTER AIDED DESIGN OF LUMPED AND DISTRIBUTED BROADBAND MATCHING AND FILTERING NETWORKS

A MASTER'S THESIS

in

Electrical and Electronic Engineering

Middle East Technical University

By

Adnan Köksal

July 1987

Approval of the Graduate School of Natural and Applied Sciences.

Prof.Dr. Bilgin KAFTANOGLU
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Electrical and Electronic Engineering.

Prof.Dr. Canan TOKER
Chairman of Department

We certify that we have read this thesis and that, in our opinion, it is fully adequate, in scope and quality, as a thesis for Master of Science in Electrical and Electronic Engineering.

Assoc.Prof.Dr. Nevzat YILDIRIM
Supervisor

Examining Committee in Charge:
Prof.Dr. Canan TOKER (Chairman)
Assoc.Prof.Dr. Zafer ÜNVER
Assoc.Prof.Dr. Nevzat YILDIRIM
Asst.Prof.Dr. Mustafa KUZUOGLU
Ergun BORA, M.sc in EE Eng.

A B S T R A C T

COMPUTER AIDED DESIGN OF LUMPED
AND DISTRIBUTED BROADBAND MATCHING
AND FILTERING NETWORKS

KÖKSAL , Adnan

M.S. in E.E.E.

Supervisor : Assoc.Prof.Dr. Nevzat YILDIRIM

July, 1987 ; 140 pages

In this study a computer program which can be used to design and optimize lumped and/or distributed equalizer networks is developed. The program aims to obtain a specified gain versus frequency response over a prescribed frequency band between complex generator and load networks.

The Simplified Real Frequency Technique is applied to both the lumped and distributed equalizer cases. Inclusion of the finite jw transmission zeros, if needed, is made possible.

An element optimization part is also added which can be used if impractical element values result from the Simplified Real Frequency part. This part also enables the trial of mixed type equalizer designs.

The same optimization technique, namely the Levenberg-Marquardt Finite Difference Optimization is applied to both the gain and element optimization parts. The element optimization part is realized by penalty method.

The computer program is run for several practical problems of single and double broadband matching. The results are quite satisfactory.

Key Words: lumped, distributed, equalizer, simplified real frequency technique, transmission zeros, levenberg-marquardt, single matching, double matching, broadband, optimization

Ö Z E T

KÖKSAL, Adnan

Yüksek Lisans Tezi: Elektrik ve Elektronik Müh. Böl.

Tez Yöneticisi : Doç.Dr. Nevzat YILDIRIM

Temmuz 1987 ; 140 sayfa

Bu çalışmada, kompleks kaynak ve yük devreleri arasında belirli bir frekans aralığında istenen kazanç-frekans eğrisini veren empedans uyumlama devrelerinin tasarımında kullanılmak üzere bir bilgisayar yazılım paketi geliştirilmiştir.

Birleşik ve dağınık elemanlı tasarımlar için Basitleştirilmiş Reel Frekans Tekniği kullanılmıştır. İhtiyaç halinde sınırlı jw-ekseni iletim sıfırları eklenmesi de mümkün hale getirilmiştir. Basitleştirilmiş Reel Frekans Tekniği ile kazanç optimize eden kısımdan pratik olmayan eleman değerleri elde edilmesi durumunda kullanılmak üzere bir eleman optimizasyonu kısmı da eklenmiştir. Böylelikle karışık tip uyumlama devreleri de denenebilir hale gelmiştir. Kazanç ve eleman

optimizasyonu yapan her iki kısımda da aynı algoritma - Finite Difference Levenberg-Marquardt Optimization Technique - kullanılmıştır. Eleman optimizasyonunun gerçeklestirilmesinde ceza metodu kullanılmıştır.

Bilgisayar yazılımı birçok tekli ve ikili uyumlama problemlerinde denenmis ve tatmin edici sonuçlar alınmıştır.

Anahtar kelimeler: toplu, dağınık, uyumlama devresi, basitleştirilmiş reel frekans tekniği, iletim sıfırları, levenberg-marquardt, tekli uyumlama, ikili uyumlama, geniş band.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Assoc.Prof.Dr.Nevzat Yıldırım and Assoc.Prof.Dr.Abdullah Atalar for their valuable guidance and supervision.

I acknowledge the help and support of all my colleagues especially Mr.Ünsal Kulein and Mr.Fuat Öztap.

I owe a thank to Mr.Sencer Koç for his understanding and help in providing the computer facilities.

Last, but not the least, my special thanks are to my fiancée, Gülser, for her support and careful proof-reading.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
ACKNOWLEDGEMENTS	vii
LIST OF TABLES	x
LIST OF FIGURES	xi
LIST OF ABBREVIATIONS	xiii
CHAPTER	
I. INTRODUCTION	1
II. SIMPLIFIED REAL FREQUENCY TECHNIQUE	6
2.1. Introduction	6
2.2. The Simplified Real Frequency Technique for Lumped Element Networks	7
2.3. The Method for the Lumped Design	11
2.4. The Simplified Real Frequency Technique for the Commensurate Distributed Design	17
2.5. The Method for the Distributed Element Equalizer Design	25
III. TRANSFER FUNCTION IN TERMS OF ELEMENT VALUES	28
3.1. Introduction	28
3.2. Formation of the Indefinite Admittance Matrix	29
3.3. Reduction of the Indefinite Admittance Matrix	31
3.4. Gain in Terms of y-parameters	32
3.5. The Method for Element Optimization	34

Page

IV. COMPUTER PROGRAM AND APPLICATIONS..... 36

4.1. Introduction..... 36

4.2. The Finite Difference Levenberg-Marquardt Nonlinear Least Squares Approximation..... 38

4.3. Fibonacci Search..... 40

4.4. Computer Program..... 43

4.5. Applications..... 46

V. CONCLUSIONS..... 61

LIST OF REFERENCES..... 63

APPENDIX A. ANALYTIC THEORY OF BROADBAND MATCHING..... 66

A1. Introduction..... 66

A2. Positive Real and Bounded Real Properties..... 67

A3. Relation Between Unit Normalized and Youla-Complex Normalized Scattering Parameters..... 69

A4. Youla Constraints..... 72

A5. Reduction of the Double Matching Problem to Youla Constaints..... 75

A6. Analytic Procedure for Broadband Matching..... 80

A7. Evaluation of Analytic Theory... 83

APPENDIX B. BLOCK DIAGRAMS OF PROGRAM..... 84

APPENDIX C. LISTING OF THE PROGRAM..... 87

LIST OF TABLES

Table	Page
4.1. Gain Values of Fifth Order Chebychev Filter	55

LIST OF FIGURES

Figure	page
1.1. Single Matching Problem	2
1.2. Double Matching Problem	2
2.1. Doubly Terminated 2-port	8
2.2.a. Unit Element	18
2.2.b. Open Circuited Stub	18
2.2.c. Short Circuited Stub	18
2.3. Finite-jw Transmission Zero Element	22
3.1. n Terminal Passive Network	29
3.2. Two-port Network with y-parameters	32
3.3. y-parameter Model of Two-ports	33
3.4. Double Matching	33
4.1. The Block Diagram of the Program	44
4.2. The Circuit of Example 1	47
4.3. The Return Loss for Example 1	48
4.4. The Circuit of Example 2.a	49
4.5. The Return Loss of Example 2.a	51
4.6. The Circuit of Example 2.b	52
4.7. The Return Loss of Example 2.b	53
4.8. The Circuit of Example 3	56
4.9. The Return Loss of Example 3	57
4.10. The Circuit of Example 4	59
4.11. The Return Loss of Example 4	60
A.1. The Double Matching Problem	66
A.2. Two-port with Complex Load	69

Figure	page
A. 3. Darlington Representation.....	76
A. 4. Circuit for Generator Extraction.....	76
A. 5. Circuit for Load Extraction.....	77
A. 6. Reduced Matching Problem.....	79
B. 1. Block Diagram for Lumped Design.....	84
B. 2. Block Diagram for Distributed Design.....	85
B. 3. Block Diagram for Element Optimization.....	86

LIST OF ABBREVIATIONS

SRFT : Simplified Real Frequency Technique

RFT : Real Frequency Technique

BBM : Broadband Matching

PR : Positive Real

BR : Bounded Real

UE : Unit Element

CHAPTER 1

INTRODUCTION

Broadband matching problem (BBM) is defined as the achievement of a desired performance between arbitrary load and generator networks over a given frequency band. Gain, noise figure and voltage standing wave ratio are typical performance measures that are mostly encountered.

In the design of communication networks matching is extremely important, especially in the microwave frequencies where the cost of each dB of gain rises rapidly.

The problem of achieving a preassigned transfer function from a real impedance generator to a frequency dependent load is known as single matching (Fig.1.1), whereas that of from a frequency dependent generator to a frequency dependent load is known as double matching (Fig.1.2).

There are two trends in solving the broadband matching problem: The analytic theory and the computerized methods.

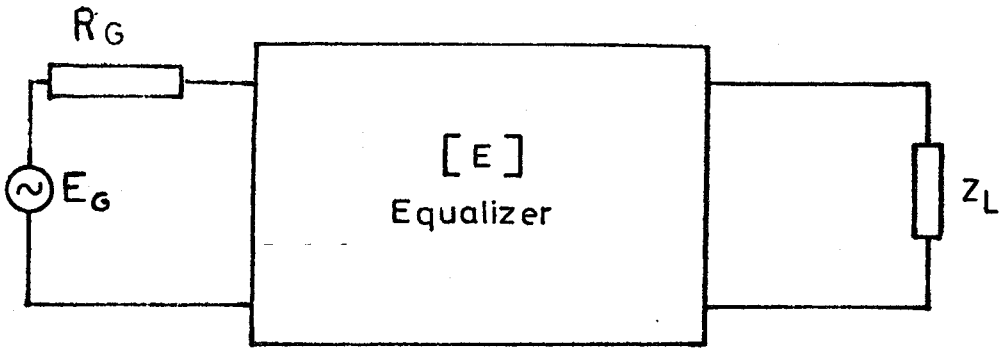

Figure 1.1. Single Matching Problem

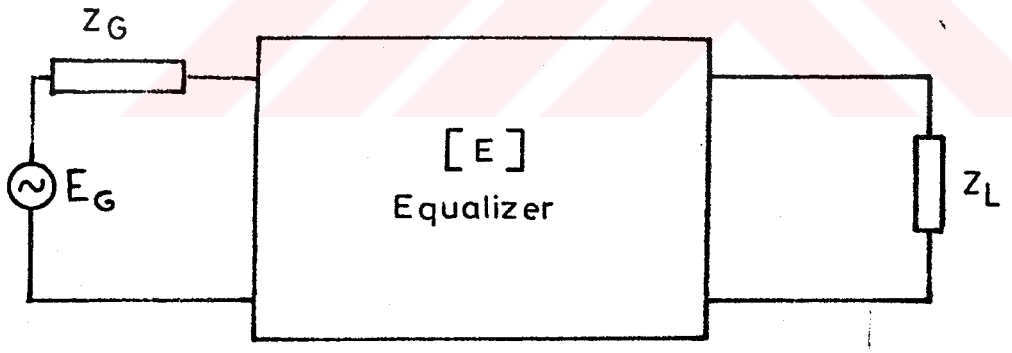

Figure 1.2. Double Matching Problem

The analytic theory of broadband matching [1, 2, 9, 16] deals with the problem of finding a realizable equalizer network giving the optimum response. In order that the analytic theory can be applied the analytic forms of the load and generator networks must be known or approximated. When the degrees of the polynomials involved are small the analytic theory is not very difficult to apply, but the difficulty in choosing the optimum transfer function for the complex generator and load networks still remains.

A review of the analytic theory of BBM developed by Youla [9, 16], Carlin and Yarman [1, 2] is given in Appendix A.

The other approach to the problem is computer oriented. The most usual trend in computer oriented methods is the optimization of the values of the elements in a chosen circuit topology. The optimization problem emerged by this approach is highly nonlinear and requires sophisticated algorithms and good choice of circuit topology and initial element values.

The Real Frequency Technique (RFT) introduced by Carlin [8, 14] is an alternative to element optimization in the single matching problems. In this technique minimum phase equalizers are assumed and the real part of the equalizer impedance Z_0 (Fig. 1. 1) is approximated by line

segments. Using the Hilbert transform relations between the real and the imaginary parts of a minimum phase function, the equalizer impedance can be constructed in terms of the same line segments. Hence the optimum line segment approximation gives a realizable equalizer. The RFT is later extended by Yarman [1] to solve the double matching problems.

The results of the problems solved by the RFT show that the method is efficient and easily applicable even in the cases the analytic theory is too difficult to be applied.

Another method, which is called Simplified Real Frequency Technique (SRFT), simplifying the optimization procedure is introduced by Yarman [4, 11, 15] and applied to the design of double matching networks and to the design of matching networks of microwave amplifiers. The coefficients of the input reflection factor to the resistively terminated equalizer are directly optimized in this method. The choice of transmission zero characteristics of the equalizer, together with the initialized coefficients mentioned, gives rise to the generation of the scattering matrix of the equalizer which is used to construct the objective function.

In this work Simplified Real Frequency Technique is investigated and extended to handle the design of

commensurate distributed equalizer networks. A computer program is developed to solve the double matching problem with this technique.

Since the SRFT is an unconstrained method, impractical element values can be obtained. Therefore an element optimization part is added to the program. This part also enables the trial of the mixed type equalizer networks.

The work can be outlined as follows: in chapter II the SRFT is explained and the methods for lumped and commensurate distributed equalizer networks are given. The approach used in the element optimization part is given in chapter III. Chapter IV consists of the explanation of the program and the examples. The conclusions are presented in chapter V.

CHAPTER 2

SIMPLIFIED REAL FREQUENCY TECHNIQUE

2.1. INTRODUCTION

The Simplified Real Frequency Technique (SRFT) is a computer aided design procedure for the solution of the double matching problem. It is introduced by B.S. Yarman [4] and is simpler in nature than the Real Frequency Technique (RFT) introduced by H.J. Carlin [9, 14, 2, 1]. The simplicity of the method is that the coefficients of the input reflection factor to the equalizer terminated by a reference resistance at the load side is optimized directly. It is not necessary to find the analytic form of the equalizer impedance in the optimization process as in the RFT.

The algorithm takes the initialized coefficients of the input reflection factor and real frequency measurement data of the generator and load impedances as input and yields the equalizer input impedance with the equalizer terminated in a reference resistor at the load side as

the output. Then by the use of the element extraction techniques the equalizer element values can be determined.

In this chapter the method is described in detail first for the lumped element equalizer design. Then the necessary theorems relating to the distributed networks are given for the development of the commensurate transmission line equalizer design algorithm using SRFT. Finally the algorithm for the distributed design case is described for the two widely applied class of networks, namely the low-pass and the band-pass types.

2.2. THE SIMPLIFIED REAL FREQUENCY TECHNIQUE FOR THE LUMPED ELEMENT NETWORKS

For the lumped element equalizer case the problem is to find a realizable two-port network [E], such that the gain between its terminations is optimized over a given frequency band (Fig. 2.1).

The core of the method lies in the selection of the real normalized scattering parameters of the equalizer [E]. If the real normalized scattering matrix of [E] is given as:

$$[S] = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \qquad 2.1$$

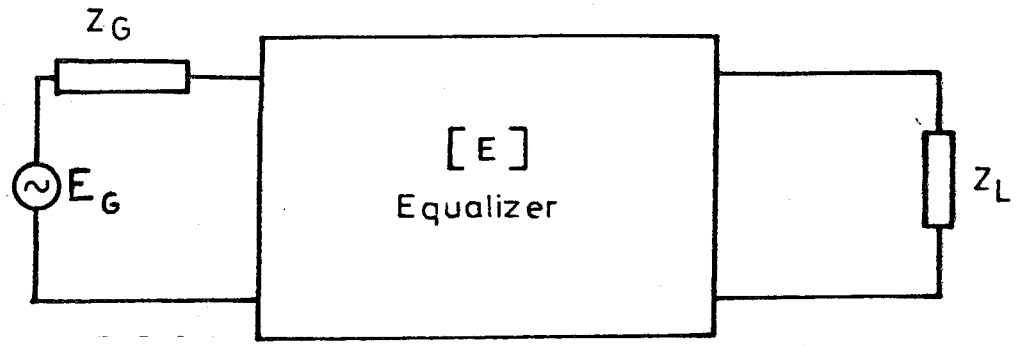

Figure 2.1. Doubly Terminated 2-port

the transducer power gain, $T(w^2)$ of the network in Fig. 2.1 can be written as [17]:

$$T(w^2) = \frac{(1 - |s_G|^2)\ (1 - |s_L|^2)\ |s_{21}|^2}{|1 - s_{11}\ s_G|^2\ |1 - s_{out}\ s_L|^2} \qquad 2.2$$

where,

$$s_G(s) = \frac{Z_G - R_O}{Z_G + R_O} \qquad 2.3.a$$

$$s_L(s) = \frac{Z_L - R_O}{Z_L + R_O} \qquad 2.3.a$$

where R_O is the normalization resistance.

If reciprocal equalizer is assumed then $s_{21}(s) = s_{12}(s)$

and $s_{21}(s)$ can be written as :

$$s_{21}(s) = s_{12}(s) = \pm \frac{f(s)}{g(s)} \qquad 2.4$$

where g(s) is a strictly Hurwitz polynomial [12] and f(s) is a polynomial containing the transmission zeros of the equalizer formed by the factors of the form:

s^k,

$(s^2 + w_p^2)^m$,

$(s^2 - s_1^2)^t$

where k, m, t being positive integers.

In order that $s_{21}(s)$ to be the scattering parameter of a realizable network it must be bounded real, hence the degree of f(s) must be less than or equal to the degree of g(s).

The zeros of f(s) are the transmission zeros of the equalizer which are located at $s = 0$, $s = \pm jw$ or at $s = \sigma \pm jw$. The transmission zeros at $s = \infty$ are created if the degree of g(s) is greater than the degree of f(s). The number of the transmission zeros at infinity is equal to the difference of the degrees of g(s) and f(s).

Using the unitary property [12] of the scattering matrix of the equalizer and employing the analytic continuation

the following group of equations can be written.

$$S_{11}(s)\ S_{11}(-s) + S_{21}(s)\ S_{21}(-s) = 1 \qquad 2.5.a$$

$$S_{11}(s)\ S_{21}(-s) + S_{21}(s)\ S_{22}(-s) = 0 \qquad 2.5.b$$

$$S_{21}(s)\ S_{11}(-s) + S_{21}(-s)\ S_{22}(s) = 0 \qquad 2.5.c$$

$$S_{22}(s)\ S_{22}(-s) + S_{21}(s)\ S_{21}(-s) = 1 \qquad 2.5.d$$

The substitution of 2.4 into 2.5.a gives:

$$S_{11}(s)\ S_{11}(-s) = \frac{g(s)\ g(-s) - f(s)\ f(-s)}{g(s)\ g(-s)} \qquad 2.6.a$$

Spectrally factorizing the numerator polynomial the following can be written.

$$\frac{g(s)\ g(-s) - f(s)\ f(-s)}{g(s)\ g(-s)} = \frac{h(s)\ h(-s)}{g(s)\ g(-s)} \qquad 2.6.b$$

The factorization used here is arbitrary, that is, the roots of h(s) and h(-s) may be chosen as required. However, usually the left half plane zeros of $S_{11}(s)\ S_{11}(-s)$ are used to construct h(s) in order to obtain minimum phase structures.

Now if the numerators are equated the well-known Feldtkeller equation is obtained.

$$g(s)\ g(-s) = f(s)\ f(-s) + h(s)\ h(-s) \qquad 2.6.c$$

The degree of g(s), as it is shown from 2.6.c is equal

to either the degree of f(s) or that of h(s), whichever is greater.

From 2.6.a $s_{11}(s)$ can be written as:

$$s_{11}(s) = \frac{h(s)}{g(s)} \qquad 2.7$$

and substituting 2.4 and 2.7 in 2.5.b or 2.5.c gives :

$$s_{22}(s) = - \frac{f(s)}{f(-s)} \cdot \frac{h(-s)}{g(s)} \qquad 2.8$$

From the discussion up to here it can be concluded that if the numerator polynomials h(s) and f(s) are known then the scattering parameters of the equalizer network can be constructed.

Here it is worthy to note that one can start by substituting 2.4 into 2.5.d and obtain $s_{22}(s)$ and then use these to construct $s_{11}(s)$. The choice is arbitrary if computer aided design is considered.

The next step is to find these parameters' unknowns satisfying the desired gain requirements.

2.3. THE METHOD FOR THE LUMPED DESIGN [4]

The results of the previous section may be combined with an optimization routine for achieving the gain

requirements. For this purpose the coefficients of the numerator polynomial of $s_{11}(s)$, i.e. h(s) are chosen as initials. Since for many practical problems equalizer networks having transmission zeros at s=0, s=∞ and at $s=\pm jw_{pi}$ are sufficient, f(s) is chosen to contain factors of s^K and $(s^2 + w_{pi}^2)^m$ only. With this choice of f(s), using 2.4 and 2.8, $s_{21}(s)$ and $s_{22}(s)$ becomes :

$$s_{21}(s) = \frac{s^K}{g(s)} \prod_{i=1}^{l} (s^2 + w_{pi}^2)^{mi} \qquad 2.9.a$$

$$s_{22}(s) = -(-1)^K \frac{h(-s)}{g(s)} \qquad 2.9.b$$

and from 2.6 the unknown polynomial g(s) is related to the specified f(s) and trial h(s) as:

$$g(s)\, g(-s) = h(s)\, h(-s) + (-1)^K\, s^{2k} \prod_{i=1}^{l} (s^2 + w_{pi}^2)^{2mi} \qquad 2.10$$

where l finite jw transmission zeros with respective multiplicities m_i are assumed.

The circuit type and the topology depends on the choice of the type and the number of the transmission zeros. The degrees of the functions are also determined by these choices. Considering 2.10 and 2.9.a the following classes can be outlined:

I) K = 0 and $m_i = 0$, i = 1, 2, .., l

This corresponds to the low-pass type equalizer since no transmission zeros at $s=0$ and at $s=\pm jw_1$ are assumed. The only transmission zeros are at $s=\infty$ due to the degree of $g(s)$. The degree relations are:

$\deg(f) = 0.$

$\deg(g) = \#$ of transmission zeros at $s=\infty$.

$\deg(h) = \deg(g)$.

Transformerless design can be made in this case by setting the coefficient of the s^0 in $h(s)$ to zero.

II) $K = 0$, and m_i not zero for all i.

This case leads to the low-pass ladder elements together with the finite jw transmission zero creating elements which are either serially inserted parallel LC combination, or parallel connected series LC combination. The degree relations can be written as follows in this case:

$$\deg(f) = \sum_{i=1}^{l} 2m_i$$

$\deg(g) = \deg(f) + \#$ of transmission zeros at ∞

$\deg(h) = \deg(g)$

Transformerless design is also possible for this case.

III) $k \rightarrow 0$ and $m_i = 0$ for $i = 1, 2, .., l$

This choice results in an equalizer network having both high-pass and low-pass elements in general. However, if deg(f) = deg(g), this results in a high-pass equalizer having no low-pass elements. Number of low-pass elements is equal to the difference:

$$\deg(g) - \deg(f)$$

The degree relations are:

$$\deg(f) = k$$

$$\deg(g) \geq \deg(f)$$

$$\deg(h) = \deg(g) \quad \text{if} \quad \deg(g) > \deg(f)$$

$$\deg(h) < \deg(g) \quad \text{if} \quad \deg(g) = \deg(f)$$

Choosing the inequality in the last relation means perfect match at $s=\infty$.
The transformerless design is not possible in this case since transmission zeros at s=0 exist.

IV) $k > 0$ and m_i not zero for all i.

This corresponds to the most general type. All types of transmission zeros which are possible in the method can be realized. However if deg(f) = deg(g) no low-pass elements occur as in III. The degree relations are as follows:

$$\deg(f) = k + \sum_{i=1}^{\ell} 2m_i$$

$\deg(g) \geq \deg(f)$

$\deg(h) = \deg(g)$ if $\deg(g) > \deg(f)$

$\deg(h) < \deg(g)$ if $\deg(g) = \deg(f)$ choosing the inequality meaning perfect match at $s=\infty$.

The choice of trial h(s) is not critical, since the transfer function created in terms of its coefficients behaves smoothly. However, initilization with alternating sign and comparable magnitude proved to be faster in convergence.

Assuming that the degree of the initial polynomial is determined according to the outlined rules and h(s) is initiated as,

$$h(s) = h_0 + h_1 s + .. h_j s^j .. + .. + h_n s^n$$

The following computational steps may be written for the optimization procedure [4].

1. The even polynomial g(s) g(-s) is constructed using 2.10, assuming k transmission zeros at s=0 and ℓ finite jw transmission zeros.

2. The roots of g(s) g(-s) are found.

3. The left half plane roots of g(s) g(-s) are chosen

and the strictly Hurwitz polynomial $g(s)$ is constructed.

4. The scattering parameters of the equalizer are evaluated at each optimization frequency w_i according to the following:

$$s_{11}(jw_i) = \frac{h(jw_i)}{g(jw_i)}$$

$$s_{21}(jw_i) = \pm \frac{(jw_i)^k \ (-w_i^2 + w_{p1}^2)^{m1} \ldots \ (-w_i^2 + w_{pl}^2)^{ml}}{g(jw_i)}$$

$$\text{and } s_{22}(jw_i) = -(-1)^k \frac{h(-jw_i)}{g(jw_i)}$$

5. The values found in 4 are substituted in transducer power gain equation and $T(w_i^2)$ are obtained at every optimization frequency.

$$T(w_i^2) = \frac{(1 - |s_G|^2)\ (1 - |s_L|^2)\ |s_{21}(jw_i)|^2}{|1 - s_{11}(jw_i)\ s_G(jw_i)|^2\ |1 - s_{out}(jw_i)\ s_L(jw_i)|^2}$$

$$\text{where } s_G(jw_i) = \frac{Z_G(jw_i) - R_O}{Z_G(jw_i) + R_O}$$

$$\text{and } s_L(jw_i) = \frac{Z_L(jw_i) - R_O}{Z_L(jw_i) + R_O}$$

6. A direction for all h_j's is obtained such that the errors between the current gain values and the required gain values are minimized at all optimization frequencies.

7. New values of h_j's are computed and the procedure is repeated beginning from step 1 until the gain errors at each optimization frequency fall in a tolerable range.

After the procedure is completed, the input impedance to the equalizer with the equalizer terminated in the reference resistance at the load side is computed using,

$$Z_{in}(s) = R_0 \frac{1 + s_{11}(s)}{1 - s_{11}(s)} \qquad 2.11$$

Finally the equalizer element values are extracted from the input impedance obtained leaving the resistance R_0 at the load side.

2.4. THE SIMPLIFIED REAL FREQUENCY TECHNIQUE FOR THE COMMENSURATE DISTRIBUTED DESIGN

In this section the SRFT will be used to obtain equalizer networks containing unit elements (Fig. 2. 2. a), open-circuited stubs (Fig. 2. 2. b), and short-circuited stubs (Fig. 2. 2. c).

The analysis is restricted to the mostly encountered two types of networks : low-pass and band-pass types.

Low-pass commensurate transmission line networks consist of series UE's and shunt open-circuited stubs

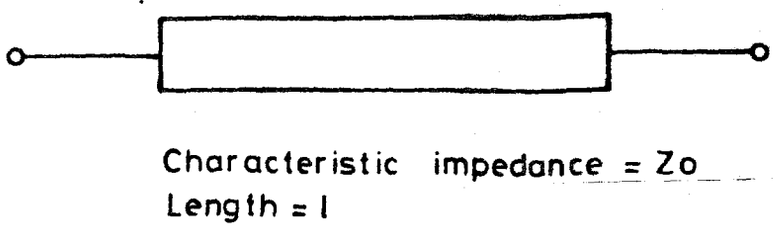

Figure 2.2.a. Unit Element

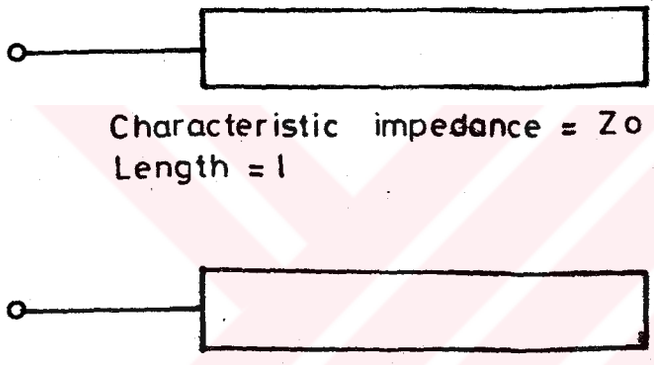

Figure 2.2.b. Open Circuited Stub

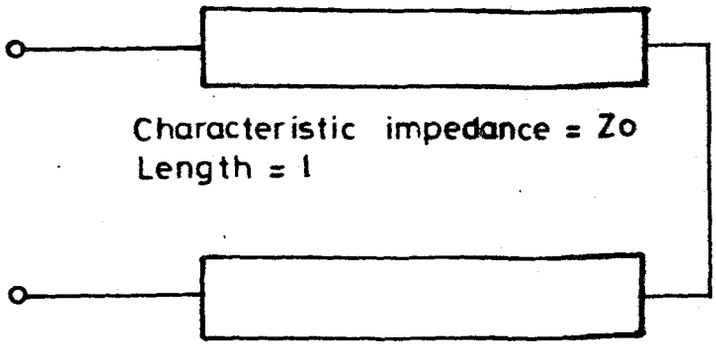

Figure 2.2.c. Short Circuited Stub

whereas band-pass commensurate transmission line networks consist of series UE's, shunt short-circuited and series open-circuited stubs.

It is well known that under the transformation

$$\lambda = \Sigma + j\Omega = \tanh\beta\ell = \tanh w\Gamma \qquad 2.14$$

where $\beta=2\pi/\lambda_G$ λ_G being the guided wavelength, ℓ the commensurate length, and Γ the one way time delay on the commensurate length; the commensurate circuit synthesis is simplified. The positive realness of impedances and bounded realness of scattering matrices hold for these circuits under the specified transformation [12]. Furthermore the distributed filter design is simplified since it is possible to use the filter tables in the λ domain and then to replace the λ domain capacitors and inductors by open-circuited and short-circuited stubs respectively. However all these stubs are connected at the same point without any physical separation. Therefore redundant UE's must be incorporated to separate the stubs.

On the other hand if a special form for $s_{21}(\lambda)$ is chosen then the realization of bounded real input reflection factor only with the UE's are guaranteed.

The following theorem given by Carlin states the conditions on $s_{21}(\lambda)$ of commensurate distributed

networks in order that the network is realizable with only UE's.

Theorem 2.1 [3]

The necessary and sufficient conditions that a two-port formed of a cascade of UE's possess a prescribed $|s_{12}(j\Omega)|^2$ $(\lambda=\Sigma+j\Omega)$ is that,

$$0 \leq |s_{21}(j\Omega)|^2 \leq 1 \qquad 2.15.a$$

$$|s_{21}(j\Omega)|^2 = \frac{(1+\Omega^2)^n}{P_n(\Omega^2)} \qquad 2.15.b$$

$$s_{21}(\lambda)\, s_{21}(-\lambda) = \frac{(1-\lambda^2)^n}{P_n(-\lambda^2)} \qquad 2.15.c$$

where P_n is an even polynomial of degree 2n. The significance of this theorem is that any cascade of UE's must have $s_{21}(\lambda)$ satisfying the theorem, and conversely any function satisfying the theorem can be realized by a cascade of UE's.

Now if $s_{21}(\lambda)$ is chosen as:

$$s_{21}(\lambda) = \frac{(1-\lambda^2)^{n/2}}{D_n(\lambda)} \qquad 2.16$$

where $D(\lambda)$ is a strictly Hurwitz polynomial of degree n which is obtained from the spectral factorization of $P_n(-\lambda^2)$, a network containing n UE's can be designed.

A transmission zero on $\lambda=\omega$ cannot be obtained with this form of $s_{21}(\lambda)$. In order to have a transmission zero at $\lambda=\omega$ the denominator of $s_{21}(\lambda)$ must be higher in degree than the numerator. This suggests that the transmission zeros at $\lambda=\omega$ can be obtained by increasing the degree of $D_n(\lambda)$, which effectively means the introduction of open-circuited stubs.

If the effect of the stubs is also included $s_{21}(\lambda)$ becomes:

$$s_{21}(\lambda) = \frac{(1 - \lambda^2)^{n/2}}{D_{n+q}(\lambda)} \qquad 2.17$$

Where n UE's and q open-circuited stubs are assumed.

Together with the added finite $j\Omega$ transmission zeros $s_{21}(\lambda)$ becomes:

$$s_{21}(\lambda) = \frac{(1 - \lambda^2)^{n/2}}{D(\lambda)} \prod_{i=1}^{\ell} (\lambda^2 + \Omega_{pi}^2)^{mi} \qquad 2.18$$

$$= \frac{F(\lambda)}{D(\lambda)}$$

where $D(\lambda)$ is a Hurwitz polynomial and ℓ finite $j\Omega$ transmission zeros are assumed, namely $\lambda_1, \ldots, \lambda_\ell$; having multiplicities $m_1, \ldots, m_\ell$ respectively.

With this choice of $s_{21}(\lambda)$ both low-pass and band-pass network design consideration is made possible, with the

proper frequency transformation.

With $\Omega = \tan(w\Gamma)$ transformation the previous scattering parameter in the λ domain gives the transmission zeros at $\lambda=\infty$ only. These corresponds to the frequencies where the open circuited stubs have a quarter-wave length. However, if the transformation is chosen as $\Omega=-\cot(w\Gamma)$ it is seen that this has the effect of replacing λ by $1/\lambda$ in all of the previous equations. And with this choice the transmission zeros at $\lambda=0$ can be obtained. By these choices of frequency variable both the low-pass and the band-pass networks are handled.

The finite $j\Omega$ transmission zero creating elements are two UE's connected in series in between, but shunt connected to the equalizer network (Fig. 2.3).

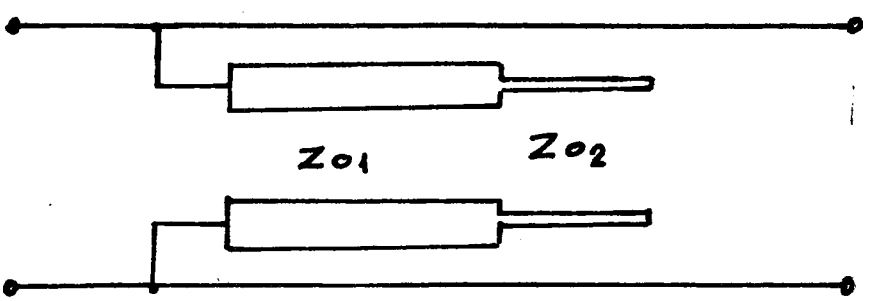

Figure 2.3. Finite-jw Transmission Zero Element

Now consider an equalizer network consisting of commensurate distributed elements of the specified types. Using the unitary property of its real normalized scattering matrix in the λ domain,

$$s_{11}(\lambda)\, s_{11}(-\lambda) = 1 - s_{21}(\lambda)\, s_{21}(-\lambda)$$

$$= \frac{D(\lambda)\, D(-\lambda) - F(\lambda)\, F(-\lambda)}{D(\lambda)\, D(-\lambda)} \qquad 2.19$$

and by making analogy to the lumped case it can be written,

$$s11(\lambda) = \frac{H(\lambda)}{D(\lambda)} \qquad 2.20.a$$

Putting 2.20.a into 2.19 and equating the numerators give the Feldtkeller equation for distributed case.

$$D(\lambda)\, D(-\lambda) = H(\lambda)\, H(-\lambda) + F(\lambda)\, F(-\lambda) \qquad 2.20.b$$

Using the unitary properties $s_{22}(s)$ can be written as:

$$s_{22}(\lambda) = - \frac{F(\lambda)}{F(-\lambda)} \frac{H(-\lambda)}{D(\lambda)}$$

$$= - \frac{H(-\lambda)}{D(\lambda)} \qquad 2.20.c$$

2.20.c results since $F(\lambda)$ is even.

If the number of UE's is n and the number of stubs is q

the degree relations of the functions involved can be stated as follows:

$$\deg(F) = n + \sum_{i=1}^{l} 2m_i$$

$$\deg(D) = \deg(F) + q$$

$$\deg(H) = \deg(D).$$

Some important design concepts should be emphasized related to the functions involved before continuing with the computational steps.

I) For the low-pass case the quarter-wave frequency must be outside of the matching region.

II) The coefficient of the λ^0, i.e H_0 for low-pass case must be zero since all lines disappear at D.C.

III) For the band-pass case, the quarter-wave frequency must be taken in the passband region.

IV) If the first extracted element is a shunt short-circuited stub; the coefficient of the highest power of λ in $H(\lambda)$ for the band pass case must be equal to the negative of the corresponding coefficient in $D(\lambda)$ since at D.C. the short-circuited stubs causes full reflection and $s_{11}=-1$ for these types of networks.

Therefore, as in the lumped case realizable scattering

parameters of the equalizer are constructed. Next the coefficients of $H(\lambda)$ are chosen as initiated values and the method for the design is given.

2.5. THE METHOD FOR THE DISTRIBUTED ELEMENT EQUALIZER DESIGN

Assuming that $H(\lambda)$ is given as,

$$H(\lambda) = H_0 + H_1 \lambda + .. H_j \lambda^j .. + .. + H_p \lambda^p$$

The following computational steps may be given as in the lumped case.

1. The even polynomial

$$D(\lambda)\, D(-\lambda) = H(\lambda)\, H(-\lambda) + F(\lambda)\, F(-\lambda)$$

$$= H(\lambda)\, H(-\lambda) + (1 - \lambda^2)^n \prod_{i=1}^{t} (\lambda^2 + \Omega_i^2)^{2mi}$$

is generated assuming n UE's and t transmission zeros.

2. The roots of $D(\lambda)\, D(-\lambda)$ are found.

3. The LHP roots of $D(\lambda)\, D(-\lambda)$ are chosen and the strictly Hurwitz polynomial $D(\lambda)$ are constructed.

4. The scattering parameters of the equalizer are evaluated at each transformed frequency $\lambda_K = j\Omega_K$.

$$s_{11}(j\Omega_K) = \frac{H(j\Omega_K)}{D(j\Omega_K)}$$

$$s_{22}(j\Omega_K) = -\frac{H(-j\Omega_K)}{D(j\Omega_K)}$$

$$s_{21}(j\Omega_K) = \frac{(1 + \Omega^2)^{n/2}}{D(j\Omega_K)} \prod_{i=1}^{l} (\Omega_i^2 - \Omega_K^2)^{mi}$$

5. The values found in 4 are substituted in transducer power gain equation and $T(w_K^2)$ are obtained at each optimization frequency using,

$$T(w_K^2) = \frac{(1 - |s_G|^2)\ (1 - |s_L|^2)\ |s_{21}(jw_K)|^2}{|1 - s_{11}(jw_K)\ s_G(jw_K)|^2\ |1 - s_{out}(jw_K)\ s_L(jw_K)|^2}$$

where $s_G(jw_K) = \dfrac{Z_G(jw_K) - R_O}{Z_G(jw_K) + R_O}$

and $s_L(jw_K) = \dfrac{Z_L(jw_K) - R_O}{Z_L(jw_K) + R_O}$

R_O being the normalization resistance.

6. A direction for all H_j's is obtained such that the errors between the required and the obtained gain values at all optimization frequencies are minimized.

7. New values of H_j's are obtained and the procedure is repeated beginning from step 1 until the errors fall in a tolerable range.

After the procedure is completed, the input impedance of the resistively terminated equalizer is obtained. The input impedance to the equalizer with the equalizer terminated in the reference resistance at the load side is,

for low-pass case :

$$Z_{in}(\lambda) = R_0 \frac{1 + s_{11}(\lambda)}{1 - s_{11}(\lambda)} \qquad 2.24$$

and for the band-pass case:

$$Z_{in}(\lambda) = R_0 \frac{1 + s_{11}(1/\lambda)}{1 - s_{11}(1/\lambda)} \qquad 2.25$$

because of the difference in the frequency transformations between these two type of designs as mentioned earlier.

The characteristic impedances of the unit elements and the stubs are obtained by Richard's extraction technique and the short-circuited and open-circuited stub extraction procedures respectively. Since the realization is not unique when the number of unit elements is greater than the number of stubs, the circuit that gives the most practical element values should be selected.

CHAPTER 3

TRANSFER FUNCTION IN TERMS OF ELEMENT VALUES

3.1 INTRODUCTION

As explained in the previous chapter the SRFT is used to directly optimize the coefficients of the numerator polynomial of the input reflection factor of the equalizer, and hence is an unconstrainted method. The element values are determined freely by the method. Whereas a computational efficiency is brought about by this property, impractical element values can be found in some problems. Another class of problems may require the trial of the mixed type equalizer networks, for example a lumped element may be desired in a distributed equalizer in order to supress the harmonic response of the network. In either of these cases an element optimization routine is needed. In order to satisfy this requirement a part of the written computer program is intended to deal with element optimization. The following sections describe the method used in that part.

3.2. FORMATION OF THE INDEFINITE ADMITTANCE MATRIX

The indefinite admittance matrix relates the total current at any node in the network to the voltages at the nodes, voltages being referenced to some node external to the network. Considering the network [N] in Fig. 3.1,

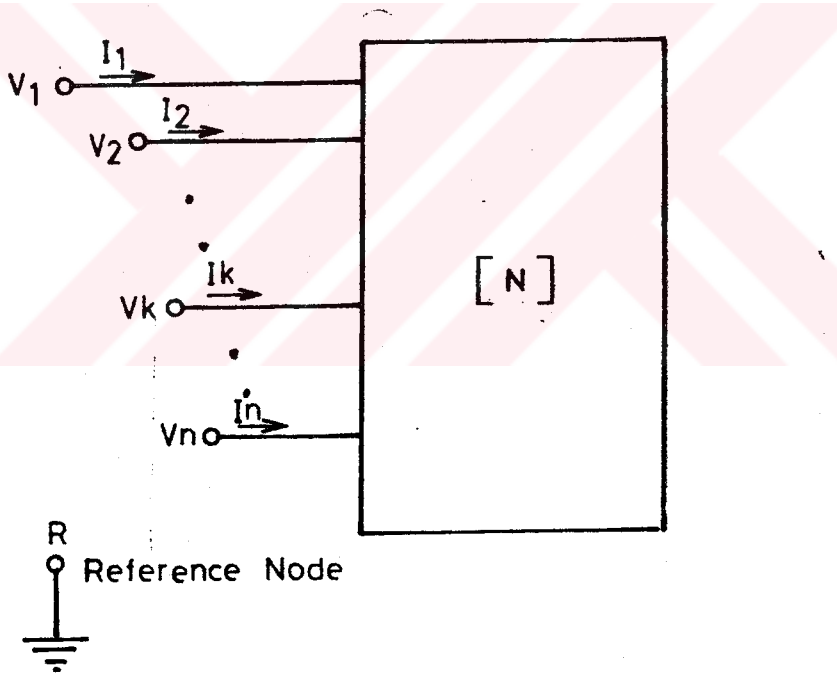

Figure 3.1. n Terminal Passive Network

its node currents can be related to the voltages

measured with respect to the reference node R as:

$$I(s)=Y(s)\ V(s) \qquad 3.1$$

where Y(s) is the indefinite admittance matrix and I(s) and V(s) are n dimensional vectors. The elements of the indefinite admittance matrix are given as,

$$Y_{ij} = \left.\frac{I_j}{V_j}\right|_{V_K=0,\ K\neq j} \qquad 3.2$$

Inspection of the matrix reveals its important properties stated below.

I) Each column sum of Y(s) is equal to zero.

II) Each row sum of Y(s) is equal to zero.

III) Y_{ii} is the sum of all admittances connected to node i.

IV) Y_{ij} is equal to minus the sum of all admittances.

V) $Y_{ij} = Y_{ji}$.

With the use of these properties the indefinite admittance matrix of passive elements can be written by inspection. The generation of the matrix in a computer program is also simple. The building blocks of lumped and distributed networks are capacitance, inductance, unit element, short circuited stub, and open circuited stubs.

3.3 REDUCTION OF THE INDEFINITE ADMITTANCE MATRIX [6]

If the n terminal network of Fig.3.1 is a two port network, then the current sum at the internal nodes of the network is zero and the indefinite admittance matrix can be reduced to the admittance matrix of the two port network.

Assuming that the jth node is an internal one,

$$I_j = y_{j1}V_1 + .. + y_{jj}V_j + .. + y_{jn}V_n = 0 \qquad 3.3$$

and V_j can be written as:

$$V_j = -\frac{y_{j1}}{y_{jj}} V_1 - ... - \frac{y_{jn}}{y_{jj}} V_n \qquad 3.4$$

Hence V_j can be eliminated from all n equations, leaving an (n-1)*(n-1) matrix. The elements at the k'th row of the new matrix can be written by using 3.4 as :

$$y_{ki}' = y_{ki} - \frac{y_{ji}}{y_{jj}} y_{kj} \qquad , \; i \neq j \qquad 3.5$$

For a two port network this process can be continued until the dimension of the reduced matrix is 3*3. In this resulting matrix one node is for input, one node is for output and the remaining one is for the reference

node or the common rail. Now if the voltage of the reference node is set equal to zero, 2*2 admittance matrix of the two-port network results (Fig. 3. 2).

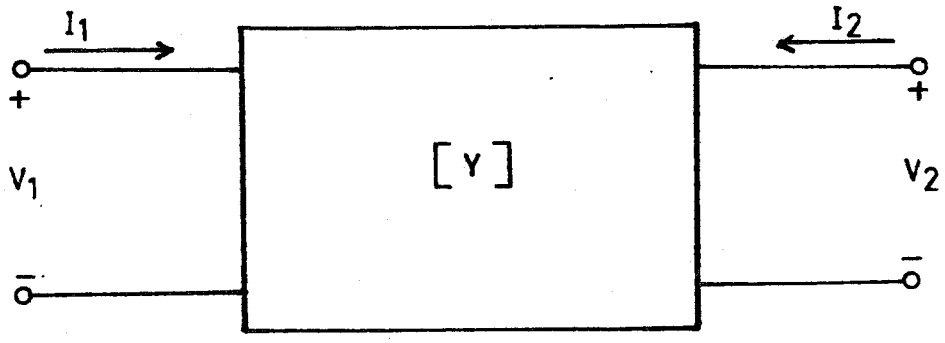

Figure 3. 2. Two-port Network with y-parameters

3. 4. GAIN IN TERMS OF Y-PARAMETERS

After reduction of the indefinite admittance matrix the two-port network can be modelled by its admittance or short circuit parameters as in Fig. 3. 3.

Now considering the broadband matching problem in Fig. 3. 4 the output impedance Z_{out} can be written in terms of the y-parameters as:

$$Z_{out} = \frac{y_{11} + Y_G}{D_y + y_{22}\, Y_G} \qquad 3.6$$

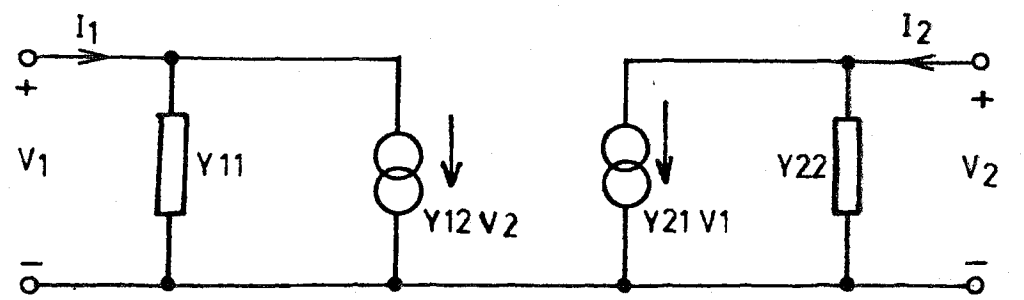

Figure 3.3. y-parameter Model of Two-ports

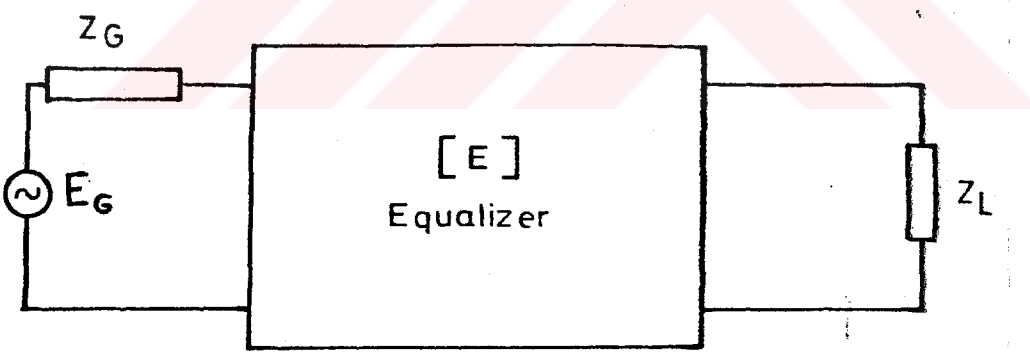

Figure 3.4. Double Matching

where $Y_G = 1/Z_G$ and $D_y = y_{11}\, y_{22} - y_{12}\, y_{21}$

The transducer power gain is given as:

$$G = 4\,\frac{Re(Z_{out})\; Re(Z_L)}{[Re(Z_{out})+Re(Z_L)]^2 + [Im(Z_{out})+Im(Z_L)]^2} \qquad 3.7$$

Up to now, it is shown how to obtain the gain values of a given two-port network between specified load and source impedances starting from the n terminal network description. In the next section, the optimization method of the element values for the BBM problem is given.

3.5 THE METHOD FOR ELEMENT OPTIMIZATION

If the given element values of a two-port equalizer are chosen as initials and the transducer gain is computed using y-parameters as objective function, the following computational steps can be outlined.

1. The indefinite admittance matrix of the n-terminal network is constructed with the current element values.

2. The indefinite admittance matrix is reduced and the y-parameters of the equalizer are constructed.

3. The gain values at each optimization frequency and hence the errors from the desired value are obtained.

4. A direction for the element values is chosen in order to minimize the error values at each optimization frequency.

5. New element values are obtained and the steps beginning from 1 are repeated until the errors fall in a tolerable range.

The optimization technique used in the element optimization part is the same as that of the SRFT part. The problems that cannot be solved by the approach used in this work may be handled by more sophisticated optimization routines.

CHAPTER 4

COMPUTER PROGRAM AND APPLICATIONS

4.1. INTRODUCTION

The BBM problem, when considered in a numerical manner is a nonlinear optimization problem. The problem consists of finding a set of unknowns related to the equalizer such that the objective function, the transducer power gain, is maximized. In the general problem of obtaining a specified gain response over a given bandwidth, the objective function becomes the difference between the desired and the actual transducer power gain values at optimization frequencies.

In this work, a gradient optimization is implemented for the solution of the BBM problem. In the gradient optimization methods, the optimum solution is approached by a convergent sequence of the unknowns. When the objective function to be minimized has more than one minimum, there is a great possibility of finding a suboptimal solution. In such a case the trial of more

than one initial point and choosing the best among the solutions avaliable are mandatory. The objective function in the BBM problem has this property, so the solution found may not be the global minimum. However, if a solution satisfies the design requirements, it can be taken as a feasible solution from the engineering point of view.

In the SRFT design methods outlined in Chapter 2, there are no constraints on the coefficients of the numerator polynomial of the input reflection factor of the resistively terminated equalizer. Hence this leads to an unconstrained nonlinear optimization problem. However when the elements are optimized as outlined in Chapter 3, the positiveness of the element values must be guaranteed. Hence the element optimization is a constrained nonlinear problem.

With a locally convergent optimization method, if good initial values can be chosen or known for element, the BBM problem can be solved in an unconstrained manner. Also a penalty function may be used for constraining the solution to positive values.

In the computer program developed, The Finite Difference Levenberg-Marquardt Least Squares Approximation is implemented for both SRFT and element optimization parts. The element optimization part is supported by a Fibonacci search section.

In this chapter, the Levenberg-Marquardt Technique is described first in 4.2. In section 4.3 the Fibonacci search part is explained. The explanation of the computer program is given in 4.4. Section 4.5 consists of the applications of the computer program.

4.2. THE FINITE DIFFERENCE LEVENBERG-MARQUARDT NONLINEAR LEAST SQUARES APPROXIMATION [5,18]

If m optimization frequencies and n unknowns are assumed, the BBM problem can be modelled with the following nonlinear least squares minimization. The unknowns are either coefficients as in Chapter 2, or element values as in Chapter 3.

$$\text{minimize} \quad e(x) = \sum_{i=1}^{m} |T_i(x) - T_{Oi}|^2$$

$$= \sum_{i=1}^{m} | f_i(x) |^2$$

$$= || F(x) ||^2 \qquad 4.1$$

where,

x : n dimensional vector containing the unknowns.

$T_i(x)$: the transducer power gain at each optimization frequency w_i.

T_{0i} : The desired transducer power gain at each optimization frequency w_i.

If the solution to the problem is denoted by x^*, then x^* is contained among the zeros of :

$$\frac{1}{2} \nabla e(x) = J(x)^T F(x) \qquad 4.2$$

where $\nabla e(x)$ denotes the gradient of $e(x)$ and $J(x)$ is the Jacobian matrix of $F(x)$.

The following sequence of approximations to x^* is shown to be convergent.

$$x_{K+1} = x_K - [\mu_K I + J(x_K)^T J(x_K)]^{-1} J(x_K)^T F(x_K) \qquad 4.3$$

where x_K is the K-th approximation to x^* and μ_K is a sequence of nonnegative real constants. I is n*n identity matrix.

When the derivatives are approximated numerically, the Finite Difference Levenberg-Marquardt approximating sequence is obtained.

$$x_{K+1} = x_K - [\mu_K I + J(x_K, h_K)^T J(x_K, h_K)]^{-1} J(x_K, h_K)^T F(x_K) \qquad 4.4$$

where the i-th, j-th element of $J(x_K, h_K)$ is given by,

$$\frac{f_i(x_K + h_K{}^j u_j) - f_i(x_K)}{h_K{}^j} \qquad 4.5$$

where u_j denotes the j-th unit column vector and μ_K and

h_K's are chosen as :

$$\mu_K = c \; || \; F(x_K) \; ||_{max} \qquad 4.6$$

$$c = \begin{cases} 10 \text{ whenever } & .6 \leq || \; F(x_K) \; ||_{max} \\ 1 \text{ whenever } & .1 \leq || \; F(x_K) \; ||_{max} \leq .6 \\ .1 \text{ whenever } & || \; F(x_K) \; ||_{max} \leq .1 \end{cases} \qquad 4.7$$

$$h_K{}^j = \min \{ \; || \; F(x_K) \; ||_{max}, \; \partial_K{}^j \; \} \qquad 4.8$$

where h_K and ∂_K are n dimensional vectors and the superscript j shows the j-th components of these vectors with,

$$\partial_K{}^j = \begin{cases} 10^{-9} & \text{if } | \; x_K{}^j \; | < 10^{-6} \\ .001 \; | \; x_K{}^j \; | & \text{if } 10^{-6} \leq | \; x_K{}^j \; | \end{cases} \qquad 4.9$$

By this choice of the new iteration point, the local convergence of the objective function is guaranteed. c is called the stability constant and decreased as the iterations get closer to x^*, that is, as the error becomes smaller. These choices of parameters lead to a good convergence characteristics.

4.3. FIBONACCI SEARCH [18]

In the element optimization part, if the boundaries of element values are known and good initial values are

available then optimization by Fibonacci search is possible. Since the Fibonacci search is a line search method, only one variable can be optimized at a time, the others being held constant. If more than one variable are to be optimized, then the solution will depend on the initial choices of the variables. It is because of this property that a good initial start must be available.

Fibonacci search is implemented by first initializing the variables in the given regions and then applying the method to all variables one at a time. The variables other than the one which is being optimized, are held constant. After a variable is optimized it is taken as constant until the search for all variables ends.

Assuming that there are n variables with lower and upper limits $L_1, .., L_n$ and $U_1, .., U_n$ respectively, the problem for optimizing the j-th variable is,

$$\text{minimize } e(x_j) = \sum_{i=1}^{m} | T_i(x_j) - T_{Oi}(x_j) |^2$$

$$= | f(x_j) |^2$$

$$\text{subject to : } \quad L_j \leq x_j \leq U_j \qquad 4.10$$

where m optimization frequencies are assumed and T_i and T_{Oi} are the same as described in section 4.2.

An important point here to note is that, x_j must have

only one relative minimum in the given region in order for the method to be correctly applied. If more than one relative minimum exists, the solution may be suboptimal, but in any case, the objective function after the technique is applied will be smaller than before.

The initial width of uncertainity is,

$$d_1 = U_j - L_j \qquad 4.11$$

If d_K denotes the width of uncertainity after k measurements, when a total of N measurements are to be made, d_K is given by :

$$d_K = \frac{F_{N-K+1}}{F_N} d_1 \qquad 4.12$$

where F_K's are the Fibonacci numbers generated by the following recurrence relation,

$$F_N = F_{N-1} + F_{N-2} \quad , F_0 = F_1 = 1 \qquad 4.13$$

The procedure for reducing the width of uncertainity to d_n is as follows: The first two calculations of objective function are made symmetrically at a distance of:

$$(F_{N-1} / F_N) d_1$$

from the ends of the initial interval.

According to which of these is of smaller value, an

uncertainity interval of width $d_2 = (F_{N-1} / F_N) d_1$ is determined. The third calculation point is placed symmetrically in this new interval with respect to the calculation point already in the interval. The result of this third calculation gives a width of uncertainity $d_3 = (F_{N-2} / F_N) d_1$. In general, each successive calculation point is placed in the current interval of uncertainity symmetrically with the point already existing in the interval. After all N calculations are completed, the smallest of the last two points gives the optimal value of the variable. This is different from the termination of the Fibonacci method but, if the number of calculations is large enough the difference will not be important.

4.4. COMPUTER PROGRAM

The computer program is an improved version of a SRFT lumped equalizer network design program written by E. Telatar and O. Arıkan for the fullfilment of B.S project in EEE [19]. The program is written in Turbo Pascal programming language. It consists of three main parts : SRFT for lumped equalizer design, SRFT for distributed equalizer design , and element optimization parts. The block diagram of the program is given in Fig. 4.1.

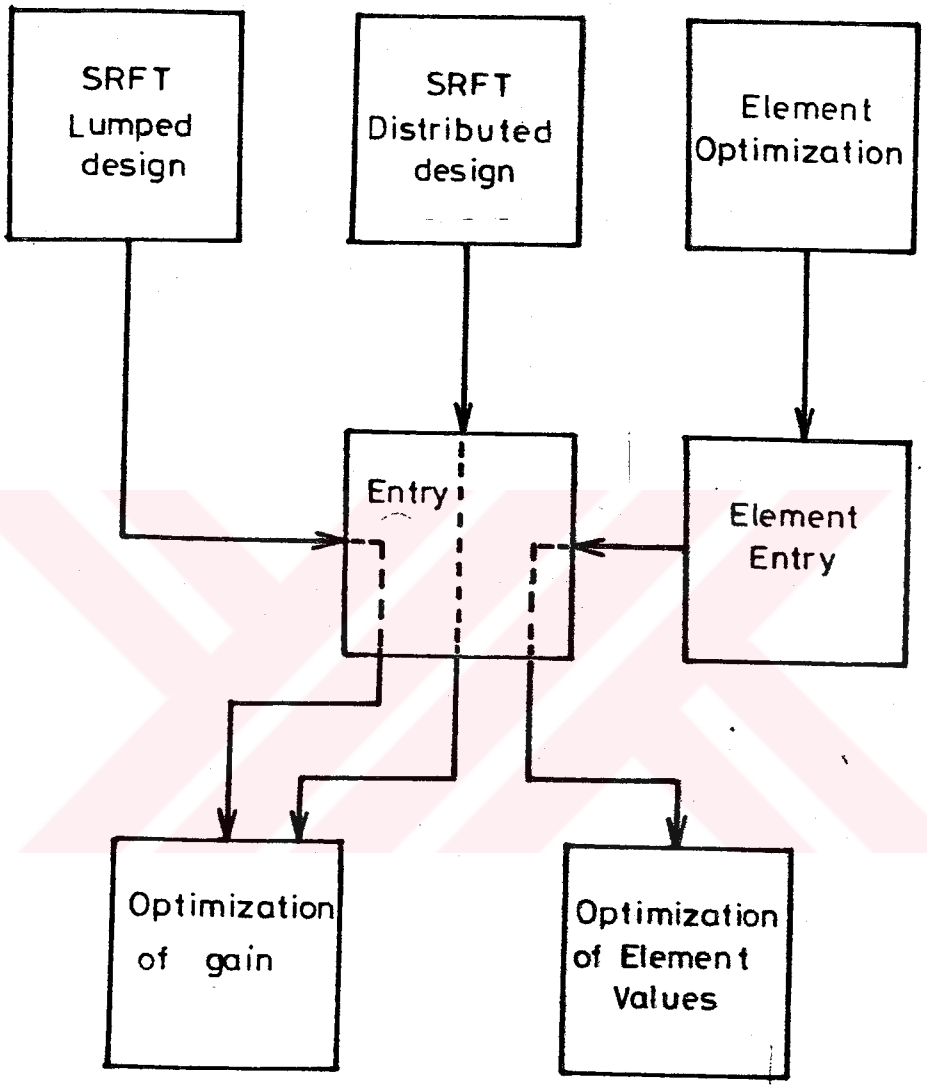

Figure 4.1. The Block Diagram of the Program

Generator and load impedance data with the desired gain response are entered in the entry section. In this section either scattering parameter or impedance based data with real-imaginary or magnitude-phase format may be entered. Saving the data and loading a pre-saved data utilities are also available.

Since the network topology is the outcome of the SRFT part, no input is required regarding the element values in these parts. The required data are the number and the location of the transmission zeros. Hence the degree of the equalizer network and the transmission zero characteristics are also entered in the entry section. Finite jw transmission zeros, with constant or variable location, may be added in this section.

The element optimization part requires the entry of the circuit topology and the element values. This is done in Entry1 section. Either variable or constant element values may be specified.

The block diagrams of the main parts of the program are given in Appendix B.

After the entry operation is completed, the optimization begins. In the optimization part a menu is available for controlling the optimization procedure. The utilities contained in the menu are :

I) Changing and observing the variables.

II) Changing the maximum tolerable error.

III) Changing the stability constants.

IV) Observing the gain values.

V) Stopping the optimization and returning to the main menu.

These utilities allow an interactive optimization procedure which becomes necessary as the complexity of the problems rises.

4.5. APPLICATIONS

EXAMPLE 1: It is required to construct a matching network to optimize the transducer power gain of the system in Fig. 4.2.

Inputs:

Generator and load networks are as shown in Fig. 4.2.

n=4; four element ladder.

k=2; two high-pass elements.

T_{01}=0.765; estimated gain level for each optimization frequency.

Initial guess of h_i's:

h_0=1, h_1=-1, h_2=1, h_3=-1, h_4=-1

Matching region: $0.3 \leq w \leq 1$ where w is the normalized frequency.

m=15; 15 evenly distributed optimization frequencies.

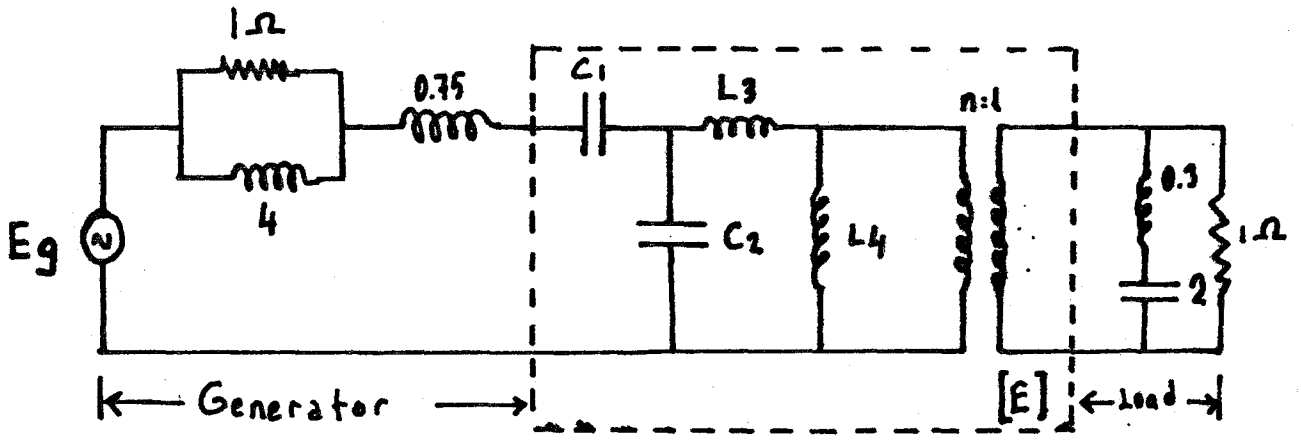

Figure 4.2. The Circuit of Example 1

Outputs:

$h_0=0.254$, $h_1=-0.365$, $h_2=1.365$, $h_3=-1.398$, $h_4=-0.79$

$T(-w^2)=0.765\pm0.0273$; obtained gain performance of the system.

The realization is not unique in this problem since two high-pass elements exist. A possible matching network is given as follows:

$C_1=2.28$, $C_2=2.26$, $L_3=0.717$, $L_4=1.024$, $n=1.024$

The return loss response of the matched network is given in Fig. 4.3.

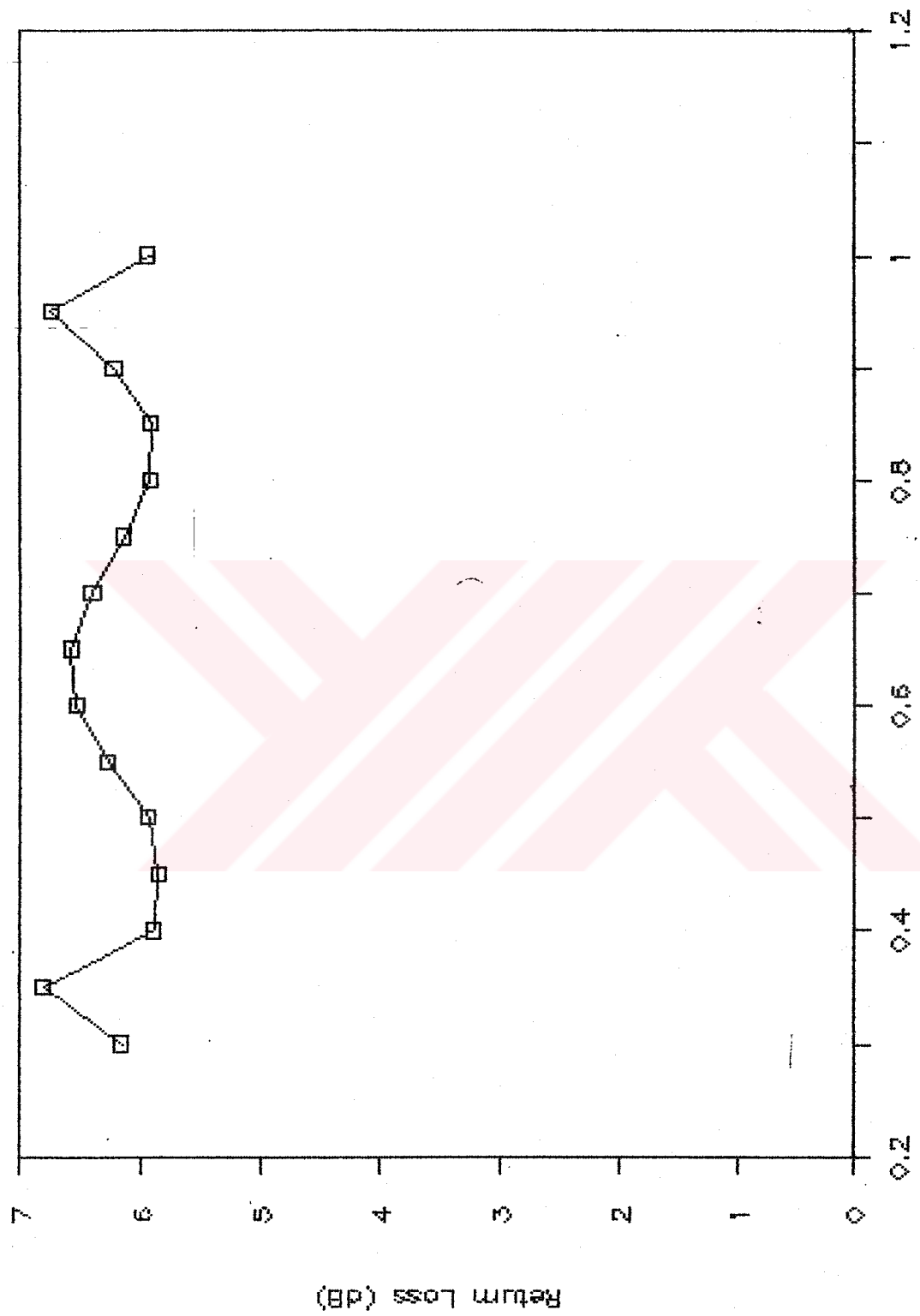

Figure 4.3. The Return Loss for Example 1

EXAMPLE 2:

a) It is desired to design a lumped element equalizer for the problem shown in Fig. 4. 4.

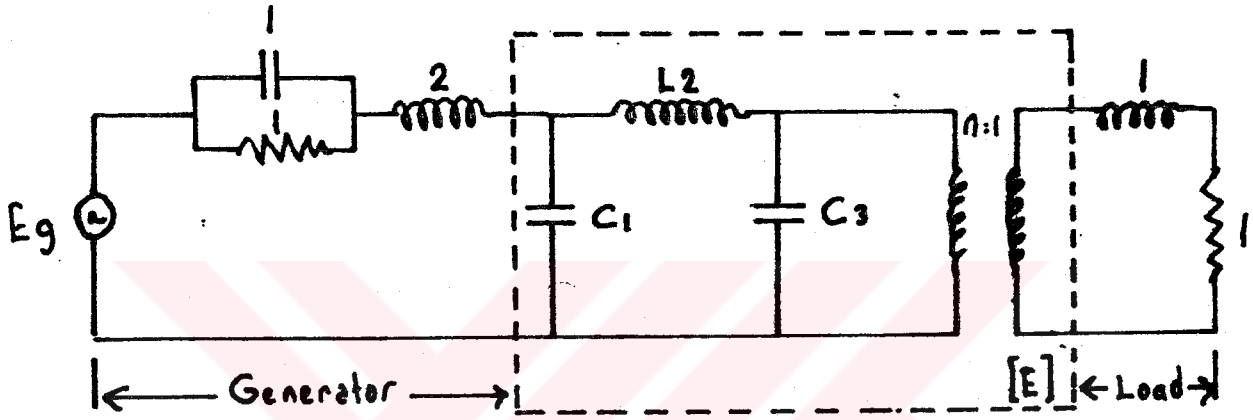

Figure 4. 4. The Circuit of Example 2. a

Inputs:

Generator and load networks are as shown in Fig. 4. 4.

n=3; three element ladder.

k=0; high-pass elements are not wanted but the transformer is allowed.

T_{01}=0. 96; estimated gain level of for each optimization frequency.

Initial guess of h_i's:

h_0=1, h_1=-1, h_2=1, h_3=-1

Matching region: 0 ≤ w ≤ 1 where w is the normalized frequency.

m=10; evenly distributed optimization frequencies.

Outputs:

h_0=0.139, h_1=-0.322, h_2=0.159, h_3=-2.093

$T(-w^2)$=0.96±0.0218; obtained gain performance after optimization.

The equalizer element values are:

C_1=1.322, L_2=2.475, C_3=1.113, n=1.15

The return loss of the matched system is given in Fig.4.5.

b) It is desired to design a distributed matching network as shown in Fig.4.6 for the same load and generator networks given in part (a) of the example.

Inputs:

n_d=3; three unit elements are specified.

q=1; one open-circuited shunt stub is allowed.

f_center=1; the normalized center frequency of the stop band region.

T_{01}=0.96; estimated gain level for each optimization frequencies.

Initial guess of H_i's:

H_1=1, H_2=-1, H_3=1, H_4=-1, with H_0 set to zero.

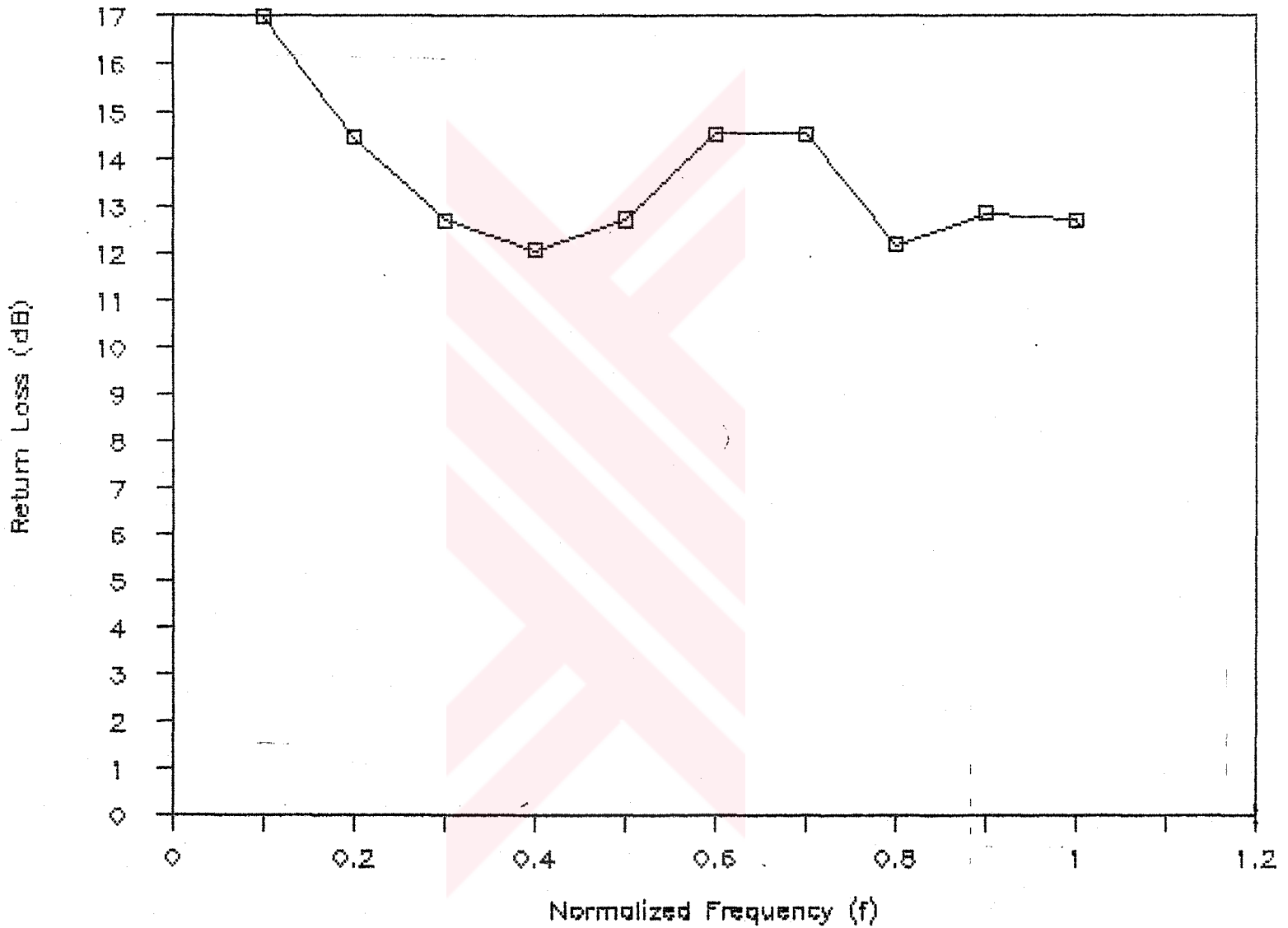

Figure 4.5. The Return Loss of Example 2.a

Matching region: 0 ≤ w ≤ 1 where w is the normalized frequency.

m=10; evenly distributed optimization frequencies.

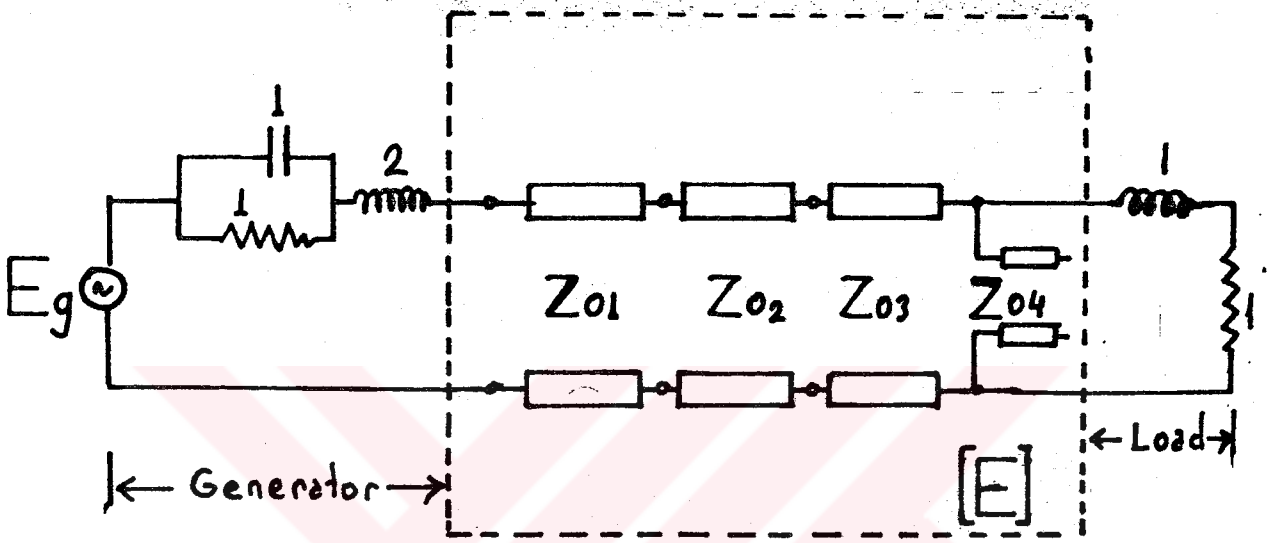

Figure 4.6. The Circuit of Example 2.b

Outputs:

H_1=-0.6804, H_2=-1.0387, H_3=1.1314, H_4=-1.3517

$T(-w^2)$=0.96±0.15; obtained gain performance after optimization. Here $T(-w^2)$≤1 relation is not violated, the gain deviation is small in the positive direction.

The equalizer element values are:

Z_{01}=1.4384, Z_{02}=0.3895, Z_{03}=1.3535, Z_{04}=1.8487

The return loss of the equalized system is shown in Fig. 4.7.

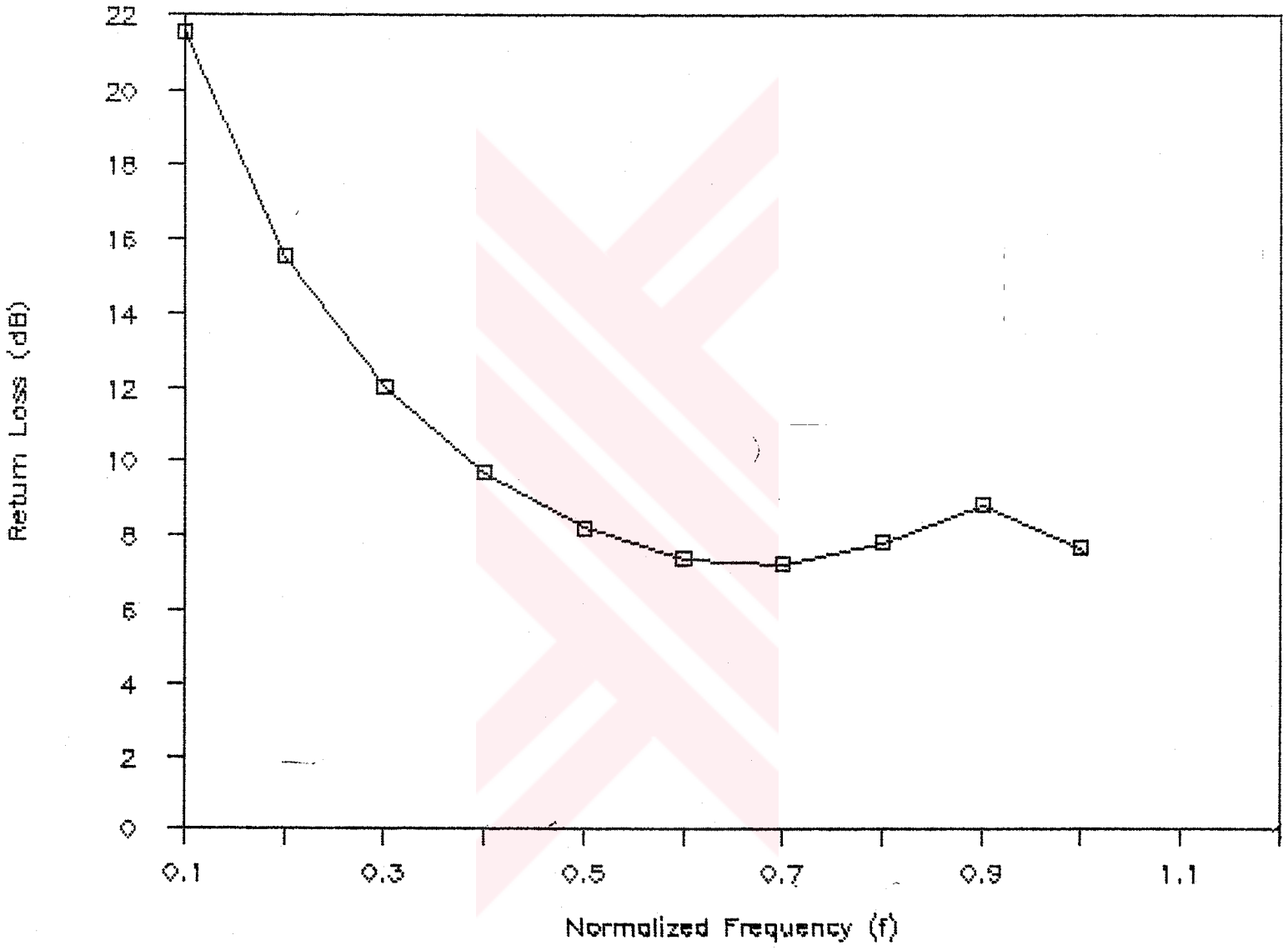

Figure 4.7. The Return Loss of Example 2.b

EXAMPLE 3: Distributed Low Pass Filter

In this example the transducer power gain values of a fifth order Chebychev filter consisting of unit elements only are entered as required gain data and the Chebychev capture effect of the program is tested.

Inputs:

$Z_G=Z_L=1$; Load and generator are pure resistances.

$n_d=5$; five unit elements are specified.

q=0; open-circuited stubs are not allowed.

Required gain values at the optimization frequencies are given in table 4.1.

Initial guess of H_i's:

$H_1=1$, $H_2=0$, $H_3=-1$, $H_4=0$, $H_5=1$, with H_0 set to zero.

Outputs:

$H_1=1.225$, $H_2=0$, $H_3=39.576$, $H_4=0$, $H_5=117.278$

The found values are close to the Chebychev coefficients which are:

$H_1=3.16$, $H_2=0$, $H_3=44.27$, $H_4=0$, $H_5=114.50$

$T(-w^2)=T_{01}\pm 0.06$ dB; obtained gain performance of the optimized network.

The resulting network is shown in Fig.4.8. Its return loss is given in Fig.4.9.

Freq. (GHz)	Gain (dB)
0.2	0
0.4	-0.06
0.6	-0.75
0.8	-1.17
1.0	-0.41
1.2	-10.68
1.4	-19.50
1.6	-25.70
1.8	-30.49
2.0	-34.11
2.2	-36.87
2.4	-38.91
2.6	-40.31
2.8	-41.14
3.0	-41.60

Table 4.1. Gain Values of Fifth Order Chebychev Filter

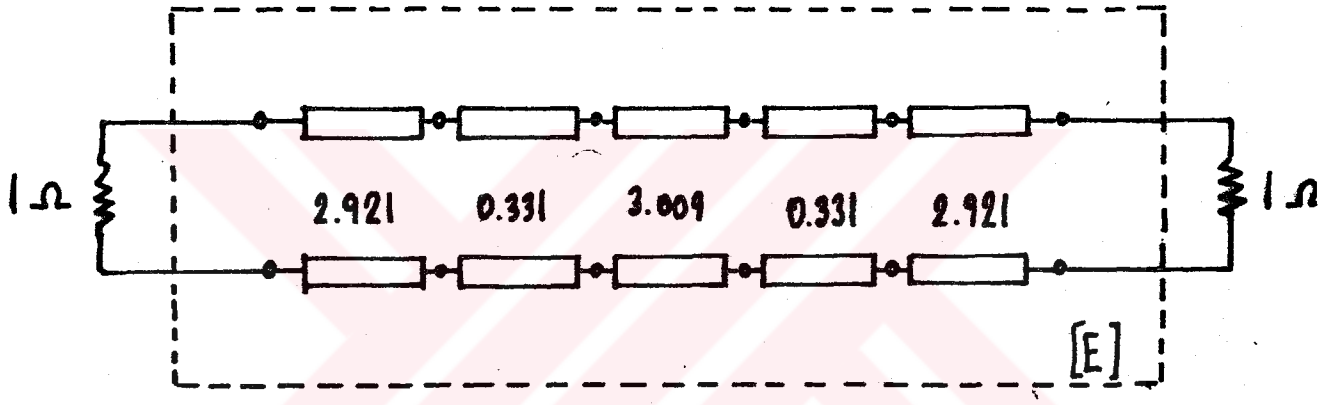

Figure 4.8. The Circuit of Example 3

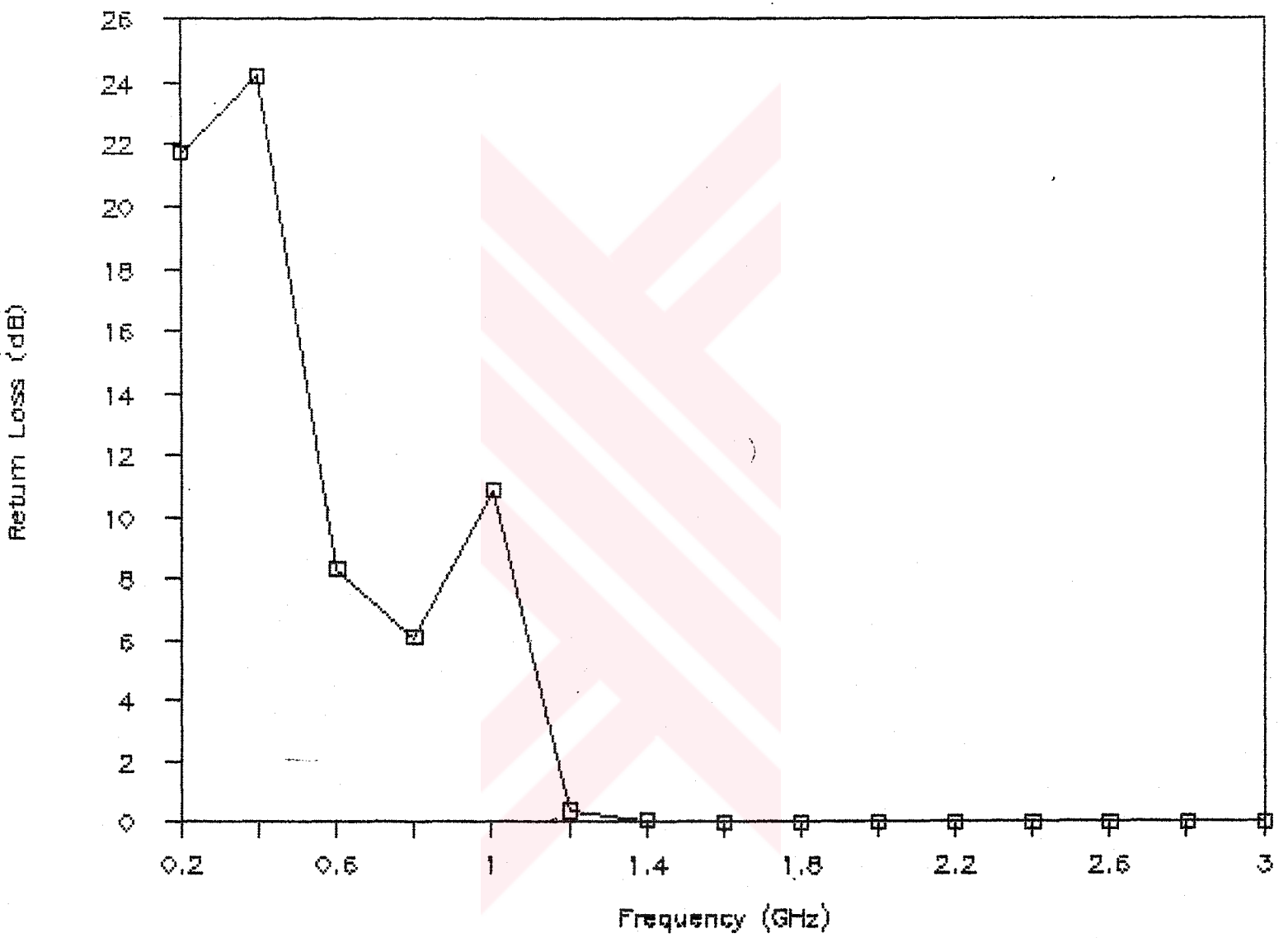

Figure 4.9. The Return Loss of Example 3

EXAMPLE 4: Distributed Broadband Transformer.

In this example a 10:1 broadband distributed transformer will be designed.

Inputs:

$Z_G=1$, $Z_L=10$

$n_d=2$; two unit elements.

q=1; one short circuited stub (High-pass design).

Passband region: 3 GHz ≤ f ≤ 6 GHz.

$T_{01}=1$; desired gain level.

f_center=4.5 GHz; center of the passband region (Quarter wave frequency).

Initial guess of H_i's:

$H_0=0.5$, $H_1=-0.5$, $H_2=0.5$, $H_3=5$

m=16; evenly distributed optimization frequencies in the passband region.

Outputs:

$H_0=1.2515$, $H_1=-3.0728$, $H_2=4.0503$, $H_3=0.8934$

Maximum deviation from T_{01} is 0.0028 after the optimization.

The transformer is shown in Fig.4.10 which has the return loss of Fig.4.11.

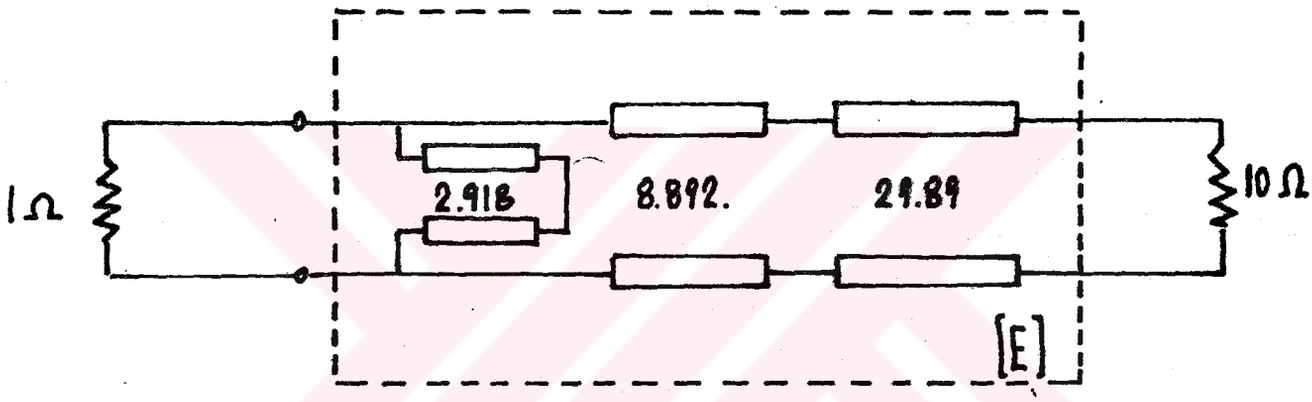

Figure 4.10. The Circuit of Example 4

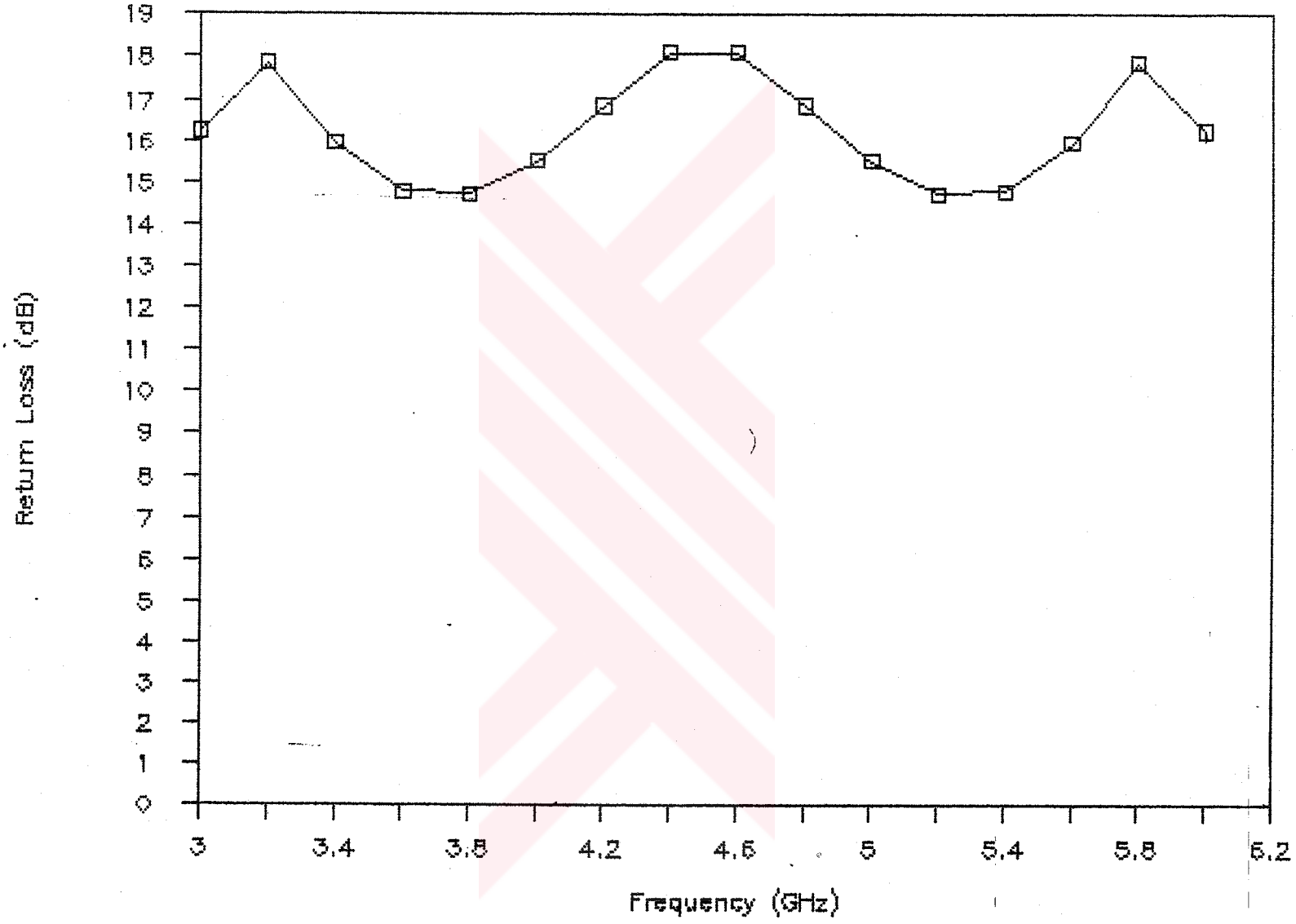

Figure 4.11. The Return Loss of Example 4

CHAPTER 5. CONCLUSIONS.

In this thesis work, a computer program is developed for the solution of the double matching problem using lumped or distributed equalizer element matching networks.

The Simplified Real Frequency Technique is extended to handle the design of commensurate distributed equalizers. The distributed networks that can be used in the program are two widely used prototypes; the low-pass and high-pass. Finite transmission zeros may be incorporated into these type of structures. The applications of the distributed design part of the program showed that it can be used in the design of distributed equalizers. Since couplers are not incorporated in the design procedure the performance of the program is not efficient as the lumped counterpart.

The performance of the lumped element equalizer design part is shown to be competent with the known analytical results of double matching problems.

Element optimization problem is found to be, as expected, a highly nonlinear problem. Unless a good initial point is known, the optimizer is found to be weak to handle the double matching problems. The reason of this is thought to be the types of the optimization method and the error function choice, together with the

highly nonlinear behaviour of the problem defined in terms of element values. However, the results of the SRFT parts may be improved using this part. The computer program developed in this work can be extended and strengthened in many ways. The element handling capacity of the distributed SRFT part can be enhanced, the element optimization part can be strengthened to handle the design of more general matching and filtering problems, the losses of the elements can be incorporated into design and so on. In these aspects, the work is believed to be the root of other studies.

LIST OF REFERENCES

1- Yarman, B. S. ; " Broadband Matching a Complex Generator to a Complex Load " , Ph. D. Thesis, 1982

2- Carlin, H. J. and Yarman, B. S. ; " The Double Matching Problem: Analytic and Real Frequency Solutions ", IEEE Trans. CAS, Jan. 1983, pp. 15-28.

3- Carlin, H. J. ; " Distributed Circuit Design with Transmission Line Elements ", IEEE Proc. Vol. 59, No 7, July 1971, pp. 1059-1081.

4- Yarman, B. S. ; " A Simplified Real Frequency Technique for Broadband Matching a Complex Generator to a Complex Load ", RCA Review Vol. 43, Sept. 1982, pp. 529-541.

5- Brown, K. M. and Dennis, J. E. ; " Derivative Free Analogues of the Levenberg-Marquardt and Gauss Algorithms for Nonlinear Least Squares Approximation ", Numer. Math. No 18, 1972, pp. 289-297.

6- Beasley, A. S; " Circuit Analysis by Small Computer ", Wireless World, Feb. 1980, pp. 38-41.

7- Beasley, A. S; " Circuit Analysis by Small Computer-2", Wireless World, April 1980, pp. 55-57.

8- Carlin, H. J. and Amstutz, P. ; " On Optimum Broadband Matching ", IEEE Trans. CAS, May 1981, pp. 401-405.

9- Youla, D. C; " A New Theory of Broadband Matching ", IEEE Trans. Circuit Theory, March 1964, pp. 30-50.

10- Carlin, H. J. ; " Scattering Matrix in Network Theory", IRE Trans. Circuit Theory, June 1956, pp. 88-97.

11- Yarman, B. S. ; " A Dynamic CAD Technique for Designing Broadband Microwave Amplifiers ", RCA Review, Vol. 44, Dec. 1983, pp. 551-565.

12- Temes, G. C. and Lapatra, J. W. ; " Introduction to Circuit Synthesis and Design ", McGraw-Hill Inc., 1977.

13- Carlin, H. J. and Komiak, J. J. ; " A New Method of Broadband Equalization Applied to Microwave Amplifiers", IEEE Trans. MTT, Feb. 1979, pp. 93-99.

14- Carlin, H. J. ; " A New Approach to Gain-Bandwidth Problems ", IEEE Trans. CAS, April 1977, pp. 170-175.

15- Yarman, B. S. and Carlin, H. J; " A Simplified Real Frequency Technique Applied to Broadband Multistage Microwave Amplifiers ", IEEE Trans. MTT, Dec. 1982, pp. 2216-2222.

16- Chen, W. K. ; " The Theory and Design of Broadband Matching Circuits ", Pergamon Press Inc. 1976.

17- Kuhn, N. ; " Simplified Signal Flow Graph Analysis " Microwave Journal, November 1963, pp. 59-66.

18- Luenberger, D. G. ; " Introduction to Linear and Nonlinear Programming ", Addison-Wesley Inc. 1973.

19- Telatar, E. and Arıkan, O. ; " The Solution of Double Matching Using the Simplified Real Frequency Technique", B. S. project in E. E. E., 1986.

APPENDIX A

ANALYTIC THEORY OF BROADBAND MATCHING

A. 1. INTRODUCTION

The analytic theory of BBM deals with finding a realizable equalizer networks for obtaining a preassigned transfer function between complex generator and load networks over a given frequency band. The double matching problem is shown in Fig. A. 1 for ease of reference.

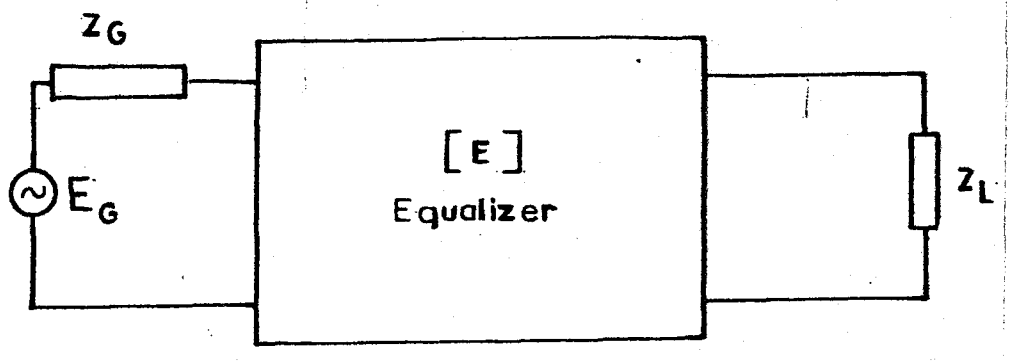

Figure A. 1. The Double Matching Problem

The analytic theory of BBM is necessary for defining and understanding of the problem.

Throughout the analysis unit-normalized scattering parameters will be used. These parameters are real normalized scattering parameters where the normalization resistance is 1 ohm as implied by the name.

A.2. POSITIVE REAL AND BOUNDED REAL PROPERTIES [1,16]

Positive real and bounded real properties are very important in network theory. In order an impedance to be physically realizable it must be positive real. For a scattering matrix to be a scattering matrix of a realizable network it must be bounded real.

Definition 2.2.1 : Z(s) represents a positive real (PR) matrix if,

i) $Z(s_*) = Z_*(s)$ in $Re[s] \geq 0$

ii) Z(s) is analytic in $Re[s] > 0$

iii) $Z(\sigma,w) = [Z(s) + Z^{T*}(s)] / 2$ is nonnegative definite in $Re[s] > 0$ and almost everywhere on $s=jw$.

If Z(s) represents a lossless network then,

iii) $Z(0,w) = [Z(jw) + Z(-jw)] / 2 = 0$

Definition 2.2.2 : S(s) represents a bounded real (BR) matrix if,

i) $S(s_*) = S_*(s)$ in $Re[s] \geq 0$

ii) $S(s)$ is analytic in $Re[s] > 0$

iii) $Q(\sigma, w) = I_n - S^{T*}(jw)\, S(-jw)$ is nonnegative definite in $Re[s] > 0$

If $S(s)$ represents a lossless network, then the constraint on $Q(\sigma, w)$ becomes :

iii) $Q(0, w) = I_n - S(jw)\, S(-jw) = 0$, and $S(s)$ is said to be para-unitary.

In the BBM problem 2-port networks are mostly encountered. So if the third of the BR properties is substituted for the scattering matrix of a lossless and reciprocal 2-port network, the following are obtained.

$$s_{11}(s)\, s_{11*}(s) + s_{21}(s)\, s_{21*}(s) = 1 \qquad \text{A. 1. a}$$

$$s_{22}(s)\, s_{22*}(s) + s_{21}(s)\, s_{21*}(s) = 1 \qquad \text{A. 1. b}$$

$$s_{11}(s)\, s_{11*}(s) = s_{22}(s)\, s_{22*}(s) \qquad \text{A. 1. c}$$

$$s_{22}(s) = - \frac{s_{11*}(s)\, s_{21}(s)}{s_{21*}(s)} \qquad \text{A. 1. d}$$

where $s_*(s)$ denotes $s(-s)$.

Hence at real frequencies, i.e at $s = jw$,

$$|s_{11}(jw)|^2 = |s_{22}(jw)|^2 = 1 - |s_{21}(jw)|^2 \qquad \text{A. 2}$$

A. 3. RELATION BETWEEN UNIT NORMALIZED AND YOULA-COMPLEX NORMALIZED SCATTERING PARAMETERS.

Since the complex terminations are very common, it is necessary to derive the input reflection coefficient of a lossless two-port network [N], when the output port of [N] is terminated by an arbitrary complex load (Fig. A. 2).

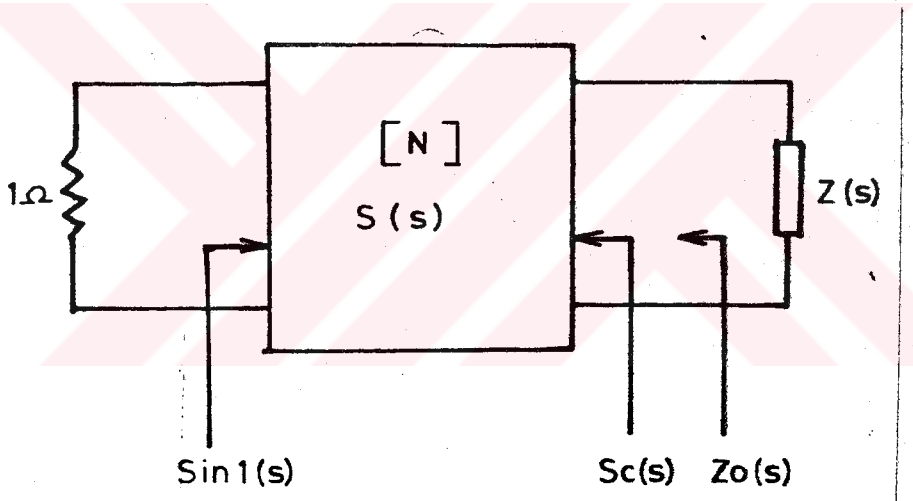

Figure A. 2. Two-port with Complex Load

Using scattering formulas,

$$S_{in1} = S_{11} + \frac{S_{21}^2 S}{1 - S S_{22}}$$

where $S = \dfrac{Z(s)-1}{Z(s)+1}$

Here s_{in1} can be written as,

$$s_{in1} = \frac{s_{11} - S\Delta}{1 - S\, s_{22}}$$

where $\Delta = s_{11}\, s_{22} - s_{21}^2$

$$= \frac{s_{21}}{s_{21*}}$$

Hence, $$s_{in1} = \frac{s_{21}}{s_{21*}} - \frac{S - s_{22*}}{1 - S\, s_{22}} \qquad \text{A. 3}$$

From A. 1. b, the following can be written.

$$s_{21}(s)\, s_{21}(-s) = 1 - s_{22}(s)\, s_{22}(-s)$$

Substituting the unit normalized output reflection coefficient which is given as,

$$s_{22} = \frac{Z_0(s)-1}{Z_0(s)+1}$$

the following is obtained.

$$s_{21}(s)\, s_{21}(-s) = 1 - \frac{Z_0 - 1}{Z_0 + 1} \cdot \frac{Z_{0*} - 1}{Z_{0*} + 1}$$

$$= \frac{2(Z_0 + Z_{0*})}{(Z_0 + 1)\,(Z_{0*} + 1)} \qquad \text{A. 4}$$

Now considering $(Z_O + Z_{O*}) / 2$,

If $Z_O = N / D$ (D is strictly Hurwitz) then,

$$\frac{Z_O + Z_{O*}}{2} = \frac{ND_* + DN_*}{2DD_*} \qquad \text{A. 5}$$

If the numerator of A. 4 is spectrally factorized then A. 4 can be rewritten as:

$$\frac{Z_O + Z_{O*}}{2} = \frac{AA_*}{DD_*} = FF_* \qquad \text{A. 6}$$

where $F=(A_*/D)$ and A has all its zeros on jw and in LHP. With the use of A. 6, A. 4 becomes,

$$s_{21}\, s_{21*} = \frac{4FF_*}{(Z_O + 1)\ (Z_{O*} + 1)} \qquad \text{A. 7}$$

This suggests that,

$$s_{21} = \frac{2F}{Z_O + 1} \qquad \text{A. 8}$$

With the substitution of A. 8 into A. 3, s_{in1} becomes,

$$s_{in1} = \frac{A_*}{A} \cdot \frac{D_*}{D} \cdot \frac{Z - Z_{O*}}{Z + Z_O} \qquad \text{A. 9}$$

Now let :

$$n = \frac{A_*}{A} \qquad \text{A. 10. a}$$

$$b = \frac{D_*}{D} \qquad \text{A. 10. b}$$

$$s_c = b\,\frac{Z - Z_{0*}}{Z + Z_0} \qquad \text{A. 10. c}$$

Then $s_{in1} = n\, s_c(s)$ A. 11

Here it is important to note that,

$$|s_{in1}(JW)| = |s_c(JW)| = |\overline{s_c}(JW)|$$

where $\overline{s_c} = \dfrac{Z - Z_{0*}}{Z + Z_0}$ A. 12

is defined as Youla complex normalized scattering parameter [9]. $s_c(s)$ is the corresponding regularized scattering parameter because RHP poles of $\overline{s_c}$ due to Z_{0*} are removed by $b = (D_* / D)$.

Hence with the use of A. 11 the unit normalized input reflection coefficient to the two-port network of Fig. A. 2 is related to the Youla scattering parameters. This result will be applied to BBM problem in the following sections.

A. 4. YOULA CONSTRAINTS [9]

Youla constraints are the necessary and sufficient

conditions on the complex scattering parameter, $\overline{s_c(s)}$

$$\overline{s_c(s)} = \frac{Z_0 - Z_*}{Z_0 + Z} \qquad \text{A. 13}$$

in order that $Z_0(s)$ be physically realizable.

Definition 2.4.1 : Let Z(s) be a non-Foster (Real part not strictly zero) PR impedance and;

$$r(s) = Ev\{Z(s)\}$$

$$\alpha(s) = \frac{r(s)}{Z(s)}$$

Then a zero of transmission $s_0 = \sigma_0 + jw_0$ with $\sigma_0 \geq 0$ of multiplicity K is said to be a zero of transmission of order K.

A zero of transmission s_0 of Z(s) of order K belongs to one of the following classes.

Class I : $\sigma_0 > 0$ which includes all the open RHP zeros of transmission.

Class II : $\sigma_0 = 0$ and $Z(jw_0) = 0$

Class III : $\sigma_0 = 0$ and $0 < Z(jw_0) < \infty$

Class IV : $\sigma_0 = 0$ and $|Z(jw_0)| = \infty$

Then,

$$Z_0(s) = \frac{2r(s)\ b(s)}{b(s) - s_c(s)} - Z(s) \qquad \text{A. 14}$$

is PR if and only if $\overline{s_c}(s)$ is a bounded real scattering coefficient satisfying the coefficient constraints given below. The constraints are on the coefficients of the Laurent series expressions of the following terms about a zero of transmission s_0.

$$b = \frac{D_*}{D} = \sum_{r=0}^{\infty} B_r\ (s - s_0)^r \qquad \text{A. 15}$$

$$F = 2r(s)\ b(s) = \sum_{r=0}^{\infty} F_r\ (s - s_0)^r \qquad \text{A. 16}$$

$$\overline{s_c} = b\ \frac{Z - Z_{0*}}{Z - Z_0} = \sum_{r=0}^{\infty} S_r\ (s - s_0)^r \qquad \text{A. 17}$$

then the coefficient constraints on the BR reflection coefficient $\overline{s_c}(s)$ are stated as :

Class I: $B_r = S_r \qquad r = 0, 1, 2, \ldots, K-1$ A. 18

Class II: $B_r = S_r \qquad r = 0, 1, 2, \ldots, K-1$ and,

$$(B_K - S_K) / F_{K+1} \geq 0 \qquad \text{A. 19}$$

Class III: $B_r = S_r \qquad r = 0, 1, 2, \ldots K-2$ and,

$$(B_{K-1} - S_{K-1}) / F_K \geq 0 \qquad \text{A. 20}$$

Class IV: $B_r = S_r \qquad r = 0, 1, 2, \ldots, K-1$ and,

$$(F_{K-1}) \;/\; (B_K - S_K) \geq 0 \qquad \text{A. 21}$$

An important point to note here is that, $\overline{s_c}(s)$ defined in A. 13 is different from that defined in A. 12. This difference occurs because of the different power transfer consideration in the two equations [9]. This will be particularly useful for reducing the double matching problem to two sets of coefficient constraint equations as will be described in the next section.

A. 5. REDUCTION OF THE DOUBLE MATCHING PROBLEM TO YOULA CONSTRAINTS.

A non-Foster PR impedance can be realized as the input impedance of a lossless two-port network terminated in a resistor [12]. Using this principle the doubly terminated structure (Fig. A. 1) can be transformed into that of Fig. A. 3.

Considering Fig. A. 3, the transducer gain from port 1 to port 2 is equal to that from port 3 to port 4, so the imaginary ports 1 and 2 can be used as the starting point.

Using A. 10. c and A. 11 and considering Fig. A. 4 and Fig. A. 5 the following can be written:

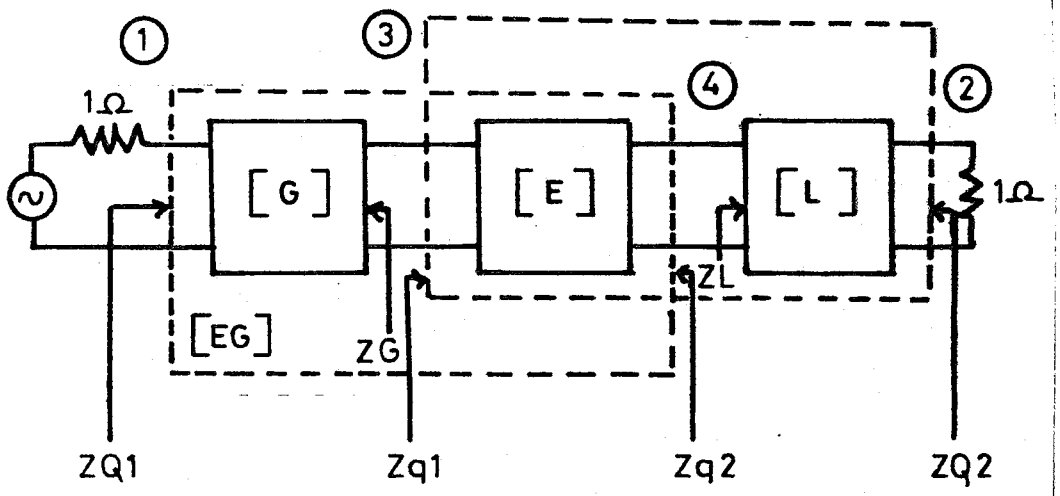

Figure A. 3. Darlington Representation

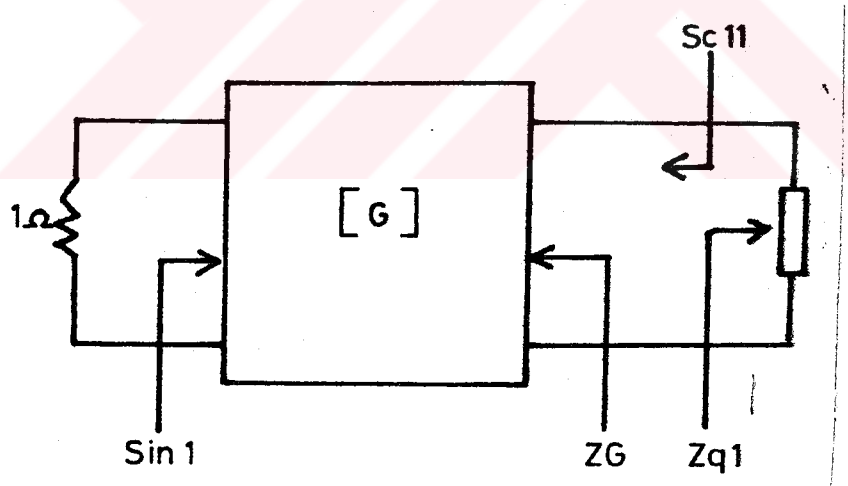

Figure A. 4. Circuit for Generator Extraction

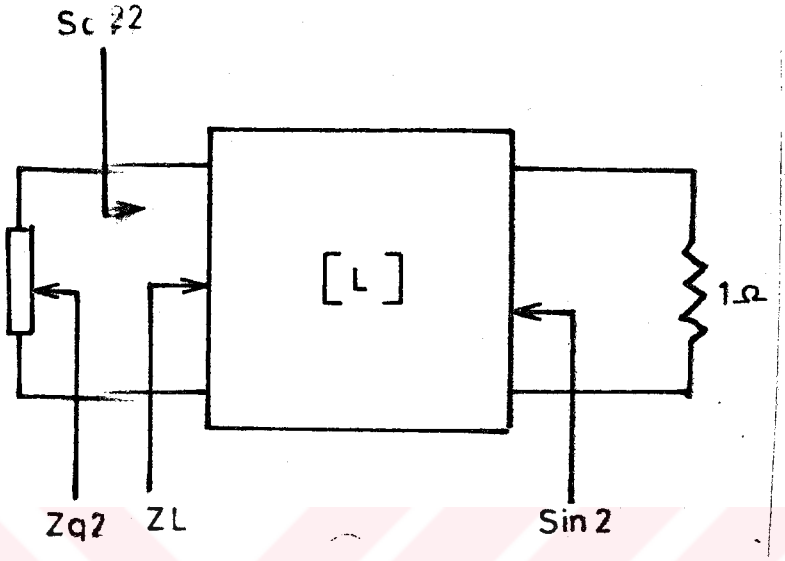

Figure A. 5. Circuit for Load Extraction

$$s_{c11}(s) = P_1 \frac{Z_{q1} - Z_{G*}}{Z_{q1} + Z_G} \qquad \text{A. 22}$$

$$\text{and} \quad s_{c22} = P_2 \frac{Z_{q2} - Z_{L*}}{Z_{q2} + Z_L} \qquad \text{A. 23}$$

where $Z_{q1} = N_1 / D_1$: input impedance to [EL],

$Z_{q2} = N_2 / D_2$: input impedance to [EG],

$P_i = D_{i*} / D_i$, $i = 1, 2$

1 for Z_G and 2 for Z_L and,

$$s_{in1} = \frac{Z_{q1} - 1}{Z_{q1} + 1} = n_G\, s_{c11}$$

where n_G is an all pass function constructed by RHP zeros of $r_G(s)$. Similarly $s_{in2} = n_L\, s_{c22}$ where n_L is constructed by RHP zeros of $r_L(s)$.

Considering Fig. A. 4 the following conclusions can be made:

I) $s_{in1}(s)$ is related to $s_{c11}(s)$ previously as:

$$s_{in1}(s) = n\, s_{c11}(s)$$

II) $\overline{s_c}(s) = \dfrac{Z_{q1} - Z_{G*}}{Z_{q1} + Z_G}$ is Youla complex scattering parameter which is used in A. 11.

III) Considering section A. 3 and the equation A. 13 it can be concluded that, the Youla coefficient constraints may be applied to $\overline{s_c}(s)$ which is given in II to leave a realizable Z_{q1}.

After the coefficient constraints are applied the remaining network becomes that of Fig. A. 6. Next the scattering parameters of the new network are found and the coefficient constraints are applied at port - 2 of Fig. A. 6 to extract [L], leaving a realizable Z_{q2}. Since

this is the input impedance of the equalizer terminated in a 1 Ω resistor, it can be realized by Darlington synthesis procedure [16].

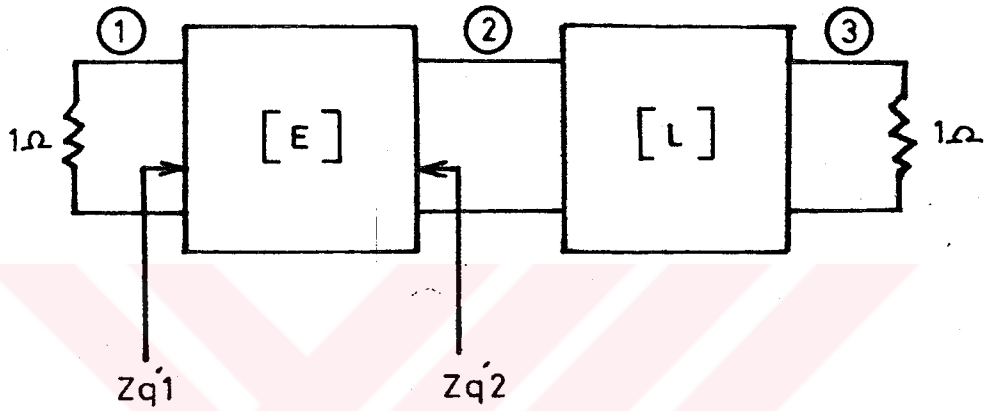

Figure A. 6. Reduced Matching Problem

Although the procedure outlined above consists of two steps in terms of scattering parameter evaluation, it is shown by Yarman [1] that the two extractions can be made simultaneously. This is achieved by applying the coefficient constraints to both the s_{c11} and s_{c22} of the initial network. This property is easily seen if the constraints are written for the two cases and compared.

Now with these results the procedure for analytic BBM can be given.

A. 6. ANALYTIC PROCEDURE FOR BROADBAND MATCHING [1]

The following steps may be given for the procedure :

I) A transfer function $T(w^2)$ having the same transmission zeros of both generator and load impedances with at least the same multiplicity is chosen. The parameters of the transfer function are to be found to optimize the performance measure.

So $s_{21}(s)$ will in general be of the form :

$$s_{21}(s) = n_G\, n_L \frac{f_*}{g}\, \mu(s)$$

where $\mu(s)$ is an all-pass function which may be required to satisfy the Youla coefficient constraints. f_* has all its zeros in the closed RHP. n_G and n_L are obtained by the RHP zeros of generator and load impedances respectively, as indicated previously. If non-distinct RHP zeros of transmission of generator and load impedances exist then they cannot be inserted to s_{21} as all-pass functions, since coefficient constraints cannot be satisfied in that case. Hence common RHP transmission zeros are inserted in $s_{21}(s)$ but not in all-pass form. These type of zeros emerge as all-pass in either $s_{11}(s)$ or $s_{22}(s)$ due to scattering equations.

Then,

$$s_{22}(s)\ s_{22}(-s) = 1 - s_{21}(s)\ s_{21}(-s)$$

$$= 1 - T(-s^2) \quad \text{is formed,}$$

and

$s_{22}(s) = h / g$ is obtained from the spectral factorizaton of :

$$hh_* + ff_* = gg_*$$

II) Consistent pair of scattering parameters $s_{11}(s)$ and $s_{22}(s)$ is defined.

$$s_{22}(s) = \Gamma \frac{h}{g} n_G\ n_L\ \mu$$

$$s_{11}(s) = - \Gamma \frac{h_*}{g} n_G\ n_L\ \mu$$

where Γ is a sign term ($\Gamma = \pm 1$) and n_G and n_L contain all the zeros of transmission whether distinct or not.

III) Youla complex normalized scattering parameters are obtained and coefficient constraints are satisfied at both generator and the load side.

$$s_{c22}(s) = \Gamma \frac{h}{g} n_G\ \mu$$

$$s_{c11}(s) = - \Gamma \frac{h_*}{g} n_L\ \mu$$

IV) Assuming the generator extraction first, the unit normalized scattering parameters of the new network [EL] are computed (Fig. A. 3).

$$s_{22}(s)' = s_{22} \frac{1 - g_{11*} / s_{11*}}{1 - g_{11*} s_{11}}$$

where $g_{11} = \frac{Z_{G*} - 1}{Z_G + 1} n_G b_1$

and $Z_G = N_G / D_G$, $b_1 = D_{G*} / D_G$

and the new Youla reflection factor $s_{c22}(s)$ is given by

$$s_{c22}(s)' = s_{22} \frac{1 - g_{11*} / s_{11*}}{1 - g_{11*} s_{11}}$$

After these, the Youla constraints are applied to $s_{c22}(s)$.

V) The input impedance Z_{q2} to the resistively terminated [E] is obtained by,

$$Z_{q2}(s) = \frac{2r_L(s)\, b_2(s)}{b_2(s) - s_{c22}'(s)} - Z_L(s)$$

and synthesized using Darlington procedure.

Here it is possible to extract the load impedance first and then continue with the extraction of the generator impedance.

A. 7. EVALUATION OF ANALYTIC THEORY [1, 2, 8, 14]

Analytic BBM method can be applied to problems where the load and generator impedances are known in an analytical form as a function of $s = \sigma + jw$. If this is not the case approximations to the measurement data of generator and load impedances must be made. A practical difficulty is posed here, since the same data can be approximated by many functions and the gain-bandwidth constraints depend on the forms of these functions.

After this, another difficulty arises in choosing the transfer function which will optimize the performance measure and having the necessary transmission zeros of the load and generator networks. Also as the degrees of generator, load and the equalizer are increased the analysis becomes extremely difficult.

These difficulties are the reasons for the developed computerized methods and the results obtained and given in the literature show that the computerized methods are more efficient than the analytical ones.

APPENDIX B. BLOCK DIAGRAMS OF PROGRAM

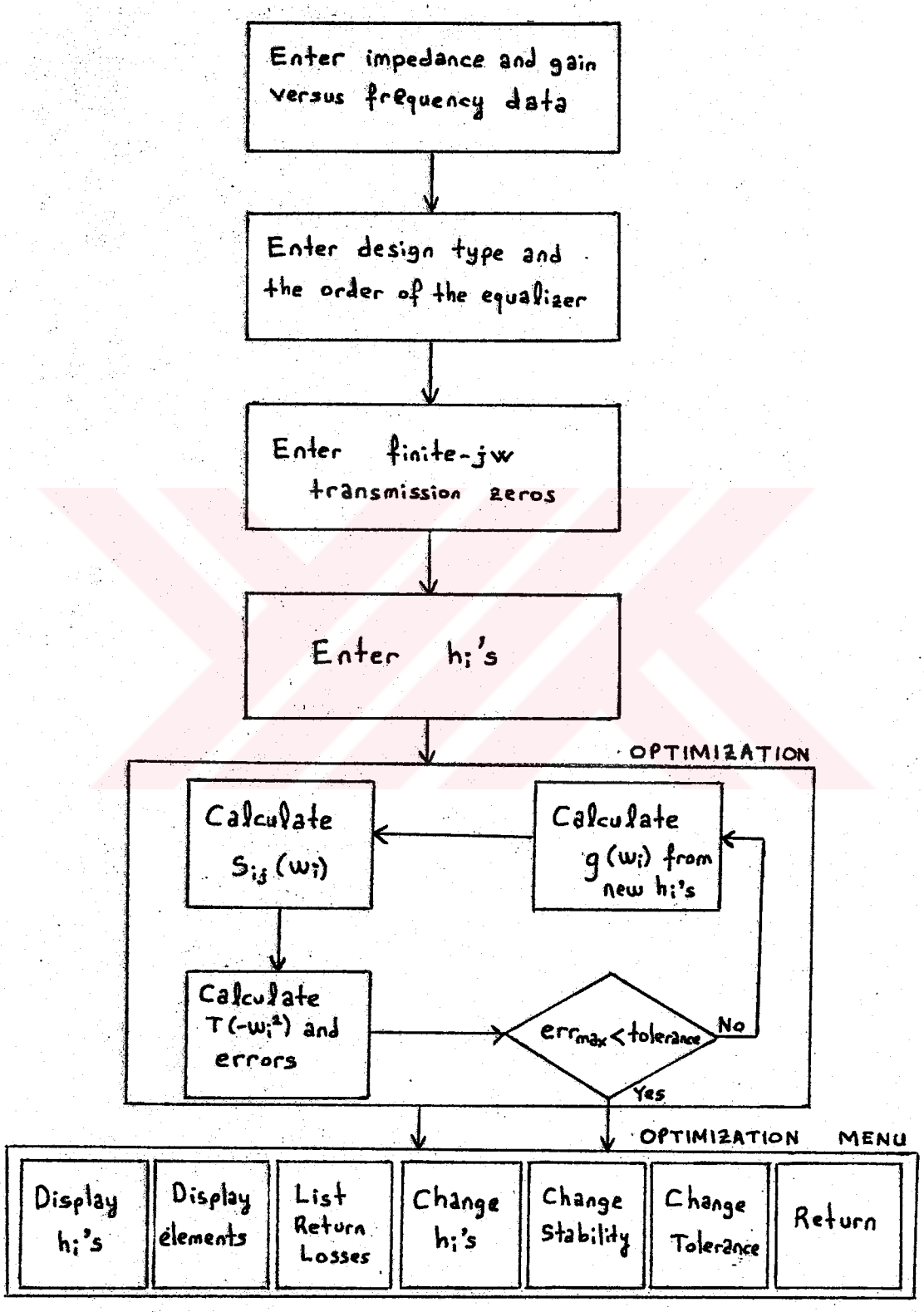

Figure B.1. Block Diagram for Lumped Design.

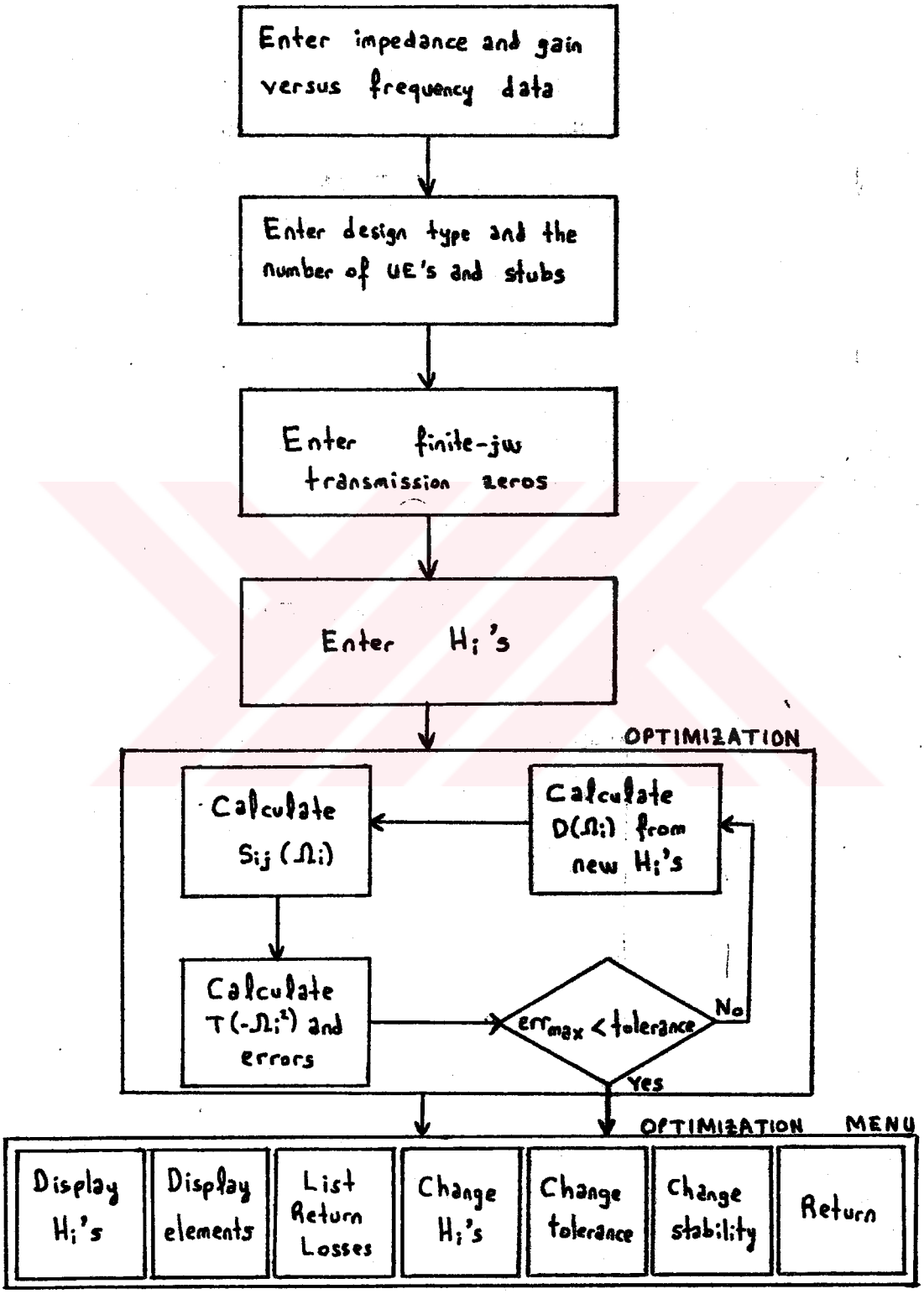

Figure B. 2. Block Diagram for Distributed Design

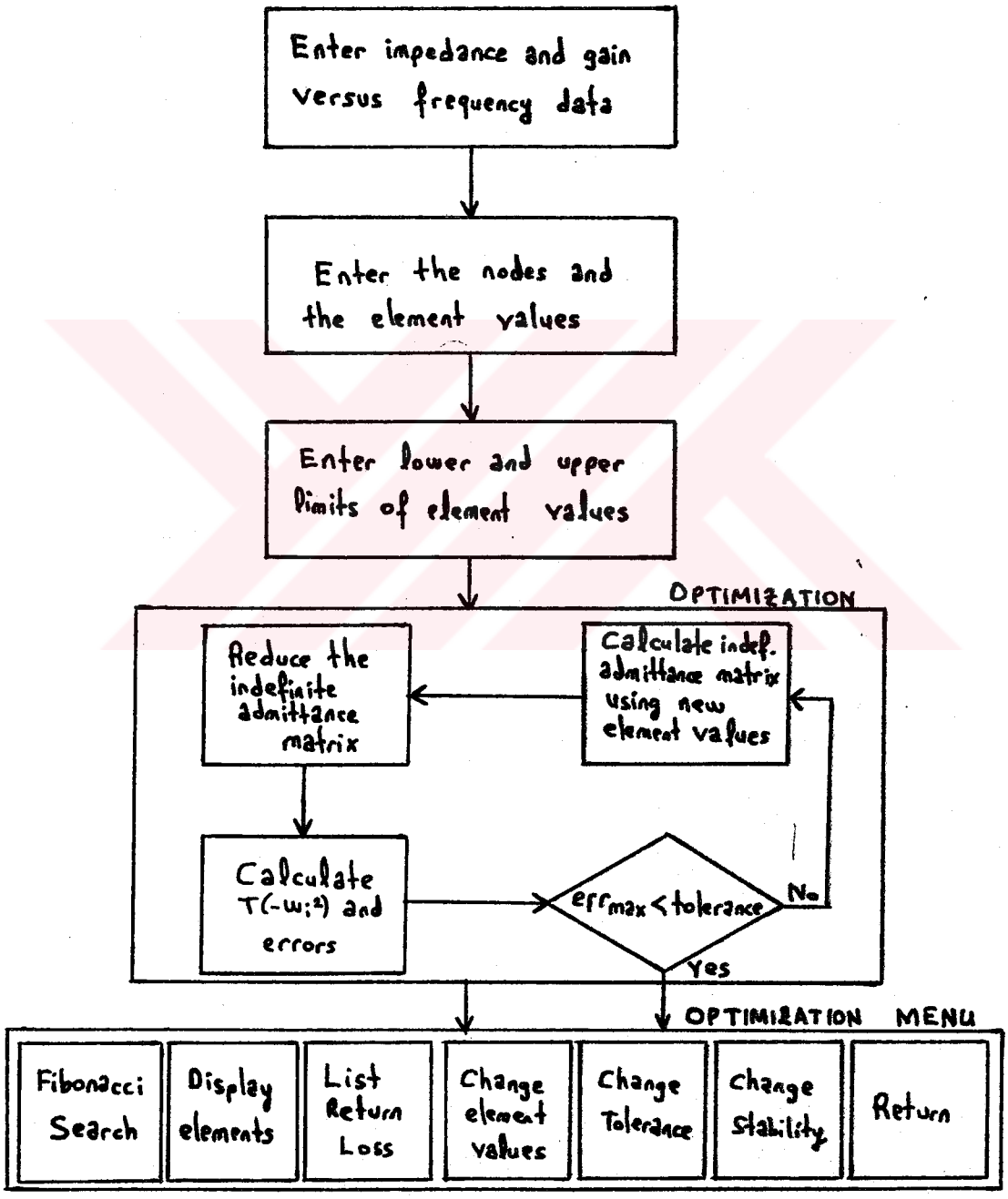

Figure B.3. Block Diagram for Element Optimization

APPENDIX C. LISTING OF THE PROGRAM.

```
PROGRAM MATCH (INPUT,OUTPUT);
type            vector = array[1..40] of real;
                matrix = array[1..40] of vector;
           polynomial = array[0..39] of real;
                intarr = array[1..40] of integer;

var             distelmt,lumpelmt,finish : boolean;
                                     key : char;
                 {$I entry1.pas}
                 {$I enter.pas}


function mlp(n:integer) : integer;
begin
  if odd(n) then
    mlp:=-1
  else
    mlp:=+1;
end;{Function mlp}

function fact( n:integer): integer;
begin
if n=0 then
fact:=1
else
fact:=fact(n-1)*n;
end; {function fact}

function power(x:real;n:integer): real;
begin
  if n=0 then
    power:=1
  else
    power:=x*power(x,n-1);
end;{Function power}

procedure evalpoly(p:polynomial;n:integer;z_re,z_im:real;var p_re,p_im:real);

var           i : integer;
    p_are,p_aim : real;

begin
  p_re:=0.; p_im:=0.;
  for i:=n downto 0 do
  begin
    p_are:=p_re*z_re-p_im*z_im; p_aim:=p_re*z_im+p_im*z_re;
    p_re:=p_are+p[i]; p_im:=p_aim;
  end;
end; {Procedure evalpoly}

procedure polyroot(var a:polynomial;ip:integer;var xx,yy:vector);

label   70,80,90,170,190,210,230,240,250,260,280,300,320;
```

```
var                 c : polynomial;
                   it : boolean;
 k,n,nx,ny,i,m,kt,ic,j : integer;
       x,y,x0,y0,x2,y2,
     xt,yt,xp,yp,dx,dy,
      ux,uy,u,v,t,s,al : real;

    begin
      it:=false; n:=ip;
      if n<=0 then goto 300;
      if a[n]=0 then goto 300;
      nx:=ip; ny:=n; j:=1; k:=n;
      for i:=0 to k do
        c[k-i]:=a[i];
70:   x0:=5.00101E-03; y0:=1.000101E-02; m:=0;
80:   x:=x0; x0:=-10*y0; y0:=-10*x; x:=x0; y:=y0; m:=m+1; ic:=0;
90:   ux:=0; uy:=0; u:=c[n]; v:=0; xt:=1; yt:=0;
      if u=0. then goto 230;
      for i:=1 to n do
      begin
        t:=c[n-i];x2:=x*xt-y*yt;y2:=x*yt+y*xt;u:=u+t*x2;v:=v+t*y2;
        ux:=ux+i*t*xt;uy:=uy-i*t*yt;xt:=x2;yt:=y2;
      end;
      s:=sqr(ux)+sqr(uy);
      if s=0. then goto 190;
      dx:=(v*uy-u*ux)/s; dy:=-(u*uy+v*ux)/s; x:=x+dx; y:=y+dy;
      if (abs(dx)+abs(dy))<1.E-07 then goto 170;
      ic:=ic+1;
      if ic<1500 then goto 90;
      if it then goto 170;
      if m<5 then goto 80;
      goto 300;
170:  for i:=0 to ny do
      begin
        t:=a[k-i]; a[k-i]:=c[i]; c[i]:=t;
      end;
      kt:=n; n:=nx; nx:=kt;
      if it then goto 210;
      it:=true;xp:=x;yp:=y;goto 90;
190:  if not it then goto 80;
      x:=xp;y:=yp;
210:  it:=false;
      if abs(y)<1.E-4*abs(x) then goto 240;
      al:=2*x;s:=sqr(x)+sqr(y);n:=n-2;goto 250;
230:  x:=0;nx:=nx-1;ny:=ny-1;
240:  y:=0;s:=0;al:=x;n:=n-1;
250:  c[1]:=c[1]+al*c[0];
      for i:=2 to n do
        c[i]:=c[i]+al*c[i-1]-s*c[i-2];
260:  xx[j]:=x;yy[j]:=y;j:=j+1;
      if s=0 then goto 280;
      y:=-y;s:=0;goto 260;
280:  if n>0 then goto 70;
```

```
      goto 320;
300:  writeln('Unable to compute after 1500 iterations');
320:end;{Procedure polyroot}

procedure polymult(var m1:integer;m2:integer;f,f2:polynomial; var ff:polynomial);
var           i,j : integer;
begin
  for i:=0 to m1+m2 do
  begin
    ff[i]:=0;
    for j:=0 to i do
      ff[i]:=ff[i]+f[j]*f2[i-j];
    end;
    m1:=m1+m2;
  end;{procedure polymult}

overlay procedure g_cal(var g_re:polynomial; h:polynomial;n,k:integer;
          var_zero,multip:intarr; zero:vector; nuvzer, num_zero:integer;
          zerflg:boolean);

var                f,ff,f2,g_im,hh : polynomial;
                      rt_re,rt_im : vector;
                            r,x,y : real;
                 m1,m2,p,l,i,j,k1,k2 : integer;
begin
  for i:=0 to n do
  begin
    hh[i]:=0.;
    if 2*i > n then
    begin
      k1:=2*i-n; k2:=n;
    end
    else
    begin
      k1:=0; k2:=2*i;
    end;
    for j:=k1 to k2 do
      hh[i]:=hh[i]+mlp(j)*h[j]*h[2*i-j];
  end;
  ff[0]:=1; m1:=0;
  if zerflg=true then
  begin
    for p:=1 to num_zero do
    begin
      m2:=2*multip[p];
      for l:=0 to m1+m2 do begin f[l]:=0; f2[l]:=0; end;
      for l:=0 to m1 do f[l]:=ff[l];
      for l:=0 to m2 do
        f2[l]:=fact(m2)/(fact(l)*fact(m2-l))*power(zero[p],(2*m2-2*l));
      polymult(m1,m2,f,f2,ff);
    end;
  for i:=0 to m1 do hh[i+k]:=hh[i+k]+mlp(k)*ff[i];
  end
```

```
  else
  hh[k]:=hh[k]+mlp(k);
  polyroot(hh,n,rt_re,rt_im);
  g_re[0]:=sqrt(abs(hh[n])); g_im[0]:=0.;
  for i:=1 to n do
  begin
    r:=sqrt(sqr(rt_re[i])+sqr(rt_im[i]));
    if rt_im[i]<0. then
      rt_im[i]:=-sqrt(abs((r-rt_re[i])/2))
    else
      rt_im[i]:=sqrt(abs((r-rt_re[i])/2));
    rt_re[i]:=sqrt(abs((r+rt_re[i])/2));
    g_re[i]:=0.; g_im[i]:=0.;
  end;
  for i:=1 to n do
  begin
    for j:=i downto 1 do
    begin
      x:=g_re[j-1]+rt_re[i]*g_re[j]-rt_im[i]*g_im[j]; y:=g_im[j-1]+rt_re[i]*g_im[j]+rt_im[i]*g_re[j];
      g_re[j]:=x; g_im[j]:=y;
    end;
    x:=rt_re[i]*g_re[0]-rt_im[i]*g_im[0]; y:=rt_re[i]*g_im[0]+rt_im[i]*g_re[0];
    g_re[0]:=x; g_im[0]:=y;
  end;
end;{Procedure g_cal}

overlay procedure gcal_dist(var g_re:polynomial; h:polynomial;n,q,
ladder,nd:integer;var_zero,multip:intarr;zero:vector;nuvzer,num_zero:integer;
zerflg:boolean);

var                f,ff,f2,g_im,hh : polynomial;
                      rt_re,rt_im : vector;
                            r,x,y : real;
           m1,m2,p,l,i,j,k1,k2 : integer;
begin
  for i:=0 to n do
  begin
    hh[i]:=0.;
    if 2*i > n then
    begin
      k1:=2*i-n; k2:=n;
    end
    else
    begin
      k1:=0; k2:=2*i;
    end;
    for j:=k1 to k2 do
      hh[i]:=hh[i]+mlp(j)*h[j]*h[2*i-j];
  end;
  for i:=0 to nd do ff[i]:=mlp(i)*fact(nd)/(fact(nd-i)*fact(i));
  m1:=nd;
  if zerflg=true then
    for p:=1 to num_zero do
```

```
    begin
      m2:=2*multip[p];
      for l:=0 to m1+m2 do begin f[l]:=0; f2[l]:=0; end;
      for l:=0 to m1 do f[l]:=ff[l];
      for l:=0 to m2 do
        f2[l]:=fact(m2)/(fact(l)*fact(m2-l))*power(zero[p],(2*m2-2*l));
      polymult(m1,m2,f,f2,ff);
    end;
  for i:=0 to m1 do hh[i]:=hh[i]+ff[i];
  polyroot(hh,n,rt_re,rt_im);
  g_re[0]:=sqrt(abs(hh[n])); g_im[0]:=0.;
  for i:=1 to n do
  begin
    r:=sqrt(sqr(rt_re[i])+sqr(rt_im[i]));
    if rt_im[i]<0. then
      rt_im[i]:=-sqrt(abs((r-rt_re[i])/2))
    else
      rt_im[i]:=+sqrt(abs((r-rt_re[i])/2));
    rt_re[i]:=sqrt(abs((r+rt_re[i])/2));
    g_re[i]:=0.; g_im[i]:=0.;
  end;
  for i:=1 to n do
  begin
    for j:=i downto 1 do
    begin
      x:=g_re[j-1]+rt_re[i]*g_re[j]-rt_im[i]*g_im[j];
      y:=g_im[j-1]+rt_re[i]*g_im[j]+rt_im[i]*g_re[j];
      g_re[j]:=x; g_im[j]:=y;
    end;
    x:=rt_re[i]*g_re[0]-rt_im[i]*g_im[0];
    y:=rt_re[i]*g_im[0]+rt_im[i]*g_re[0];
    g_re[0]:=x; g_im[0]:=y;
  end;
  if (ladder=0) and ((h[n]/g_re[n])>0) then
    for i:=0 to n do g_re[i]:=-1*g_re[i];
  if (ladder=1) and ((h[0]/g_re[0])<0) then
    for i:=0 to n do  g_re[i]:=-1*g_re[i];
end;{Procedure gcal_dist}

procedure w;
begin
  {to separate}
end;

overlay procedure yf_cal (freq : real; fnode,tnode,el_type : intarr;
                  init_el,init_ang,ffreq : vector; num_el : integer;
                  var y11,y12,y21,y22 : real);

var          admit     : matrix;
             nmax,i,j,k : integer;
         te,si,co,coo   : real;
```

```
begin
  nmax:=0;
  for i:=1 to num_el do
  begin
    if fnode[i]>nmax then nmax:=fnode[i];
    if tnode[i]>nmax then nmax:=tnode[i];
  end;
  if nmax<3 then nmax:=3;
  for i:=1 to nmax do
  begin
    for j:=1 to nmax do
    admit[i,j]:=0;
  end;
  for i:=1 to num_el do
  begin
    if (el_type[i]<>0) AND (el_type[i]<>1) then
    begin
      te:=init_ang[i]*freq/ffreq[i];
      if (te=0) or (te=PI) or (te=2*PI) then si:=1E-14
      else si:=sin(te);
      if (te=PI) or (te=3*PI)  then co:=1E-14
      else co:=cos(te/2);
      if(te=PI/2) or (te=3*PI/2) then coo:=1E-14
      else coo:=cos(te);
    end;
    if el_type[i]=1 then
    begin
      admit[fnode[i],fnode[i]]:=admit[fnode[i],fnode[i]]-1/(freq*init_el[i]);
      admit[tnode[i],tnode[i]]:=admit[tnode[i],tnode[i]]-1/(freq*init_el[i]);
      admit[fnode[i],tnode[i]]:=admit[fnode[i],tnode[i]]+1/(freq*init_el[i]);
      admit[tnode[i],fnode[i]]:=admit[fnode[i],tnode[i]];
    end;
    if el_type[i]=0 then
    begin
      admit[fnode[i],fnode[i]]:=admit[fnode[i],fnode[i]]+freq*init_el[i];
      admit[tnode[i],tnode[i]]:=admit[tnode[i],tnode[i]]+freq*init_el[i];
      admit[fnode[i],tnode[i]]:=admit[fnode[i],tnode[i]]-freq*init_el[i];
      admit[tnode[i],fnode[i]]:=admit[fnode[i],tnode[i]];
    end;
    if el_type[i]=3 then
    begin
      admit[fnode[i],fnode[i]]:=admit[fnode[i],fnode[i]]-coo/(init_el[i]*si);
      admit[tnode[i],tnode[i]]:=admit[tnode[i],tnode[i]]-coo/(init_el[i]*si);
      admit[fnode[i],tnode[i]]:=admit[fnode[i],tnode[i]]+1/(init_el[i]*si);
      admit[tnode[i],fnode[i]]:=admit[fnode[i],tnode[i]];
      admit[fnode[i],3]:=admit[fnode[i],3]-sin(te/2)/(init_el[i]*co);
      admit[3,fnode[i]]:=admit[fnode[i],3];
      admit[tnode[i],3]:=admit[tnode[i],3]-sin(te/2)/(init_el[i]*co);
      admit[3,tnode[i]]:=admit[tnode[i],3];
      admit[3,3]:=admit[3,3]+2*sin(te/2)/(init_el[i]*co);
    end;
    if el_type[i]=4 then
```

```
    begin
      admit[fnode[i],fnode[i]]:=admit[fnode[i],fnode[i]]-coo/(init_el[i]*si);
      admit[tnode[i],tnode[i]]:=admit[tnode[i],tnode[i]]-coo/(init_el[i]*si);
      admit[fnode[i],tnode[i]]:=admit[fnode[i],tnode[i]]+coo/(init_el[i]*si);
      admit[tnode[i],fnode[i]]:=admit[fnode[i],tnode[i]];
    end;
    if el_type[i]=5 then
    begin
      admit[fnode[i],fnode[i]]:=admit[fnode[i],fnode[i]]+si/(init_el[i]*coo);
      admit[tnode[i],tnode[i]]:=admit[tnode[i],tnode[i]]+si/(init_el[i]*coo);
      admit[fnode[i],tnode[i]]:=admit[fnode[i],tnode[i]]-si/(init_el[i]*coo);
      admit[tnode[i],fnode[i]]:=admit[fnode[i],tnode[i]];
    end;
  end;
  for i:=nmax downto 4 do
  begin
    for j:=1 to i-1 do
    begin
      for k:=1 to i-1 do
      admit[j,k]:=admit[j,k]-admit[j,i]*admit[i,k]/admit[i,i];
    end;
  end;
  y11:=admit[1,1]; y12:=admit[1,2];
  y21:=admit[2,1]; y22:=admit[2,2];
end;{procedure yf_cal}

procedure gauss(var a:matrix; var b:vector; n:integer; var solution:boolean);

var       i,imax,j,k : integer;
            max,temp : real;

begin
  k:=1; solution:=true;
  while solution and (k<=n) do
  begin
    max:=0.;
    for i:=k to n do
      if max<abs(a[i,k]) then
      begin
        max:=abs(a[i,k]); imax:=i;
      end;
   if max>0. then
   begin
     if imax<>k then
     begin
       for j:=k to n do
       begin
         temp:=a[imax,j]; a[imax,j]:=a[k,j]; a[k,j]:=temp;
       end;
       temp:=b[imax]; b[imax]:=b[k]; b[k]:=temp;
     end;
     for i:=1 to n do
       if i<>k then
```

```
      begin
        temp:=a[i,k]/a[k,k];
        for j:=k to n do
          a[i,j]:=a[i,j]-temp*a[k,j];
        b[i]:=b[i]-temp*b[k];
      end;
   end
   else
     solution:=false;
  k:=k+1;
end;
if solution then
  for i:=1 to n do
    b[i]:=b[i]/a[i,i];
end;{Procedure Gauss}

overlay procedure err(freq,sl_re,sl_im,sg_re,sg_im,gopt,x: vector;
                      var f,tt: vector;nopt,m,ladder,nd,k: integer;
                      var_zero,multip:intarr;zero:vector;nuvzer,
                      num_zero:integer;zerflg:boolean; g_type:integer);

var                             h,g : polynomial;
                               j,i,n : integer;
          hp_re,hp_im,hn_re,hn_im,
         tzer,g_re,g_im,gmag,gain,
    sqre21,sk_re,sk_im,tg,l21,den2,
   den1_re,den1_im,num1_re,num1_im,
      e11_re,e11_im,e21_re,e21_im,
     e22_re,e22_im,e22p_re,e22p_im : real;


begin
  case ladder OF
    0: begin
         n:=nopt-1;
         for i:=1 to nopt do
           h[i-1]:=x[i];
       end;
   +1: begin
         n:=nopt;
         h[0]:=0.;
         for i:=1 to nopt do
           h[i]:=x[i];
       end;
   -1: begin
         n:=nopt;
         h[n]:=0.;
         for i:=1 to nopt do
           h[i-1]:=x[i];
       end;
  end;
  g_cal(g,h,n,k,var_zero,multip,zero,nuvzer,num_zero,zerflg);
  for i:=1 to m do
```

```
  begin
    tzer:=1;
    evalpoly(h,n,0,+freq[i],hp_re,hp_im);
    evalpoly(h,n,0,-freq[i],hn_re,hn_im);
    evalpoly(g,n,0,+freq[i],g_re,g_im);
    gmag:=sqr(g_re)+sqr(g_im);
    tg:=1-(sqr(sg_re[i])+sqr(sg_im[i])); l21:=1-(sqr(sl_re[i])+sqr(sl_im[i]));
    e11_re:=(hp_re*g_re+hp_im*g_im)/gmag; e11_im:=(hp_im*g_re-hp_re*g_im)/gmag;
    sk_re:=power(freq[i],k)*COS(k*PI/2); sk_im:=power(freq[i],k)*SIN(k*PI/2);
    if zerflg=true then for j:=1 to num_zero do
    tzer:=tzer*power(sqr(zero[j])-sqr(freq[i]),multip[j]);
    e21_re:=(sk_re*g_re+sk_im*g_im)/gmag*tzer;
    e21_im:=(sk_im*g_re-sk_re*g_im)/gmag*tzer;
    sqre21:=sqr(e21_re)+sqr(e21_im);
    e22_re:=-mip(k)*(hn_re*g_re+hn_im*g_im)/gmag; e22_im:=-mip(k)*(hn_im*g_re-hn_re*g_im)/gmag;
    num1_re:=(sqr(e21_re)-sqr(e21_im))*sg_re[i]-2*e21_re*e21_im*sg_im[i];
    num1_im:=(sqr(e21_re)-sqr(e21_im))*sg_im[i]+2*e21_re*e21_im*sg_re[i];
    den1_re:=1-(e11_re*sg_re[i]-e11_im*sg_im[i]); den1_im:=-(e11_re*sg_im[i]+e11_im*sg_re[i]);
    e22p_re:=e22_re+(num1_re*den1_re+num1_im*den1_im)/(sqr(den1_re)+sqr(den1_im));
    e22p_im:=e22_im+(num1_im*den1_re-num1_re*den1_im)/(sqr(den1_re)+sqr(den1_im));
    den2:=sqr(1-(e22p_re*sl_re[i]-e22p_im*sl_im[i]))+sqr(-(e22p_re*sl_im[i]+e22p_im*sl_re[i]));
    gain:=tg*l21*sqre21/((sqr(den1_re)+sqr(den1_im))*den2);tt[i]:=gain;
    if g_type=1 then gain:=10*ln(gain)/ln(10);
    f[i]:=gopt[i]-gain;
  end;
end;{Function err}

overlay procedure errdist(freq,sl_re,sl_im,sg_re,sg_im,gopt,x: vector;
                          var f,tt: vector;nopt,m,ladder,nd,q: integer;var_zero,
                          multip:intarr;zero:vector;nuvzer,num_zero:integer;
                          zerflg:boolean; g_type:integer);

var                             h,g : polynomial;
                              j,i,n : integer;
       xx,hp_re,hp_im,hn_re,hn_im,
         tzer,g_re,g_im,gmag,gain,
    sqre21,sk_re,sk_im,tg,l21,den2,
   den1_re,den1_im,num1_re,num1_im,
      e11_re,e11_im,e21_re,e21_im,
     e22_re,e22_im,e22p_re,e22p_im : real;


begin
  case ladder OF
   +1: begin
         n:=nopt;
         h[0]:=0.;
         for i:=1 to nopt do
           h[i]:=x[i];
       end;
    0: begin
         n:=nopt-1;
         for i:=1 to nopt do
         h[i-1]:=x[i];
       end;
```

```
  end;
  gcal_dist(g,h,n,q,ladder,nd,var_zero,multip,zero,nuvzer,num_zero,zerflg);
  for i:=1 to m do
  begin
    tzer:=1;
    evalpoly(h,n,0,+freq[i],hp_re,hp_im);
    evalpoly(h,n,0,-freq[i],hn_re,hn_im);
    evalpoly(g,n,0,+freq[i],g_re,g_im);
    gmag:=sqr(g_re)+sqr(g_im);
    tg:=1-(sqr(sg_re[i])+sqr(sg_im[i])); l21:=1-(sqr(sl_re[i])+sqr(sl_im[i]));
    e11_re:=(hp_re*g_re+hp_im*g_im)/gmag; e11_im:=(hp_im*g_re-hp_re*g_im)/gmag;
    e22_re:=-(hn_re*g_re+hn_im*g_im)/gmag;
    e22_im:=-(hn_im*g_re-hn_re*g_im)/gmag;
    if zerflg=true then for j:=1 to num_zero do
    tzer:=tzer*power(sqr(zero[j])-sqr(freq[i]),multip[j]);
    xx:=power((1+sqr(freq[i])),nd); xx:=sqrt(xx);
    e21_re:=xx*tzer*g_re/gmag; e21_im:=-xx*tzer*g_im/gmag;
    sqre21:=sqr(e21_re)+sqr(e21_im);
    num1_re:=(sqr(e21_re)-sqr(e21_im))*sg_re[i]-2*e21_re*e21_im*sg_im[i];
    num1_im:=(sqr(e21_re)-sqr(e21_im))*sg_im[i]+2*e21_re*e21_im*sg_re[i];
    den1_re:=1-(e11_re*sg_re[i]-e11_im*sg_im[i]); den1_im:=-(e11_re*sg_im[i]+e11_im*sg_re[i]);
    e22p_re:=e22_re+(num1_re*den1_re+num1_im*den1_im)/(sqr(den1_re)+sqr(den1_im));
    e22p_im:=e22_im+(num1_im*den1_re-num1_re*den1_im)/(sqr(den1_re)+sqr(den1_im));
    den2:=sqr(1-(e22p_re*sl_re[i]-e22p_im*sl_im[i]))+sqr(-(e22p_re*sl_im[i]+e22p_im*sl_re[i]));
    gain:=tg*l21*sqre21/((sqr(den1_re)+sqr(den1_im))*den2);
    tt[i]:=gain;
    if g_type=1 then gain:=10*ln(gain)/ln(10);
    f[i]:=gopt[i]-gain;
  end;
end;{Function errdist}

overlay procedure err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt : vector;
                           m : integer;fnode,tnode,el_type,vari,vari2:intarr;
                           l_limit,u_limit,llang,ulang,ffreq : vector ;
                           var init_el,init_ang : vector;num_el : integer;
                           var f,fl : vector; g_type : integer;var tt:vector);

var          y11,y12,y21,y22,
             den1,ys_re,ys_im,
             den2,den_re,den_im,zq_re,
             zq_im,t,den3_re,den3_im              : real;
                                        k,i,j : integer;

begin
  for i:=1 to m do
  begin
    ys_re:=zg_re[i]/(sqr(zg_re[i])+sqr(zg_im[i]));
    ys_im:=-zg_im[i]/(sqr(zg_re[i])+sqr(zg_im[i]));
    yf_cal(freq[i],fnode,tnode,el_type,init_el,init_ang,ffreq,num_el,
          y11,y12,y21,y22);
    den1:=-y11*y22;
    den2:=-y12*y21;
    den3_re:=-y22*ys_im;den3_im:=y22*ys_re;
    den_re:=den1-den2+den3_re; den_im:=den3_im;
```

```
    zq_re:=(ys_re*den_re+den_im*(ys_im+y11))/(sqr(den_re)+sqr(den_im));
    zq_im:=((ys_im+y11)*den_re-den_im*ys_re)/(sqr(den_re)+sqr(den_im));
    t:=abs(4*zl_re[i]*zq_re/(sqr(zl_re[i]+zq_re)+sqr(zl_im[i]+zq_im)));
    tt[i]:=t;
    if g_type=1 then t:=10*ln(t)/ln(10);
    f1[i]:=gopt[i]-t;
    for j:=1 to num_el do
    begin
      if vari1[i]=1 then
        if (init_el[j]<l_limit[j])  or (init_el[j]>u_limit[j]) then t:=t+1;
      if vari2[i]=1 then
        if (init_ang[j]<llang[j])  or (init_ang[j]>ulang[j]) then t:=t+1;
    end;
    f[i]:=gopt[i]-t;
  end;
end;{procedure err_elmt}


procedure separate2;
begin
{to separate2 overlays}
end;

overlay procedure optim(freq,sl_re,sl_im,sg_re,sg_im,gopt : vector;
                        var x: vector;ladder,kq,nd,n,y,m,dist: integer;
                        var tolerance,zref,fmax,f_center: real;
                        var_zero,multip : intarr; var zero : vector;
                        nuvzer,num_zero : integer; zerflg : boolean;
                        g_type:integer);

var                        jac,jactjac : matrix;
      jactf,h1,func,funcp,hh,func1,tt : vector;
                        g,h,znm,zdn,el :polynomial;
                                 stab : array[1..3] of real;
              gg,gm,max1,max,temp,u,c : real;
                           np,nn,i,j,k : integer;
                 solution,lindep,rett : boolean;
                                  opt : char;

procedure change_stab;
begin
  window(50,2,80,6);gotoxy(1,1);
  writeln('Enter Stability factors for:');
  write('   .6 < Error     :');readln(stab[1]);
  write('   .1 < Error <.6 :');readln(stab[2]);
  write('        Error <.1 :');readln(stab[3]);
  window(1,21,50,25);
end;

procedure change_tol;
var      temp : real;
begin
  window(50,7,80,8);gotoxy(1,1);
```

```
  write('Enter tolerance:');readln(temp);
  if (temp<=tolerance) and (solution) then solution:=false;
  tolerance:=temp;window(1,21,80,25);
end;

procedure change_h;
var  i,j : integer;
     max : real;
begin
  window(50,9,80,15);gotoxy(1,1);
  for i:=1 to n do
  begin
    j:=i+round((ladder-1.2)/2);
    write('   h(',j:2,')=');readln(x[i]);
    solution:=false;
  end;
  window(1,21,80,25);
end;

procedure disp_h;
var  i,j : integer;
begin
  window(50,16,80,25);gotoxy(1,1);
  for i:=1 to n do
  begin
    j:=i+round((ladder-1.2)/2);
    writeln('   h(',j:2,')=',x[i]);
  end;
  window(1,21,80,25);
end;

procedure list_gain;
var              xp,yp : integer;
                 max,p : real;
                 fil   : text;
              filename : string[20];
                   ok : boolean;
begin
  window(1,1,80,20); clrscr; normvideo;
  for i:=1 to m do
  begin
    tt[i]:=1-tt[i];
    tt[i]:=-10*ln(tt[i])/ln(10);
    writeln('return loss at freq   ',freq[i]:10,'   is    ',tt[i]:10);
  end;
  write('filename :');readln(filename);
  assign(fil,filename);{$I-} reset(fil);{$I+}
  ok:=(ioresult=0);
  if ok then
  begin
  rewrite(fil);
  for i:=1 to m do writeln(fil,freq[i],tt[i]);
  end;
```

```
  close(fil);
  window(1,21,80,25);
end;{procedure list_gain}

procedure disp_elmts;
var                                      ord : string[20];
                                          ch : char;
    imp,imp1,imp2,imp3,z_sss,z_sos,z_ocs,z_ch : real;
                                       i,j,yy : integer;
                                    znm1,zdn1 : polynomial;
                                        flag :boolean;

procedure cfrac(var a,b,s:polynomial;n:integer;var flag:boolean);

var            temp : real;
                i,o : integer;
       realization : boolean;
begin
  flag:=true;
  if abs(a[n])<1.E-7 then
  begin
    flag:=false;
    for i:=0 to n do
    begin
      temp:=a[i]; a[i]:=b[i]; b[i]:=temp;
    end;
  end;
  o:=n; realization:=true;
  while (o>0) and realization do
  begin
    if abs(b[o])>1.E-7 then
      realization:=false
    else
    begin
      s[o]:=a[o]/b[o-1];
      for i:=o downto 1 do
      begin
        temp:=a[i]-s[o]*b[i-1]; a[i]:=b[i]; b[i]:=temp;
      end;
      temp:=a[0]; a[0]:=b[0]; b[0]:=temp;
    end;
    o:=o-1;
  end;
  if realization then
    s[0]:=a[0]/b[0]
  else
    writeln('No ladder realization');
end; {Procedure cfrac}

procedure evaluate(p:polynomial;n:integer;z_re,z_im:real;var p_re,p_im:real);

var          i : integer;
    p_are,p_aim : real;
```

```
begin
  p_re:=0.; p_im:=0.;
  for i:=n downto 0 do
  begin
    p_are:=p_re*z_re-p_im*z_im; p_aim:=p_re*z_im+p_im*z_re;
    p_re:=p_are+p[i]; p_im:=p_aim;
  end;
end; {Procedure evaluate}

procedure pretty
(x : real; unit : char);
var          expo : integer;
           expstr : string[9];
begin
  expstr:='pnfm kMGT';expo:=0;
  if x<>0. then
  begin
    while (abs(x))>=1000.0) and (expo<4) do begin x:=x/1000.;expo:=expo+1; end;
    while (abs(x)<1.0) and (expo>-4) do begin x:=x*1000.;expo:=expo-1; end;
  end;
  writeln(x:8:2,expstr[expo+5],unit);
end;{procedure pretty}

procedure richard(var z:polynomial; n:integer);
var      temp : polynomial;
            i : integer;
begin
  for i:=n downto 0 do temp[i]:=0.;
  for i:=n downto 2 do
  begin
    temp[i-2]:=z[i];z[i-2]:=z[i-2]+z[i];z[i]:=0;
  end;
  for i:=n-2 downto 0 do z[i]:=temp[i];
end;{procedure richard}

procedure sos_extract(var znm,zdn : polynomial; var z_sos : real; n:integer);
var           zn,zd : polynomial;
                  i : integer;
begin
  for i:=0 to n do
  begin
    zn[i]:=znm[i]; zd[i]:=zdn[i];
  end;
  z_sos:=zn[0]/zd[1];
  for i:=1 to n-1 do
  begin
    zn[i]:=zn[i+1]-zn[i+2]*z_sos;
    zd[i]:=zd[i+1];
  end;
  for i:=0 to n do
  begin
    znm[i]:=zn[i]; zdn[i]:=zd[i];
  end;
end;{procedure sos_extract}
```

```
procedure sss_extract(var znm,zdn : polynomial; var z_sss : real; n:integer);
var          zn,zd : polynomial;
                 i : integer;
begin
  for i:=0 to n do
  begin
    zd[i]:=znm[i]; zn[i]:=zdn[i];
  end;
  z_sss:=zn[0]/zd[1];
  for i:=1 to n-1 do
  begin
    zn[i]:=zn[i+1]-zn[i+2]*z_sss;
    zd[i]:=zd[i+1];
  end;
  for i:=0 to n do
  begin
    znm[i]:=zd[i]; zdn[i]:=zn[i];
  end;
end;{procedure sss_extract}

procedure open_extract(var znm,zdn : polynomial; var z_ocs : real; n:integer);
var          zn,zd : polynomial;
                 i : integer;
              temp : real;
begin
  for i:=0 to n do
  begin
    zn[i]:=znm[i];zd[i]:=zdn[i];
  end;
  z_ocs:=zd[n]/zn[n-1];zd[n]:=0;
  for i:=1 to n-1 do zd[n-i]:=zd[n-i]-zn[n-1-i]*z_ocs;
  for i:=0 to n do
  begin
    znm[i]:=zn[i];zdn[i]:=zd[i];
  end;
end;{procedure open_extract}

begin
  if dist= 0 then
  begin
  case ladder of
    0: begin
         np:=n-1;
         for i:=1 to np do h[i-1]:=x[i];
       end;
   +1: begin
         h[0]:=0.; for i:=1 to n do h[i]:=x[i];
       end;
   -1: begin
         h[y]:=0.; for i:=1 to n do h[i-1]:=x[i];
       end;
  end;
  g_cal(g,h,y,kq,var_zero,multip,zero,nuvzer,num_zero,zerflg);
```

```
window(1,1,49,20);clrscr;normvideo;gotoxy(1,1);
case ladder of
  0:begin
      writeln('No ladder realization');
      writeln('Z_in(s)=a(s)/b(s);');
      for i:=0 to y do writeln('a[',i:2,']=',(g[i]+h[i]):10);
      for i:=0 to y do writeln('b[',i:2,']=',(g[i]-h[i]):10);
      writeln('Z-ref=',zref:5:1);
      writeln('ref.freq.=',fmax:10);
    end;
 +1:begin
      writeln('Low_pass ladder ...');
      for i:=0 to y do
      begin
        znm[i]:=h[i]+g[i];zdn[i]:=g[i]-h[i];
      end;
      if zerflg=true then
      begin
        writeln('with finite-jw transmission zeros');
        for i:=0 to y do writeln('a[',i:2,']=',(g[i]+h[i]):10);
        for i:=0 to y do writeln('b[',i:2,']=',(g[i]-h[i]):10);
        writeln('Z-ref=',zref:5:1);
        writeln('ref.freq.=',fmax:10);
      end
      else cfrac(znm,zdn,el,y,flag);
    end;
 -1:begin
      writeln('High_pass ladder ...');
      for i:=0 to y do
      begin
        znm[i]:=h[y-i]+g[y-i];zdn[i]:=g[y-i]-h[y-i];
      end;
      if zerflg=true then
      begin
        writeln('with finite-jw transmission zeros');
        for i:=0 to y do writeln('a[',i:2,']=',(g[i]+h[i]):10);
        for i:=0 to y do writeln('b[',i:2,']=',(g[i]-h[i]):10);
        writeln('Z-ref=',zref:5:1);
        writeln('ref.freq.=',fmax:10);
      end
      else
      begin
        cfrac(zdn,znm,el,y,flag);
        for i:=0 to y do el[i]:=1/el[i];
      end;
    end;
end;
if (ladder<>0) AND (zerflg=false) then
begin
  writeln('Ladder elements (from gen.)');
  for i:=y downto 1 do
  begin
    if flag then
```

```
      begin
        write('L',(y+1-i):2,'=');pretty(el[i]*zref/(2*pi*fmax),'H');
      end
      else
      begin
        write('C',(y+1-i):2,'=');pretty(el[i]/(zref*2*pi*fmax),'F');
      end;
      flag:=not flag;
    end;
  end;
end;
if dist=1 then
begin
case ladder of
  +1: begin
        h[0]:=0.;for i:=1 to n do h[i]:=x[i];
      end;
   0: begin
      for i:=1 to n do h[i-1]:=x[i];
      end;
    end;
gcal_dist(g,h,y,kq,ladder,nd,var_zero,multip,zero,nuvzer,num_zero,zerflg);
case ladder of
  +1: begin
        for i:=0 to y do
        begin
          znm[i]:=g[i]+h[i];
          zdn[i]:=g[i]-h[i];
          if zerflg=true then
          begin
            writeln('Low-pass design ...');
            writeln('with finite-jw transmission zeros');
            for i:=0 to y do writeln('a[',i:2,']=',(g[i]+h[i]):10);
            for i:=0 to y do writeln('b[',i:2,']=',(g[i]-h[i]):10);
            writeln('Z-ref=',zref:5:1);
            writeln('quarter wave freq.=',f_center:10);
          end;
        end;
      end;
   0: begin
        for i:=0 to y do
        begin
          znm[i]:=g[y-i]+h[y-i];
          zdn[i]:=g[y-i]-h[y-i];
          if zerflg=true then
          begin
            writeln('Band-pass design ...');
            writeln('with finite-jw transmission zeros');
            for i:=0 to y do writeln('a[',i:2,']=',znm[i]:10);
            for i:=0 to y do writeln('b[',i:2,']=',zdn[i]:10);
            writeln('Z-ref=',zref:5:1);
            writeln('quarter wave freq.=',f_center:10);
          end;
```

```
      end;
     end;
   end;
if zerflg=false then
begin
window(1,1,80,19);clrscr;gotoxy(1,1);
writeln(' ENTER Element Extraction Order ...');
writeln(' Write The Element Types In The Order You');
writeln(' Want The Extraction To Be Made.');
writeln(' Use 3 for UNIT ELEMENT ');
writeln('     4 for SHUNT OPEN-CIRCUITED STUB');
writeln('     5 for SHUNT SHORT-CIRCUITED STUB');
writeln('     6 for SERIES OPEN-CIRCUITED STUB');
writeln(' DO NOT USE any other characters');
writeln('*****************************************');
write(' ORDER= ');readln(ord);clrscr;yy:=y;
for i:=1 to y  do
begin
  ch:=copy(ord,i,1);
  case ch of
  '3': begin
        evaluate(znm,yy,1,0,imp,imp1);
        evaluate(zdn,yy,1,0,imp2,imp3);
        z_ch:=zref*imp/imp2;yy:=yy+1;
        znm1[0]:=znm[0];znm1[yy]:=-1*zdn[yy-1]*imp/imp2;
        for j:=yy-1 downto 1 do znm1[j]:=znm[j]-zdn[j-1]*imp/imp2;
        zdn1[0]:=zdn[0]*imp/imp2;zdn1[yy]:=-1*znm[yy-1];
        for j:=yy-1 downto 1 do zdn1[j]:=zdn[j]*imp/imp2-znm[j-1];
        for j:=0 to yy do
        begin
          znm[j]:=znm1[j]*imp/imp2;zdn[j]:=zdn1[j];
        end;
        richard(znm,yy);richard(zdn,yy);
        writeln(i:2,'th element; UE of Ch.Impedance',z_ch:10);
        yy:=yy-2;
       end;
  '5': begin
        sss_extract(znm,zdn,z_sss,yy);
        yy:=yy-1;
        writeln(i:2,'th element; SCS of Ch.Impedance',zref/z_sss:10);
       end;
  '6': begin
        sos_extract(znm,zdn,z_sos,yy);
        yy:=yy-1;
        writeln(i:2,'th element; OCS of Ch.Impedance',zref*z_sos:10);
       end;
  '4': begin
        open_extract(znm,zdn,z_ocs,yy);
        yy:=yy-1;
        writeln(i:2,'th element; OCS of Ch.Impedance',zref/z_ocs:10);
       end;
    end;
  end;
```

```
 end;
 end;
window(1,21,80,25);
end;{procedure disp_elmts}

procedure unify;
var             i,j : integer;
begin
  j:=1;
  for i:=1 to num_zero do
    if var_zero[i]=1 then begin x[n+j]:=zero[i]; j:=j+1; end;
end;{procedure unify}

procedure resolve;
var             i,j : integer;
begin
  j:=1;
  for i:=1 to num_zero do
    if var_zero[i]=1 then begin zero[i]:=x[n+j]; j:=j+1; end;
end;{procedure resolve}

begin
  gm:=0;
  if g_type=1 then
  for i:=1 to m do
  begin
    gg:=abs(gopt[i]);
    if gg>gm then gm:=gg;
  end;
  solution:=false;stab[1]:=1.0;stab[2]:=5.0;stab[3]:=10.0;
  window(1,21,80,25);clrscr;gotoxy(1,1);rett:=false;
  if (zerflg=true) and (nuvzer>0) then unify;
  writeln('Start of Optimization...');
  writeln('F1 : Change Stab  F2 : Disp Elmts    F3 : Change Tol
          F6 : List-RL  ');
  writeln('F5 : Chg h[i]''s  F7 : Disp h[i]''s  F4 : Return');
  normvideo;
  repeat
    if keypressed then
    begin
      read(kbd,opt);
      if opt=#27 then
      begin
        read(kbd,opt);
        case opt of
          #59:change_stab;
          #64:list_gain;
          #61:change_tol;
          #63:change_h;
          #65:disp_h;
          #60:begin
                if (zerflg=true) and (nuvzer>0) then resolve;
                disp_elmts;
```

```
        end;
      #62:rett:=true;

    end;
  end;
end;
if not solution then
begin
  if dist=1 then errdist(freq,sl_re,sl_im,sg_re,sg_im,gopt,x,func,tt,n,m,
                         ladder,nd,kq,var_zero,multip,zero,nuvzer,num_zero,
                         zerflg,g_type)
else  err(freq,sl_re,sl_im,sg_re,sg_im,gopt,x,func,tt,n,m,ladder,nd,kq,
          var_zero,multip,zero,nuvzer,num_zero,zerflg,g_type);
max:=0.; nn:=n+nuvzer;
for i:=1 to m  do
  if max < abs(func[i]) then max:=abs(func[i]);
gotoxy(10,4);writeln('Error max =',max:8:6);
if max > tolerance then
begin
  c:=stab[2]; if max < 0.1*gm  then c:=stab[3];
  if max > 0.6*gm then c:=stab[1]; u:=c*max;
  for i:=1 to nn do
  begin
    h1[i]:=1.E-3*abs(x[i]);
    if h1[i] < 1.E-9 then h1[i]:=1.E-9;
    if h1[i] > max then h1[i]:=max;
  end;
  for j:=1 to nn do
  begin
    x[j]:=x[j]+h1[j];
    if (zerflg=true) and (nuvzer>0) then resolve;
    if dist=1 then errdist(freq,sl_re,sl_im,sg_re,sg_im,gopt,x,funcp,tt,n,
                           m,ladder,nd,kq,var_zero,multip,zero,nuvzer,
                           num_zero,zerflg,g_type)
    else err(freq,sl_re,sl_im,sg_re,sg_im,gopt,x,funcp,tt,n,m,ladder,nd,kq,
             var_zero,multip,zero,nuvzer,num_zero,zerflg,g_type);
    x[j]:=x[j]-h1[j];
    for i:=1 to m do
      jac[i,j]:=(funcp[i]-func[i])/h1[j];
  end;
  for i:=1 to nn do
  begin
    for j:=1 to nn do
    begin
      temp:=0.;
      for k:=1 to m do
        temp:=temp+jac[k,i]*jac[k,j];
      jactjac[i,j]:=temp;
    end;
    jactjac[i,i]:=jactjac[i,i]+u;
  end;
  for i:=1 to nn do
  begin
```

```
      temp:=0.;
      for j:=1 to m do
        temp:=temp+jac[j,i]*func[j];
      jactf[i]:=temp;
    end;
    gauss(jactjac,jactf,n,lindep);
    for i:=1 to nn do x[i]:=x[i]-jactf[i];
    if (zerflg=true) and (nuvzer>0) then resolve;
  end
  else
  begin
    if (zerflg=true) and (nuvzer>0) then resolve;
    solution:=true;
  end;
 end;
 until rett;
 window(1,1,80,25);clrscr;
end; {Procedure optim}

overlay procedure opt_elmt(fnode,tnode,el_type,vari1,vari2 : intarr;
                  l_limit,u_limit,llang,ulang,ffreq: vector;
                  var init1_el,init1_ang: vector; num_el,num_var: integer;
                  freq,zg_re,zg_im,zl_re,zl_im,gopt : vector; m: integer;
                  var tolerance: real; zref,fmax: real; g_type : integer);

var                              jac,jactjac : matrix;
    funcp2,func1,jactf,h,func,funcp,init_el,tt : vector;
                                        stab : array[1..3] of real;
                  gg,gm,max,max1,temp,u,c : real;
                                  r,i,j,k,p : integer;
                  rett,solution,eq,lindep : boolean;
                                      opt : char;

procedure fibonacci(l_limit,u_limit,llang,ulang : vector; var init1_el,
            init1_ang : vector;num_el,num_var : integer);

var                                  f : array[0..10] of integer;
                           err1,err2 : vector;
        new,old,temp,test1,test2,
        max2,max1,upper,lower,dif1,dif2 : real;
                                i,j,k : integer;
                                 flag : boolean;
label                            B,C;
begin
 f[0]:=1; f[1]:=1;
 for i:=2 to 10 do f[i]:=f[i-1]+f[i-2];
 window(1,1,80,20);clrscr;gotoxy(1,1);
 for i:=1 to num_el do
 begin
  if vari1[i]=1 then init1_el[i]:=l_limit[i]+random*(u_limit[i]-l_limit[i]);
  if vari2[i]=1 then init1_ang[i]:=llang[i]+random*(ulang[i]-llang[i]);
 end;
 for i:=1 to num_el do
```

```
begin
  if vari1[i]=1 then
  begin
    dif1:=u_limit[i]-l_limit[i];
    lower:=l_limit[i];upper:=u_limit[i];
    old:=upper-f[9]/f[10]*dif1;
    init1_el[i]:=old;
    err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt,m,fnode,tnode,el_type,
    vari1,vari2,l_limit,u_limit,llang,ulang,ffreq,init1_el,init1_ang,
    num_el,err1,err2,g_type,tt);
    max1:=0;
    for k:=1 to m do max1:=max1+sqr(err1[k]);max1:=max1/m;
    for j:=9 downto 2 do
    begin
      test1:=lower+f[j]/f[10]*dif1;
      test2:=upper-f[j]/f[10]*dif1;
      if test1<test2 then
      begin
        temp:=test1; test1:=test2; test2:=temp;
      end;
      if old=test1 then
      begin
        new:=test2;
      end
      else
      begin
        new:=test1;
      end;
      if new<old then flag:=true
      else flag:=false;
      init1_el[i]:=new;
B:    err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt,m,fnode,tnode,el_type,
      vari1,vari2,l_limit,u_limit,llang,ulang,ffreq,init1_el,init1_ang,
      num_el,err1,err2,g_type,tt);
      max2:=0;
      for k:=1 to m do max2:=max2+sqr(err1[k]); max2:=max2/m;
      if (max2-max1) = 0.0 then
      begin
        temp:=init1_el[i]/1000; temp:=temp+init1_el[i];
        init1_el[i]:=temp; goto B;
      end;
      if (max2-max1) < 0.0 then
      begin
        if flag=true then
        begin
          upper:=test1; old:=new; max1:=max2;
        end
        else
        begin
          upper:=new; max1:=max2;
        end;
      end;
      if (max2-max1) > 0.0 then
```

```
      begin
        if flag=true then lower:=test2
        else
        begin
          lower:=old; old:=new;
        end;
      end;
    end;
    init1_el[i]:=(lower+upper)/2;
  end;
  if vari2[i]=1 then
  begin
    dif1:=ulang[i]-llang[i];
    lower:=llang[i];upper:=ulang[i];
    old:=upper-f[9]/f[10]*dif1;
    init1_ang[i]:=old;
    err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt,m,fnode,tnode,el_type,vari1,
    vari2,l_limit,u_limit,llang,ulang,ffreq,init1_el,init1_ang,num_el,err1,
    err2,g_type,tt);
    max1:=0;
    for k:=1 to m do max1:=max1+sqr(err1[k]);max1:=max1/m;
    for j:=9 downto 2 do
    begin
      test1:=lower+f[j]/f[10]*dif1;
      test2:=upper-f[j]/f[10]*dif1;
      if test1<test2 then
      begin
        temp:=test1; test1:=test2; test2:=temp;
      end;
      if old=test1 then
      begin
        new:=test2;
      end
      else
      begin
        new:=test1;
      end;
      if new<old then flag:=true
      else flag:=false;
      init1_ang[i]:=new;
C:    err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt,m,fnode,tnode,el_type,
      vari1,vari2,l_limit,u_limit,llang,ulang,ffreq,init1_el,init1_ang,
      num_el,err1,err2,g_type,tt);
      max2:=0;
      for k:=1 to m do max2:=max2+sqr(err1[k]); max2:=max2/m;
      if (max2-max1) = 0.0 then
      begin
        temp:=init1_ang[i]/1000; temp:=temp+init1_ang[i];
        init1_ang[i]:=temp; goto B;
      end;
      if (max2-max1) < 0.0 then
      begin
        if flag=true then
```

```
          begin
            upper:=test1; old:=new; max1:=max2;
          end
          else
          begin
            upper:=new; max1:=max2;
          end;
        end;
        if (max2-max1) > 0.0 then
        begin
          if flag=true then lower:=test2
          else
          begin
            lower:=old; old:=new;
          end;
        end;
      end;
      init1_ang[i]:=(lower+upper)/2;
    end;
  end;
end;{procedure fibonacci}

procedure stab_change;
begin
  window(1,1,80,20);clrscr;normvideo;gotoxy(1,1);
  writeln('Enter Stability factors for:');
  write('   .6 < Error     :');readln(stab[1]);
  write('   .1 < Error <.6 :');readln(stab[2]);
  write('        Error <.1 :');readln(stab[3]);
  window(1,21,80,25);
end;{procedure stab_change}

procedure tol_change;
var           temp : real;
begin
  window(1,1,80,20);clrscr;normvideo;gotoxy(1,1);
  write('Enter tolerance:');readln(temp);
  if (temp<tolerance) then solution:=false;
  tolerance:=temp;
  window(1,21,80,25);
end;{procedure tol_change}

procedure elmt_change;
var      i : integer;
begin
  window(1,1,80,20);clrscr;normvideo;gotoxy(1,1);
  for i:=1 to num_el do
  begin
    writeln('element between nodes',fnode[i]:2,'-',tnode[i]:2);
    if (el_type[i]=0) or (el_type[i]=1) then
    begin
      if vari[i]=1 then
      begin
```

```
        write('Enter lower limit ');readln(l_limit[i]);
        write('Enter upper limit '); readln(u_limit[i]);
      end;
      write('Enter element value '); readln(init1_el[i]);
    end;
    if el_type[i]>1 then
    begin
      if vari1[i]=1 then
      begin
        write('Enter lower impedance limit'); readln(l_limit[i]);
        write('Enter upper impedance limit'); readln(u_limit[i]);
      end;
      write('Enter element impedance'); readln(init1_el[i]);
      if vari2[i]=1 then
      begin
        write('Enter lower angle limit'); readln(llang[i]);
        write('Enter upper angle limit'); readln(ulang[i]);
      end;
      write('Enter electrical angle'); readln(init1_ang[i]);
      write('Enter quarter wave freq.'); readln(ffreq[i]);
    end;
  end;
  window(1,21,80,25);solution:=false;
end;{procedure elmt_change}

procedure elmt_disp;

var  i,j : integer;

procedure pret3(x : real; unit : char);
var          expo : integer;
           expstr : string[9];
begin
  expstr:='pnfm kMGT';expo:=0;
  if x<>0. then
  begin
    while (abs(x)>=1000.0) and (expo<4) do begin x:=x/1000.;expo:=expo+1; end;
    while (abs(x)<1.0) and (expo>-4) do begin x:=x*1000.;expo:=expo-1; end;
  end;
  writeln(x:8:2,expstr[expo+5],unit);
end;{procedure pret3}

begin
  window(1,1,80,20);clrscr;gotoxy(1,1);
  for i:=1 to num_el do
  begin
    if el_type[i]=1 then
    begin
      write('Nodes:',fnode[i]:2,'-',tnode[i]:2,', L[',i:2,']=');
      pret3(init_el[i]*zref/(2*pi*fmax),'H');
    end;
    if el_type[i]=0 then
```

```
    begin
      write('Nodes:',fnode[i]:2,'-',tnode[i]:2,', C[',i:2,']=');
      pret3(init_el[i]/(zref*2*pi*fmax),'F');
    end;
    if el_type[i]=3 then
    begin
      write('Nodes:',fnode[i]:2,'-',tnode[i]:2,',UE of Ch.Impedance=');
      writeln(init1_el[i]:10,' , and El.Angle=',init1_ang[i]:10);
    end;
    if el_type[i]=4 then
    begin
      write('Nodes:',fnode[i]:2,'-',tnode[i]:2,',OCS of Ch.Impedance=');
      writeln(init1_el[i]:10,' , and El.Angle=',init1_ang[i]:10);
    end;
    if el_type[i]=5 then
    begin
      write('Nodes:',fnode[i]:2,'-',tnode[i]:2,',SCS of Ch.Impedance=');
      writeln(init1_el[i]:10,' , and El.Angle=',init1_ang[i]:10);
    end;
  end;
  window(1,21,80,25);
end;{procedure elmt_disp}

procedure list_gain;
var              xp,yp : integer;
                 max,p : real;
                 fil   : text;
              filename : string[20];
                   ok : boolean;
begin
  window(1,1,80,20); clrscr; normvideo;
  for i:=1 to m do
  begin
    tt[i]:=1-tt[i];
    tt[i]:=-10*ln(tt[i])/ln(10);
    writeln('return loss at freq   ',freq[i]:10,'   is    ',tt[i]:10);
  end;
  write('filename :');readln(filename);
  assign(fil,filename);{$I-} reset(fil);{$I+}
  ok:=(ioresult=0);
  if ok then
  begin
  rewrite(fil);
  for i:=1 to m do writeln(fil,freq[i],tt[i]);
  end;
  close(fil);
  window(1,21,80,25);
end;{procedure list_gain}

procedure unify;
begin
  p:=num_var;r:=1;
  for i:=1 to num_var do
```

```
  begin
    if vari1[i]=1 then
    begin
      init_el[p]:=init1_el[i];
      p:=p-1;
    end;
    if vari2[i]=1 then
    begin
      init_el[r]:=init1_ang[i];
      r:=r+1;
    end;
  end;
end;{procedure unify}

procedure resolve;
begin
  p:=num_var;r:=1;
  for i:=1 to num_var do
  begin
    if vari1[i]=1 then
    begin
      init1_el[i]:=init_el[p];
      p:=p-1;
    end;
    if vari2[i]=1 then
    begin
      init1_ang[i]:=init_el[r];
      r:=r+1;
    end;
  end;
end;{procedure resolve}

begin
  gm:=0;
  if g_type=1 then
  for i:=1 to n do
  begin
    gg:=abs(gopt[i]);
    if gg>gm then gm:=gg;
  end;
  unify;rett:=false;
  solution:=false;stab[1]:=1.0;stab[2]:=5.0;stab[3]:=10.0;
  window(1,1,80,25);clrscr;
  window(1,21,80,25);gotoxy(1,1);
  writeln('Start of Optimization...');
  writeln('F1 : Chg Stab    F2 : Ret    F3 : Chg Tol   F8 : Jump   ');
  writeln('F5 : Change elmts  F4 : List-RL   F7 : Display elmts');
  normvideo;
  repeat
    if keypressed then
    begin
      read(kbd,opt);
      if opt=#27 then
```

```
  begin
    read(kbd,opt);
    case opt of
      #59:stab_change;
      #60:rett:=true;
      #61:tol_change;
      #62:list_gain;
      #63:begin
            elmt_change;
            unify;
          end;
      #65:begin
            resolve;
            elmt_disp;
          end;
      #66:begin
            fibonacci(l_limit,u_limit,llang,ulang,initl_el,initl_ang,
                num_el,num_var);
            stab[1]:=20;stab[2]:=20;stab[3]:=20;
          end;
    end;
  end;
end;
if not solution then
begin
  err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt,m,fnode,tnode,el_type,
  vari1,vari2,l_limit,u_limit,llang,ulang,ffreq,initl_el,initl_ang,
  num_el,func,func1,g_type,tt);
  max:=0.;max1:=0;
  for i:=1 to m  do
  begin
    if max1 < abs(func1[i]) then max1:=abs(func1[i]);
    if max < abs(func[i]) then max:=abs(func[i]);
  end;
  gotoxy(1,4);writeln('Error max =',max1:8:6);
  if max1 > tolerance then
  begin
    c:=stab[2]; if max < 0.1*gm  then c:=stab[3];
    if max > 0.6*gm then c:=stab[1]; u:=c*max;
    for i:=1 to num_var do
    begin
      h[i]:=1.E-3*abs(init_el[i]);
      if h[i] < 1.E-9 then h[i]:=1.E-9;
      if h[i] > max then h[i]:=max;
    end;
    for j:=1 to num_var do
    begin
      init_el[j]:=init_el[j]+h[j];resolve;
      err_elmt(freq,zg_re,zg_im,zl_re,zl_im,gopt,m,fnode,tnode,el_type,
      vari1,vari2,l_limit,u_limit,llang,ulang,ffreq,initl_el,initl_ang,
      num_el,funcp,funcp2,g_type,tt);
      init_el[j]:=init_el[j]-h[j];
      for i:=1 to m do
```

```
          jac[i,j]:=(funcp[i]-func[i])/h[j];
        end;
        for i:=1 to num_var do
        begin
          for j:=1 to num_var do
          begin
            temp:=0.;
            for k:=1 to m do
            temp:=temp+jac[k,i]*jac[k,j];
            jactjac[i,j]:=temp;
          end;
          jactjac[i,i]:=jactjac[i,i]+u;
        end;
        for i:=1 to num_var do
        begin
          temp:=0.;
          for j:=1 to m do
            temp:=temp+jac[j,i]*func[j];
          jactf[i]:=temp;
        end;
        gauss(jactjac,jactf,num_var,lindep);
        for i:=1 to num_var do
          init_el[i]:=init_el[i]-jactf[i];resolve;
      end
      else
      begin
        solution:=true;resolve;
      end;
  end;
  until rett;
end; {Procedure opt_elmt}

procedure xxx;
begin
{ to separate overlays}
end;

overlay procedure elmt;
var  freq,sl_re,sl_im,sg_re,sg_im,gopt,
     zl_re,zl_im,zg_re,zg_im,l_limit,u_limit,init_el,
                        llang,ulang,init_ang,ffreq  : vector;
                 fnode,tnode,el_type ,vari1,vari2  : intarr;
                             tolerance,zref,fmax : real;
                       g_type,num_var,i,m,num_el  : integer;

function stoz_re(s_re,s_im:real):real;
begin
  if s_re=1. then stoz_re:=100.E+12
  else stoz_re:=(1-(sqr(s_re)+sqr(s_im)))/(sqr(1-s_re)+sqr(s_im));
end;
function stoz_im(s_re,s_im:real):real;
begin
  if s_re=1. then stoz_im:=0.
```

```
  else stoz_im:=2*s_im/((sqr(1-s_re)+sqr(s_im)));
end;

procedure pret2(x : real; unit : char);
var         expo : integer;
          expstr : string[9];
begin
  expstr:='pnfm kMGT';expo:=0;
  if x<>0. then
  begin
    while (abs(x)>=1000.0) and (expo<4) do begin x:=x/1000.;expo:=expo+1; end;
    while (abs(x)<1.0) and (expo>-4) do begin x:=x*1000.;expo:=expo-1; end;
  end;
  writeln(x:8:2,expstr[expo+5],unit);
end;{procedure pret2}

begin
  window(1,1,80,25);clrscr;normvideo;
  entry1(freq,sl_re,sl_im,sg_re,sg_im,gopt,zref,m,g_type);
  fmax:=freq[m];
  for i:=1 to m do
  begin
    freq[i]:=freq[i]/fmax;
    zg_re[i]:=stoz_re(sg_re[i],sg_im[i]);
    zg_im[i]:=stoz_im(sg_re[i],sg_im[i]);
    zl_re[i]:=stoz_re(sl_re[i],sl_im[i]);
    zl_im[i]:=stoz_im(sl_re[i],sl_im[i]);
  end;
  window(1,1,80,25);clrscr;normvideo;gotoxy(1,1);
  write('Enter Tolerance From Optimum Gain  ');readln(tolerance);
  clrscr;enter(fnode,tnode,el_type,vari1,vari2,fmax,zref,
               l_limit,u_limit,init_el,llang,ulang,init_ang,ffreq,num_el,
               num_var);
  opt_elmt(fnode,tnode,el_type,vari1,vari2,l_limit,u_limit,llang,
           ulang,ffreq,init_el,init_ang,num_el,num_var,
           freq, zg_re,zg_im,zl_re,zl_im,gopt,m,tolerance,zref,fmax,g_type);
  window(1,1,80,25);clrscr;normvideo;gotoxy(1,1);
  window(1,1,80,25); clrscr;
  window(1,21,80,25);
  gotoxy(6,1);writeln('F1 : Optimize Gain');
  gotoxy(53,1);writeln('F2 : Optimize Elements');
  gotoxy(53,2);writeln('F5 : Stop');
end;{procedure elmt}

overlay procedure dist_gain;
 var                   gopt, h_init,freq,sl_re,
                         zero,sl_im,sg_re,sg_im : vector;
                                h,g,znm,zdn,el : polynomial;
           temp,fmax,f_center,td,tolerance,zref : real;
                 nd,q,dist,i,j,m,n,nopt,ladder,
                  g_type,total,nuvzer,num_zero : integer;
                             multip,var_zero : intarr;
                                 zerflg,flag : boolean;
                                         opt : char;
```

```
begin
  dist:=1;
  entry1(freq,sl_re,sl_im,sg_re,sg_im,gopt,zref,m,g_type);fmax:=freq[m];
  window(1,1,80,25);clrscr;normvideo;gotoxy(1,1);
  write('Design type(b');lowvideo;write('p');
  normvideo;write('/l');lowvideo;write('p');normvideo;write(')');
  read(kbd,opt);writeln;normvideo;
  if opt in['b','B'] then
  begin
    writeln ('Band_pass ...'); ladder:=0;
    write('Center of pass_band:'); readln(f_center);
    write ('No. of UEs:'); readln (nd);
    write ('No. of SC_Stubs:'); readln (q);
    n:=nd+q ; nopt:=n+1 ;
  end
  else
  begin
    writeln('Low_pass ...');  ladder:=+1;
    write('Center of Stop_band:'); readln (f_center);
    write('No. of UEs:') ; readln (nd);
    write('No. of OC_Stubs:') ; readln (q);
    n:=nd+q ; nopt:=n ;
  end;
  clrscr; gotoxy(1,1); zerflg:=false; nuvzer:=0; num_zero:=0;
  write(' Design with finite-jw transmission zeros (y/n,Def:n)');
  read(kbd,opt); writeln;
  if opt in ['y','Y'] then
  begin
    zerflg:=true;
    write('Enter number of distinct zeros = '); readln(num_zero);
    for i:=1 to num_zero do
    begin
      clrscr; gotoxy(1,1);
      writeln(' Transmission zero ',i:2);
      write('Frequency = '); readln(zero[i]);
      write('Multiplicity ='); readln(multip[i]);
      write('To be optimized (y/n,Def:y)'); read(kbd,opt); writeln;
      if opt in ['n','N'] then var_zero[i]:=0
      else begin var_zero[i]:=1; nuvzer:=nuvzer+1; end;
      zero[i]:=zero[i]/fmax;
    end;
    total:=0; for i:=1 to num_zero do total:=total+multip[i];
    nopt:=nopt+2*total; clrscr; gotoxy(1,1);
  end;
  writeln('Initial guess of h(i)''s  ...');writeln('Enter');
  for i:=1 to nopt do
  begin
    j:=i+round((ladder-1.2)/2);
    write('   h(',j:2,')=');readln(h_init[i]);
  end;
  write('      tolerance value ');readln(tolerance);
  td:=1/(4*f_center);
  writeln('f_center=',f_center);delay(5000);
```

```
case ladder of
  +1: begin
        for i:=1 to m do
        begin
          temp:=PI*freq[i]/(2*f_center);
          if (abs(temp-PI/2)<1E-6) or (abs(temp-3*PI/2)<1E-6) then
            freq[i]:=100
          else freq[i]:=sin(temp)/cos(temp);
        end;
      end;
   0: begin
        for i:=1 to m do
        begin
          temp:=PI*freq[i]/(2*f_center);
          if (temp=0) OR
          (abs(temp-PI)<1E-6) OR (abs(temp-2*PI)<1E-6) then freq[i]:=100
          else freq[i]:=-cos (temp)/sin(temp);
        end;
      end;
 end;
clrscr;lowvideo;
optim(freq,sl_re,sl_im,sg_re,sg_im,gopt,h_init,ladder,q,nd,nopt,n,m,dist,
     tolerance,zref,fmax,f_center,var_zero,multip,zero,nuvzer,num_zero,
     zerflg,g_type);
window(50,2,80,25);clrscr;
gotoxy(1,1);
case ladder of
 +1: begin
       h[0]:=0.; for i:=1 to nopt do h[i]:=h_init[i];
     end;
  0: begin
       for i:=1 to nopt do h[i-1]:=h_init[i];
     end;
end;
gcal_dist(g,h,n,q,ladder,nd,var_zero,multip,zero,nuvzer,num_zero,zerflg);
window(1,1,50,20);clrscr;normvideo;
case ladder of
 +1:begin
     writeln('Low_pass design ...');
     writeln('Z_in(S)=a(S)/b(S);');
     for i:=0 to n do
     begin
     write('g[',i:2,']=',g[i]:10);
     writeln('a[',i:2,']=',(g[i]+h[i]):10);
     end;
     for i:=0 to n do
     begin
     write('h[',i:2,']=',h[i]:10);
     writeln('b[',i:2,']=',(g[i]-h[i]):10);
     end;
     writeln('Z_ref=',zref:5:1);
   end;
 0:begin
```

```
      writeln('Band_pass design ...');
      writeln('Z_in(S)=a(S)/b(S);');
      for i:=0 to n do writeln('a[',i:2,']=',(g[n-i]+h[n-i]):10);
      for i:=0 to n do writeln('b[',i:2,']=',(g[n-i]-h[n-i]):10);
      writeln('Z_ref=',zref:5:1);
      end;
  end;
window(1,1,80,25); clrscr;
window(1,21,80,25);
gotoxy(6,1);writeln('F1: Lumped Matching');
gotoxy(53,1);writeln('F2: Distributed Matching');
gotoxy(53,2);writeln('F5:  Return');
end;{procedure dist_gain}

overlay procedure lump_gain;
 var            gopt, h_init,freq,sl_re,
                  zero,sl_im,sg_re,sg_im : vector;
                         h,g,znm,zdn,el : polynomial;
            fmax,tolerance,zref,f_center : real;
                        multip,var_zero : intarr;
                    k, nd,q,dist,i,j,m,n,
  g_type,total,num_zero,nuvzer,nopt,ladder: integer;
                             zerflg,flag : boolean;
                                    opt : char;

begin
  dist:=0; nd:=0; q:=0; zerflg:=false; nuvzer:=0; k:=0; num_zero:=0;
  entry1(freq,sl_re,sl_im,sg_re,sg_im,gopt,zref,m,g_type);
  fmax:=freq[m]; window(1,1,80,25);clrscr;
  writeln('Enter:');
  write('  No. of reactive elements:');readln(n);
  write('  Any transformers (y/n,Def:n)');read(kbd,opt);writeln;
  if opt in ['y','Y'] then
  begin
    ladder:=0;writeln('Design with transformer');
    write('  No. of high_pass elements ');readln(k);
    nopt:=n+1;
  end
  else
  begin
    nopt:=n;write('  Ladder type (h');lowvideo;write('i');
    normvideo;write('/l');lowvideo;write('o');
    normvideo;write(',Def:l');lowvideo;write('o');normvideo;write(')');
    read(kbd,opt);writeln;normvideo;
    if opt in['h','H'] then
    begin
      writeln('High_pass ...');ladder:=-1;k:=n;
    end
    else
    begin
      writeln('Low_pass ...');ladder:=+1;k:=0;
    end;
  end;
```

```
  clrscr; gotoxy(1,1);
  write(' Design with finite-jw transmission zeros (y/n,Def:n)');
  read(kbd,opt); writeln;
  if opt in ['y','Y'] then
  begin
    zerflg:=true;
    write('Enter number of distinct zeros = '); readln(num_zero);
    for i:=1 to num_zero do
    begin
      clrscr; gotoxy(1,1);
      writeln(' Transmission zero ',i:2);
      write('Frequency = '); readln(zero[i]);
      write('Multiplicity ='); readln(multip[i]);
      write('To be optimized (y/n,Def:y)'); read(kbd,opt); writeln;
      if opt in ['n','N'] then var_zero[i]:=0
      else begin var_zero[i]:=1; nuvzer:=nuvzer+1; end;
      zero[i]:=zero[i]/fmax;
    end;
  total:=0; for i:=1 to num_zero do total:=total+multip[i];
  nopt:=nopt+2*total;
  end;
  clrscr; gotoxy(1,1);
  writeln('Initial guess of h(i)''s  ...');writeln('Enter');
  for i:=1 to nopt do
  begin
    j:=i+round((ladder-1.2)/2);
    write('   h(',j:2,')=');readln(h_init[i]);
  end;
  write('     tolerance value ');readln(tolerance);
  for i:=1 to m do freq[i]:=freq[i]/fmax;
  optim(freq,sl_re,sl_im,sg_re,sg_im,gopt,h_init,ladder,k,nd,nopt,n,m,dist,
        tolerance,zref,fmax,f_center,var_zero,multip,zero,nuvzer,num_zero,
        zerflg,g_type);
window(1,1,80,25); clrscr;
window(1,21,80,25);
gotoxy(6,1);writeln('F1: Lumped Matching');
gotoxy(53,1);writeln('F2: Distributed Matching');
gotoxy(53,2);writeln('F5:  Return');
end;{procedure lump_gain}

procedure menu;
var             return : boolean;
                   key : char;
begin
  return:=false;
  window (1,21,80,25);clrscr;
  gotoxy(6,1);writeln('F1: Lumped Matching');
  gotoxy(53,1);writeln('F2: Distributed Matching');
  gotoxy(53,2);writeln('F5:  Return');
  while not return do
  if keypressed then
  begin
    read(kbd,key);
```

```
    if (key=#27) and keypressed then
    begin
      read(kbd,key);
      case key of
      #59 : lump_gain;
      #60 : dist_gain;
      #63 : begin
              window(1,1,80,25); clrscr;
              window(1,21,80,25);
              gotoxy(6,1);writeln('F1 : Optimize Gain');
              gotoxy(53,1);writeln('F2 : Optimize Elements');
              gotoxy(53,2);writeln('F5 : Stop');
              return:=true;
            end;
      end;
    end;
  end;
end;{procedure menu}

begin
  clrscr;lowvideo;
  gotoXY(5,10);write('IMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;');
  gotoXY(5,11);write(':                                                                :');
  gotoXY(5,12);write(':                Broadband Matching Program                      :');
  gotoXY(5,13);write(':                            by                                  :');
  gotoXY(5,14);write(':                Orhan ARIKAN & Emre TELATAR                     :');
  gotoXY(5,15);write(':               Modified   by   Adnan   KÖKSAL                   :');
  gotoXY(5,16);write(':                May 1986, July 1987, Ankara                     :');
  gotoXY(5,17);write(':                                                                :');
  gotoXY(5,18);write('HMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM<');
  delay(3000);clrscr;finish:=false;
    window(1,1,80,25); clrscr;
    window(1,21,80,25);
    gotoxy(6,1);writeln('F1 : Optimize Gain');
    gotoxy(53,1);writeln('F2 : Optimize Elements');
    gotoxy(53,2);writeln('F5 : Stop');
    while not finish do
    if keypressed then
    begin
      read(kbd,key);
      if(key=#27) and keypressed then
      begin
        read(kbd,key);
        case key of
          #59 : menu;
          #60 : elmt;
          #63 : begin
                  finish:=true;
                  window(1,1,80,25); clrscr;
                end;
        end;
      end;
    end;
end.
```

```
{ INCLUDE FILE ENTRY1.PAS }

overlay procedure entry1( var freq,sl_re,sl_im,sg_re,sg_im,gopt : vector;
                var zref: real; var m: integer; var g_type:integer);

type            caps = array[0..1,0..1] of string[50];


var    i,j,xpos,ypos,
           disp,form : integer;
             key,opt : char;
              inpstr : string[10];
              auxstr : string[3];
          cont, exit : boolean;
               table : array[1..6,1..20] of real;

const caption : caps = ((' Freq (Hz)   SL_mag  SL_pha(x) Sg_mag    Sg_pha(x)',
                         ' Freq (Hz) ZL_mag(j) ZL_pha(x) Zg_mag(j) Zg_pha(x)'),
                        (' Freq (Hz)   SL_re     SL_im     Sg_re      Sg_im ',
                         ' Freq (Hz)  ZL_re(j)  ZL_im(j) Zg_re(j)  Zg_im(j) '));

function atan2(x,y:real):real;
begin
if y=0. then
  if x<0 then atan2:=pi
  else atan2:=0.
else if x=0. then
       atan2:=y/abs(y)*pi/2
     else
     begin
       if x>0 then atan2:=arctan(y/x)
       else atan2:=arctan(y/x)+pi;
     end;
end;
procedure re(var x,y : real);
var            xt,yt : real;
begin
  xt:=x*cos(y*pi/180);yt:=x*sin(y*pi/180);x:=xt;y:=yt;
end;
procedure mag(var x,y : real);
        var mag,phase : real;
begin
  mag:=sqrt(sqr(x)+sqr(y));phase:=180/pi*atan2(x,y);x:=mag;y:=phase;
end;
function s_to_z_re(s_re,s_im:real):real;
begin
  if s_re=1. then s_to_z_re:=100.E+12
  else s_to_z_re:=zref*(1-(sqr(s_re)+sqr(s_im)))/(sqr(1-s_re)+sqr(s_im));
end;
function s_to_z_im(s_re,s_im:real):real;
begin
  if s_re=1. then s_to_z_im:=0.
  else s_to_z_im:=2*zref*s_im/((sqr(1-s_re)+sqr(s_im)));
end;
```

```
procedure s_to_z(var x,y : real);
var               xt,yt : real;
begin
  xt:=s_to_z_re(x,y);yt:=s_to_z_im(x,y);x:=xt;y:=yt;
end;
function z_to_s_re(z_re,z_im:real):real;
begin
  z_to_s_re:=(sqr(z_re/zref)+sqr(z_im/zref)-1)/(sqr(z_re/zref+1)+sqr(z_im/zref));
end;
function z_to_s_im(z_re,z_im:real):real;
begin
  z_to_s_im:=2*z_im/zref/(sqr(z_re/zref+1)+sqr(z_im/zref));
end;
procedure z_to_s(var x,y : real);
var               xt,yt : real;
begin
  xt:=z_to_s_re(x,y);yt:=z_to_s_im(x,y);x:=xt;y:=yt;
end;
procedure cursor;
begin
  gotoXY(10*xpos-9,ypos);write('>');
end;
procedure print(x : real; i,j : integer);
var          expo : integer;
           expstr : string[9];
begin
  if i<>6 then
  begin
    expstr:='pnfm kMGT';expo:=0;
    if x<>0. then
    begin
      while (abs(x)>=1000.0) and (expo<4) do begin x:=x/1000.;expo:=expo+1; end;
      while (abs(x)<1.0) and (expo>-4) do begin x:=x*1000.;expo:=expo-1; end;
    end;
  end;
  gotoxy(10*i-9,j);write(x:8:2,expstr[expo+5],' ');
end;{procedure print}

procedure right;
begin
  print(table[xpos,ypos],xpos,ypos);
  if xpos<6 then xpos:=succ(xpos)
  else begin xpos:=1;if ypos<18 then ypos:=succ(ypos);end;
  cursor;inpstr:='';
end;{procedure right}

procedure left;
begin
  print(table[xpos,ypos],xpos,ypos);
  if xpos>1 then xpos:=pred(xpos)
  else begin xpos:=6;if ypos>1 then ypos:=pred(ypos);end;
  cursor;inpstr:='';
end;{procedure left}
```

```
procedure up;
begin
  print(table[xpos,ypos],xpos,ypos);
  if ypos>1 then ypos:=pred(ypos);
  cursor;inpstr:='';
end;{procedure up}

procedure down;
begin
  print(table[xpos,ypos],xpos,ypos);
  if ypos<18 then ypos:=succ(ypos);
  cursor;inpstr:='';
end;{procedure down}

procedure ins;
var   i,j : integer;
begin
  if (m<18) and (ypos<=m) then
  begin
    gotoXY(10*xpos-9,ypos);write(' ');insline;
    for i:=m downto ypos do
      for j:=1 to 6 do
        table[j,i+1]:=table[j,i];
  m:=m+1;cursor;
  end;
end;{procedure ins}

procedure del;
var   i,j : integer;
begin
  if ypos<=m then
  begin
    delline;m:=m-1;cursor;
    for i:=ypos to m do
      for j:=1 to 6 do
        table[j,i]:=table[j,i+1];
  end;
end;{procedure del}

procedure togdisp;
var i : integer;
begin
  window(1,1,80,25);disp:=1-disp;gotoXY(1,1);writeln(caption[disp,form]);
  gotoXY(1,21);
  if disp=0 then writeln('F1 : Mag,Phase / Real,Imag')
  else writeln('F1 : Real,Imag / Mag,Phase');
  if m>0 then
  begin
    gotoXY(1,25);normvideo;write('Update data?(y/n,def:n)');read(kbd,opt);
    gotoXY(1,25);clreol;lowvideo;window(1,2,50,20);
    if opt in ['y','Y'] then
    begin
      for i:=1 to m do
```

```
      begin
        case disp of
          0: begin
               mag(table[2,i],table[3,i]); mag(table[4,i],table[5,i]);
             end;
          1: begin
               re(table[2,i],table[3,i]); re(table[4,i],table[5,i]);
             end;
        end;
        print(table[1,i],1,i);
        print(table[2,i],2,i);print(table[3,i],3,i);
        print(table[4,i],4,i);print(table[5,i],5,i);
      end;
      print(table[xpos,ypos],xpos,ypos);inpstr:='';cursor;
    end
    else
      gotoXY(10*xpos-8+length(inpstr),ypos);
  end;
  gotoXY(2,2);
end;{procedure togdisp}

procedure togform;
begin
  window(1,1,80,25);form:=1-form;gotoXY(1,1);writeln(caption[disp,form]);
  gotoXY(28,21);
  if form=0 then writeln('F2 : Reflection / Impedance')
  else writeln('F2 : Impedance / Reflection');
  if m>0 then
  begin
    gotoXY(1,25);normvideo;write('Update data?(y/n,def:n)');read(kbd,opt);
    gotoXY(1,25);clreol;lowvideo;window(1,2,50,20);
    if opt in ['y','Y'] then
    begin
      for i:=1 to m do
      begin
        case form of
          0:case disp of
              0: begin
                   re(table[2,i],table[3,i]); re(table[4,i],table[5,i]);
                   z_to_s(table[2,i],table[3,i]);
                   z_to_s(table[4,i],table[5,i]);
                   mag(table[2,i],table[3,i]);
                   mag(table[4,i],table[5,i]);
                 end;
              1: begin
                   z_to_s(table[2,i],table[3,i]);
                   z_to_s(table[4,i],table[5,i]);
                 end;
            end;
          1:case disp of
              0: begin
                   re(table[2,i],table[3,i]); re(table[4,i],table[5,i]);
                   s_to_z(table[2,i],table[3,i]);
```

```
                s_to_z(table[4,i],table[5,i]);
                mag(table[2,i],table[3,i]); mag(table[4,i],table[5,i]);
              end;
            1: begin
                s_to_z(table[2,i],table[3,i]);
                s_to_z(table[4,i],table[5,i]);
              end;
          end;
      end;
      print(table[1,i],1,i);
      print(table[2,i],2,i);print(table[3,i],3,i);
      print(table[4,i],4,i);print(table[5,i],5,i);
    end;
    print(table[xpos,ypos],xpos,ypos);inpstr:='';cursor;
  end
  else
    gotoXY(10*xpos-8+length(inpstr),ypos);
 end;
 gotoXY(2,2);
end;{procedure togform}

procedure enter;
var   x:real;code:integer;
begin
  if length(inpstr)>0 then
  begin
    val(inpstr,x,code);inpstr:='';
    if code=0 then
    begin
      table[xpos,ypos]:=x;
      if m<ypos then m:=ypos;
    end;
  end
  else
  begin
    if ((xpos=4) or (xpos=5)) and (ypos>1) then
    begin
      table[xpos,ypos]:=table[xpos,ypos-1];inpstr:='';
      if m<ypos then m:=ypos;
    end;
    if  ((xpos=2) or (xpos=3)) and (ypos>1) then
    begin
      table[xpos,ypos]:=table[xpos,ypos-1];inpstr:='';
      if m<ypos then m:=ypos;
    end;
    if (xpos=6) and (ypos>1) then
    begin
      table[xpos,ypos]:=table[xpos,ypos-1];inpstr:='';
      if m<ypos then m:=ypos;
    end;
end;
  right;
end;{procedure enter}
```

```
procedure menu_entry;
begin
  normvideo;
  window(1,1,80,25);gotoxy(1,1);writeln(caption[disp,form]);
  window(1,21,80,25);clrscr;
  gotoXY(1,1);
  if disp=0 then writeln('F1 : Mag,Phase / Real,Imag')
  else writeln('F1 : Real,Imag / Mag,Phase');
  gotoXY(28,1);
  if form=0 then writeln('F2 : Reflection / Impedance')
  else writeln('F2 : Impedance / Reflection');
  gotoxy(56,1);writeln('F3:  MORE');
  gotoxy(1,2);writeln('F4:  Save to disk');
  gotoxy(28,2);writeln('F5:  Load from disk');
  gotoxy(56,2);writeln('F6:  Exit');
  gotoxy(1,3);writeln('Del: Deletes line');
  gotoxy(28,3);writeln('Ins: Inserts line');
  gotoxy(56,3);writeln('Ctrl-c: Aborts');
end;{procedure menu_entry}

procedure save;
var         fil : file of real;
       filename : string[20];
xg_type,xm,xform,xdisp : real;
              i : integer;
         ok,nop : boolean;

begin
  window(1,24,80,25);gotoXY(1,1);
  nop:=true;
  normvideo;write('Filename : ');readln(filename);
  assign(fil,filename);{$I-} reset(fil); {$I+}
  ok:=(ioresult=0);
  if ok then
  begin
    normvideo;writeln('File exists, overwrite (y/n Def:n)');read(kbd,opt);
    if opt in ['y','Y'] then nop:=false
    else nop:=true;
  end;
  if not nop then
  begin
    rewrite(fil);xm:=m;xform:=form;xdisp:=disp;xg_type:=g_type;write(fil,xm,xdisp,xform,xg_type,zref);
    for i:=1 to m do write(fil,table[1,i],table[2,i],table[3,i],table[4,i],table[5,i],table[6,i]);
  end;
  close(fil);
  clrscr;
  window(1,2,80,20);print(table[xpos,ypos],xpos,ypos);cursor;
end;{procedure save}

procedure load;
var         fil : file of real;
       filename : string[20];
xg_type,xm,xform,xdisp : real;
```

```
          i,j : integer;
        ok,nop : boolean;

begin
  window(1,24,80,25);gotoXY(1,1);
  nop:=true;
  normvideo;write('Filename : ');readln(filename);
  assign(fil,filename);{$I-} reset(fil); {$I+}
  nop:=not (ioresult=0);
  if not nop then
  begin
    read(fil,xm,xdisp,xform,xg_type,zref);m:=trunc(xm);form:=trunc(xform);disp:=trunc(xdisp);
    g_type:=trunc(xg_type);
    for i:=1 to m do read(fil,table[1,i],table[2,i],table[3,i],table[4,i],table[5,i],table[6,i]);
    window(1,1,80,25);clrscr;gotoXY(1,1);writeln(caption[disp,form]);gotoxy(52,1);
    if g_type=0 then writeln('Gain (ratio)')
    else writeln('  Gain ( dB )');
    window(1,21,80,25);gotoXY(1,1);
    if disp=0 then writeln('F1 : Mag,Phase / Real.Imag')
    else writeln('F1 : Real,Imag / Mag,Phase');
    gotoxy(28,1);
    if form=0 then writeln('F2 : Reflection / Impedance')
    else writeln('F2 : Impedance / Reflection');
    gotoxy(56,4);writeln('Z_ref=',zref:5:1);
    menu_entry;
    window(1,2,80,20);
    for i:=1 to m do
      for j:=1 to 6 do
        print(table[j,i],j,i);
    xpos:=1;ypos:=1;cursor;
  end
  else
  begin
    normvideo;writeln('File cannot be accessed');lowvideo;delay(1000);clrscr;
    window(1,2,50,20);gotoXY(10*xpos-8+length(inpstr),ypos);
  end;
end;{procedure load}


procedure log(var x: real);
begin
x:=10*ln(x)/ln(10);
end;{procedure log}

procedure inv_log(var x: real);
begin
x:=exp(x/10*ln(10));
end;{procedure inv_log}


procedure gzref;
begin
  window(1,21,80,25);clrscr;gotoXY(1,4);normvideo;write('Enter Z_ref(j) ');read(zref);
```

```
  gotoXY(1,4);clreol;gotoXY(56,4);clreol;lowvideo;write('F3 : Z_ref=',zref:5:1,'j');
  window(1,2,50,20);gotoXY(10*xpos-8+length(inpstr),ypos);
end;{procedure gzref}


procedure tog_gain;
var              opt  :char;
  begin
  clrscr;window(1,1,80,20);g_type:=1-g_type;
  gotoxy(52,1);
  if g_type=0 then writeln('Gain (ratio)')
  else writeln(' Gain ( dB )');
  window(1,21,80,25);gotoxy(1,1);write('update data? (y/n,def:n)');
  read(kbd,opt);writeln;
  if opt in ['y','Y'] then
    begin
    window(1,1,80,20);
    for i:=1 to m do
      begin
      if g_type=1 then
        begin
          log(table[6,i]);print(table[1,i],1,i);print(table[2,i],2,i);
          print(table[3,i],3,i);print(table[4,i],4,i);print(table[5,i],5,i);
          print(table[6,i],6,i);
        end
        else
        begin
          inv_log(table[6,i]);print(table[1,i],1,i);print(table[2,i],2,i);
          print(table[3,i],3,i);print(table[4,i],4,i);print(table[5,i],5,i);
          print(table[6,i],6,i);
        end;
        print(table[xpos,ypos],xpos,ypos);inpstr:='';cursor;
      end;
    end
  else
  gotoxy(10*xpos-8+length(inpstr),ypos);
  end;{procedure tog_gain}


procedure more;
var                          key  : char;
                            exit2 : boolean;
begin
  window(1,21,80,25);clrscr;normvideo;exit2:=false;
  gotoxy(1,1);writeln('F1 : Change Z_ref(',zref:5:1,')');
  gotoxy(1,2);
  if g_type=0 then writeln('F2 : Power ratio / dB')
  else writeln('F2 : dB / Power ratio');
  while not exit2 do
    if keypressed then
    begin
      read(kbd,key);
      if (key=#27) and keypressed then
```

```
      begin
        read(kbd,key);
        case key of
          #59 : begin
                  gzref;
                  exit2:=true;
                end;
          #60 : begin
                  tog_gain;
                  exit2:=true;
                end;
              end;
            end;
          end;
        menu_entry;
        window(1,2,50,20);gotoxy(10*xpos-8+length(inpstr),ypos);
      end;{procedure more}

begin
  if lumpelmt=true then cont:=true;
  window(1,1,80,25);clrscr;normvideo;cont:=false;gotoxy(1,1);
  writeln(' If the data is frequency independent, after entering');
  writeln(' the first line of data press ENTER only when the same');
  writeln(' data is required ..................................');
  writeln;writeln;writeln;writeln;
  writeln('                                Press   F1   to continue');
  while not cont do
    if keypressed then
    begin
      read(kbd,key);
      if (key=#27) and keypressed then
      begin
        read(kbd,key);
        if(key=#59) then cont:=true;
      end;
    end;
  zref:=50;xpos:=1;ypos:=1;m:=0;inpstr:='';disp:=0;form:=0;
  exit:=false;g_type:=0;
  for i:=1 to 5 do for j:=1 to 20 do table[i,j]:=0.;
  clrscr;togdisp;togform;window(1,1,80,25);
  gotoxy(52,1);
  if g_type=0 then writeln('Gain (ratio)')
  else writeln('  Gain ( dB )');
  menu_entry;
  window(1,2,80,20);gotoXY(1,1);write('>');
  while not exit do
    if keypressed then
    begin
      read(kbd,key);
      if (key=#27) and keypressed then
      begin
        read(kbd,key);
        case key of
```

```
        #59..#64:case key of
                   #59:togdisp;
                   #60:togform;
                   #61:more;
                   #62:save;
                   #63:load;
                   #64:exit:=true;
                 end;
        #75,#77,#72,#80,#82,#83:case key of
                                  #75:left;
                                  #77:right;
                                  #72:up;
                                  #80:down;
                                  #82:ins;
                                  #83:del;
                                end;
      end;
    end
    else
      case key of
        #48..#57,#45,#46:begin
                           write(key:1);inpstr:=inpstr+key;
                         end;
        #71,#77,#84,#107,
        #109,#110,#112,#117:begin
                              str(3*(pos(key,'pnum kMGT')-5),auxstr);
                              inpstr:=inpstr+'E'+auxstr;
                              enter;
                            end;
        #13:enter;
      end;
  end;
for i:=1 to m do
begin
  freq[i]:=table[1,i];
  case disp of
    0: case form of
         0:begin
             re(table[2,i],table[3,i]);re(table[4,i],table[5,i]);
             sl_re[i]:=table[2,i];sl_im[i]:=table[3,i];
             sg_re[i]:=table[4,i];sg_im[i]:=table[5,i];
           end;
         1:begin
             re(table[2,i],table[3,i]);re(table[4,i],table[5,i]);
             sl_re[i]:=z_to_s_re(table[2,i],table[3,i]);
             sl_im[i]:=z_to_s_im(table[2,i],table[3,i]);
             sg_re[i]:=z_to_s_re(table[4,i],table[5,i]);
             sg_im[i]:=z_to_s_im(table[4,i],table[5,i]);
           end;
       end;
    1: case form of
         0:begin
             sl_re[i]:=table[2,i];sl_im[i]:=table[3,i];
```

```
            sg_re[i]:=table[4,i];sg_im[i]:=table[5,i];
          end;
        1:begin
            sl_re[i]:=z_to_s_re(table[2,i],table[3,i]);sl_im[i]:=z_to_s_im(table[2,i],table[3,i]);
            sg_re[i]:=z_to_s_re(table[4,i],table[5,i]);sg_im[i]:=z_to_s_im(table[4,i],table[5,i]);
          end;
      end;
  end;
  case g_type of
    0: gopt[i]:=table[6,i];
    1: begin
         {@inv_log(table[6,i]);}
         gopt[i]:=table[6,i];
       end;
    end;
end;
end;{procedure entry1}

{END OF INCLUDE FILE ENTRY1.PAS}
```

```
{ INCLUDE FILE ENTER.PAS}

overlay procedure enter(var fnode,tnode,el_type,vari1,vari2 : intarr;
          fmax,zref :real;
          var l_limit,u_limit,init_el,llang,ulang,init_ang,ffreq : vector;
          var num_el,num_var : integer);

type                              ij = string[24];
                                  tt = array[1..4,1..20] of ij;
var               m,i,j,xpos,ypos,oc          : integer;
                  key,opt                     : char;
                  inpstr                      : string[24];
                  st                          : string[15];
                  auxstr                      : string[3];
                  exit,lumpelmt,distelmt      : boolean;
                  xx,norm                     : real;
                  table                       :tt;


procedure movel;
begin
  if xpos=1 then   gotoxy(2,ypos);
  if xpos=4 then gotoxy(62,ypos);
  if xpos=2 then gotoxy(10,ypos);
  if xpos=3 then gotoxy(26,ypos);
end;{procedure movel}

procedure cursorl;
begin
  movel;
  write('>');
end;{procedure cursorl}

procedure printl(x : ij; i,j : integer);
var l : integer;
begin
  xpos:=i; ypos:=j;
  l:=length(inpstr);
  movel;
  write(x:l+1);
end;{procedure printl}

procedure rightl;
begin
  printl(table[xpos,ypos],xpos,ypos);
  if xpos<4 then xpos:=succ(xpos)
  else
  begin
    xpos:=1; if ypos<18 then ypos:=succ(ypos);
  end;
  cursorl;inpstr:='';
end;{procedure rightl}
```

```
procedure leftl;
begin
  if xpos>1 then xpos:=pred(xpos)
  else
  begin
    xpos:=4;
    if ypos>1 then ypos:=pred(ypos);
  end;
  cursorl;inpstr:='';
end;{procedure leftl}

procedure upl;
begin
  if ypos>1 then ypos:=pred(ypos);
  cursorl;inpstr:='';
end;{procedure upl}

procedure downl;
begin
  if ypos<18 then ypos:=succ(ypos);
  cursorl;inpstr:='';
end;{procedure downl}

procedure insl;
var   i,j,k : integer;
begin
  k:=4;
  if (m<18) and (ypos<=m) then
  begin
    movel;write(' ');insline;
    for i:=m downto ypos do
      for j:=1 to k do
        table[j,i+1]:=table[j,i];
  m:=m+1;cursorl;
  end;
end;{procedure insl}

procedure dell;
var   i,j,k : integer;
begin
  k:=4;
  if ypos<=m then
  begin
    delline;m:=m-1;cursorl;
    for i:=ypos to m do
      for j:=1 to k do
        table[j,i]:=table[j,i+1];
  end;
end;{procedure dell}

procedure enterl;
begin
  table[xpos,ypos]:=inpstr;
```

```
  if m<ypos then m:=ypos;
  right1;
end;{procedure enter1}

procedure save1;
type        ij=string[24];
var        fil : file of ij;
            mm : ij;
      filename : string[20];
             i : integer;
       ok,nop : boolean;

begin
  window(1,24,80,25);gotoXY(1,1);
  nop:=true;
  normvideo;write('Filename : ');readln(filename);
  assign(fil,filename);{$I-} reset(fil); {$I+}
  ok:=(ioresult=0);
  if ok then
  begin
    normvideo;writeln('File exists, overwrite (y/n Def:n)');read(kbd,opt);
    if opt in ['y','Y'] then nop:=false
    else nop:=true;
  end;
  if not nop then
  begin
    rewrite(fil); str(m,mm);write(fil,mm);
    for i:=1 to m do
     write(fil,table[1,i],table[2,i],table[3,i],table[4,i]);
  end;
  close(fil);clrscr;
  window(1,2,80,20);
end;{procedure save1}

procedure load1;
type        ij = string[24];
var        fil : file of ij;
      filename : string[20];
            mm : ij;
    k,i,code,j : integer;
       ok,nop : boolean;

begin
  window(1,24,80,25);gotoXY(1,1); nop:=true;
  normvideo;write('Filename : ');readln(filename);
  assign(fil,filename);{$I-} reset(fil); {$I+}
  nop:=not (ioresult=0);
  if not nop then
  begin
    read(fil,mm);val(mm,m,code);
    for i:=1 to m do
     read(fil,table[1,i],table[2,i],table[3,i],table[4,i]);
    window(1,1,80,25);clrscr;gotoXY(1,1);
```

```
    writeln('  Nodes-Type      Values');
    window (1,21,80,25);clrscr;normvideo;
    gotoxy(1,2);writeln('F4:  Save to disk');
    gotoxy(28,2);writeln('F5:  Load from disk');
    gotoxy(56,2);writeln('F6:  Exit');
    gotoxy(1,3);writeln('Del: Deletes line');
    gotoxy(28,3);writeln('Ins: Inserts line');
    gotoxy(56,3);writeln('Ctrl-c: Aborts');
    window(1,2,80,20); k:=4;
    for i:=1 to m do
      for j:=1 to k do
        printl(table[j,i],j,i);
    xpos:=1;ypos:=1;cursorl;
  end
  else
  begin
    normvideo;writeln('File cannot be accessed');lowvideo;delay(1000);clrscr;
    window(1,2,50,20);
  end;
end;{procedure loadl}

procedure inpreter(st : ij;var x : real);
var               exp : string[3];
                  opt : char;
       l,code,num,numl : integer;
begin
  num:=length(st);
  opt:=copy(st,num,1);
  numl:=pos(opt,'pnum kMGT');
  if numl=0 then val(st,x,code)
  else
  begin
    delete(st,num,num);
    num:=3*numl-15;str(num,exp);
    st:=st+'E'+exp;val(st,x,code);
     st:='';
  end;
end;{procedure inpreter}

procedure interpreter(table : tt);
var      i,j,code,oc,ocl : integer;
begin
  for i:=1 to m do
  for j:=1 to 3 do
  begin
    oc:=pos('/',table[1,i]);st:=copy(table[1,i],1,oc-1);
    val(st,xx,code);
    if j=1 then fnode[i]:=trunc(xx);
    if j=2 then tnode[i]:=trunc(xx);
    if j=3 then el_type[i]:=trunc(xx);
    if el_type[i]=0 then
    begin
      lumpelmt:=true;
      norm:=2*fmax*PI*zref;
    end;
```

```
    if el_type[i]=1 then
    begin
      lumpelmt:=true;
      norm:=2*fmax*PI/zref;
    end;
    if (el_type[i]<>0) and (el_type[i]<>1) then lumpelmt:=false;
    delete(table[1,i],1,oc);
  end;
  if lumpelmt=true then
  for i:=1 to m do
  begin
    oc:=pos('/',table[2,i]);st:=copy(table[2,i],1,oc-1);
    delete(table[2,i],1,oc);oc1:=pos('/',table[2,i]);
    if oc1=0 then
    begin
      opt:=copy(st,oc-1,1);
      if opt in ['c','C'] then
      begin
        delete(st,oc-1,oc-1);inpreter(st,init_el[i]);
        init_el[i]:=init_el[i]*norm;
        vari[i]:=0;
        l_limit[i]:=0.01*norm;u_limit[i]:=100*norm;
      end
      else
      begin
        inpreter(st,init_el[i]); init_el[i]:=init_el[i]*norm;
        vari[i]:=1;
        l_limit[i]:=0.01*norm;u_limit[i]:=100*norm;
      end;
    end
    else
    begin
      inpreter(st,l_limit[i]);oc:=pos('/',table[2,i]);l_limit[i]:=l_limit[i]*norm;
      st:=copy(table[2,i],1,oc-1);delete(table[2,i],1,oc);
      inpreter(st,init_el[i]);vari[i]:=1;init_el[i]:=init_el[i]*norm;
      oc:=pos('/',table[2,i]);st:=copy(table[2,i],1,oc-1);
      inpreter(st,u_limit[i]);u_limit[i]:=u_limit[i]*norm;
    end;
  end;
  if lumpelmt=false then
  begin
    for i:=1 to m do
    for j:=1 to 3 do
    begin
      oc:=pos('/',table[j+1,i]);st:=copy(table[j+1,i],1,oc-1);
      delete(table[j+1,i],1,oc);oc1:=pos('/',table[j+1,i]);
      if oc1=0 then
      begin
        opt:=copy(st,oc-1,1);
        if opt in['c','C'] then
        begin
          delete(st,oc-1,oc-1);
          if j=1 then
```

```
      begin
        inpreter(st,init_el[i]); init_el[i]:=init_el[i]/zref;
        vari1[i]:=0;
      end;
      if j=2 then
      begin
        inpreter(st,init_ang[i]); init_ang[i]:=init_ang[i]*PI/180;
        vari2[i]:=0;
      end;
      if j=3 then inpreter(st,ffreq[i]); ffreq[i]:=ffreq[i]/fmax;
    end
    else
    begin
      if j=1 then
      begin
        inpreter(st,init_el[i]); init_el[i]:=init_el[i]/zref;
        vari1[i]:=1;
        l_limit[i]:=0.;
        u_limit[i]:=5.;
      end;
      if j=2 then
      begin
        inpreter(st,init_ang[i]);init_ang[i]:=init_ang[i]*PI/180;
        vari2[i]:=1;
        llang[i]:=0.;
        ulang[i]:=5*PI;
      end;
    end;
  end
  else
  begin
    if j=1 then
    begin
      inpreter(st,l_limit[i]);l_limit[i]:=l_limit[i]/zref;
      vari1[i]:=1;
      oc:=pos('/',table[2,i]);st:=copy(table[2,i],1,oc-1);
      delete(table[2,i],1,oc);inpreter(st,init_el[i]);init_el[i]:=init_el[i]/zref;
      oc:=pos('/',table[2,i]);st:=copy(table[2,i],1,oc-1);
      delete(table[2,i],1,oc);inpreter(st,u_limit[i]);
      u_limit[i]:=u_limit[i]/zref;
    end;
    if j=2 then
    begin
      inpreter(st,llang[i]); llang[i]:=llang[i]*PI/180;
      vari2[i]:=1;
      oc:=pos('/',table[3,i]);st:=copy(table[3,i],1,oc-1);
      delete(table[3,i],1,oc);inpreter(st,init_ang[i]);
      init_ang[i]:=init_ang[i]*PI/180;
      oc:=pos('/',table[3,i]);st:=copy(table[3,i],1,oc-1);
      delete(table[3,i],1,oc);inpreter(st,ulang[i]);
      ulang[i]:=ulang[i]*PI/180;
    end;
  end;
```

```
    end;
  end;
end;{procedure interpreter}

begin
  xpos:=1;ypos:=1; num_el:=0;inpstr:='';exit:=false;m:=0;
  window(1,1,80,25);clrscr;normvideo;gotoxy(1,1);
  for i:=1 to 4 do
  for j:=1 to 20 do
  table[i,j]:='';
  window(1,21,80,25);clrscr;
  gotoxy(1,2);writeln('F4:  Save to disk');
  gotoxy(28,2);writeln('F5:  Load from disk');
  gotoxy(56,2);writeln('F6:  Exit');
  gotoxy(1,3);writeln('Del: Deletes line');
  gotoxy(28,3);writeln('Ins: Inserts line');
  gotoxy(56,3);writeln('Ctrl-c: Aborts');
  window(1,2,80,20);gotoXY(1,1);write('>');
  while not exit do
    if keypressed then
    begin
      read(kbd,key);
      if (key=#27) and keypressed then
      begin
        read(kbd,key);
        case key of
          #62..#64:case key of
                     #62:savel;
                     #63:loadl;
                     #64:exit:=true;
                   end;
          #75,#77,#72,#80,#82,#83:case key of
                                    #75:leftl;
                                    #77:rightl;
                                    #72:upl;
                                    #80:downl;
                                    #82:insl;
                                    #83:dell;
                                  end;
        end;
      end
      else
        case key of
          #48..#57,#45,#46,
          #71,#77,#84,#107,
          #109,#110,#112,#117,
          #47,#63,#99,#67    :begin
                                write(key:1);inpstr:=inpstr+key;
                              end;
          #13:enterl;
        end;
    end;
  for i:=1 to m do
```

```
  begin
    init_el[i]:=0;vari1[i]:=0;vari2[i]:=0;ffreq[i]:=0;
    init_ang[i]:=0;llang[i]:=0;ulang[i]:=0;el_type[i]:=0;
  end;
interpreter(table);
num_el:=m;
num_var:=0;
for i:=1 to num_el do
begin
  if vari1[i]=1 then num_var:=num_var+1;
  if vari2[i]=1 then num_var:=num_var+1;
end;
end;{procedure enter}

{ END OF INCLUDE FILE ENTER.PAS }
```