

INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.
2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

**University
Microfilms
International**

300 N. Zeeb Road
Ann Arbor, MI 48106

Order Number 1332408

An interactive program for elliptic filters using an IBM personal computer

Al-Zariey, Mohamed, M.S.

The University of Arizona, 1987

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

AN INTERACTIVE PROGRAM FOR ELLIPTIC FILTERS
USING AN IBM PERSONAL COMPUTER

by

Mohamed Al-Zariey

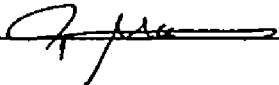
A Thesis Submitted to the Faculty of the
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN ELECTRICAL ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1987

STATEMENT BY AUTHOR

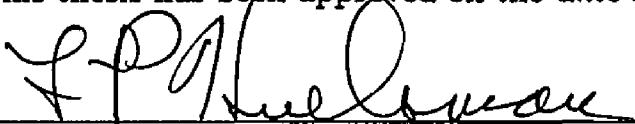
This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: 

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:



L. P. HUELSMAN

Professor of Electrical and Computer Engineering

12/17/87

Date

ACKNOWLEDGMENTS

I would like to express my sincere thanks and gratitude to my advisor, professor Lawrence P. Huelsman, for the endless support and encouragement during the development of the instructional computer program, PC-ELLIP. During all phases of this work, he offered his valuable advice and suggestions. He was of great help in developing and improving the program.

Last, but far from least, come the reviewers: Professor Donald G. Dudley and professor Roy H. Mattson of the University of Arizona had a major hand in shaping the final manuscript. Their comments, suggestions and expert advice were of great help. Thanks to both of them.

I have enjoyed writing this thesis. I hope that you enjoy reading it.

TABLE OF CONTENTS

	page
LIST OF ILLUSTRATIONS	v
LIST OF TABLES	vi
ABSTRACT	vii
CHAPTER	
1. INTRODUCTION	1
2. MAGNITUDE APPROXIMATION	3
THE ELLIPTIC CHARACTERISTIC	3
2.1 THE ELLIPTIC RESPONSE	3
Elliptic Functions	5
The Jacobi's Functions with imaginary arguments	7
2.2 THE CHARACTERISTIC FUNCTION $F_n(\omega)$	9
A. n odd	11
B. n even	12
2.3 POLES AND ZEROS OF ELLIPTIC RESPONSE	13
3. SYNTHESIS OF <i>LC</i> DOUBLY-TERMINATED NETWORKS	16
4. USER GUIDE: DETAILS PROGRAM DESCRIPTION	26
How to Run the Program	28
5. SUMMARY AND RECOMMENDATIONS	32
APPENDIX A INPUT/OUTPUT EXAMPLES	33
APPENDIX B PROGRAM LISTING	49
REFERENCES	104

LIST OF ILLUSTRATIONS

Figure		page
2-1	Double terminated two-port networks	4
2-2	Typical response of equiripple low-pass filter	4
2-3	Typical plot of the behavior of $F_n(\omega)$	10
3-1	Typical frequency response for low-pass filter, ($n = \text{odd}$)	18
3-2	Frequency responses of elliptic low-pass filters, ($n = \text{even}$) . . .	20
3-3	Elliptic lowpass ladder circuit	21

LIST OF TABLES

Table	page
3-1 Design Impedance Equations for Doubly terminating Filters	18
3-2 Elements Values and Remainders of Functions of Lowpass Filters	22
A-1 Third Order Filter ($\theta = 30^\circ$, $\rho = 20\%$, $n = 3$)	34
A-2 Third Order Filter ($\omega_s = 1.2rad/s$, $A_{max} = 0.01dB$, $n = 3$)	35
A-3 Fourth Order Filter Case B	36
A-4 Fourth Order Filter Case C "	37
A-5 Fifth Order Filter	38
A-6 Sixth Order Filter Case A	40
A-7 Sixth Order Filter Case B	41
A-8 Sixth Order Filter Case C	43
A-9 Seventh Order Filter	45
A-10 Tenth Order Filter Case B	47

ABSTRACT

The purpose of this thesis is to present a mathematical model in which we describe the theoretical approximation of the elliptic passive filters. From this theoretical treatment, the transfer function and its magnitude, as well as its poles and zeros, were programmed in Turbo Pascal for AT&T and IBM (PC/XT/AT) personal computers and their compatibles. The program, PC-ELLIP, also calculates the element values for double terminated two-port networks. Source code is included.

CHAPTER 1

INTRODUCTION

Filter networks are of great importance in almost all fields of electrical measurements and equipment engineering. At present, there are several categories of electrical networks in use, such as Butterworth, Chebyshev, Inverse Chebyshev, and Elliptic filters.

In the Butterworth approximation, the loss increases monotonically as a function of the frequency ω , and as ω approaches ω_c , the approximation to a flat passband gets progressively worse. A more balanced characteristic can be achieved by employing the Chebyshev approximation, in which the passband loss oscillates between zero and a prescribed maximum A_{max} . Although a Chebyshev filter leads to a much better passband characteristic than Butterworth filter, the corresponding stopband characteristics are similar. They are both very good at high frequencies, but tend to deteriorate progressively as the frequency is decreased. An improved stopband characteristic can be achieved by using the elliptic approximation. In this the passband loss oscillates between zero and a prescribed maximum A_{max} as in the Chebyshev approximation, and the stopband loss oscillates between infinity and a prescribed minimum A_{min} .

The elliptic approximation is more efficient than the preceding two in that the transition between passband and stopband is steeper for a given approximation order. However, the derivation of the elliptic approximation is somewhat complicated, as it entails a basic understanding of the elliptic functions. Fortunately, however, the ultimate results can be put in a simple form which is easy to apply.

The purpose of this thesis is to present a mathematical model in which we describe the theoretical approximation of the elliptic passive filters. Our simple approach to this approximations follows the formulation presented by W. K. Chen [1]. A summary of the main formulas are presented in Chapter 2. From this theoretical treatment, the transfer function and its magnitude, as well as its poles and zeros, were programed in Turbo Pascal for AT&T and IBM (PC/XT/AT) personal computers and their compatibles.

Having determined the transfer function $N(s)$ and the characteristic function $F_n(\omega)$, one can obtain the elliptic passive filter element values for double terminated two-port networks as described in Chapter 3.

Although the theoretical model presented in Chapters 2 & 3 is different from those of Ref. [3, 4, & 5], the output results from our interactive program, PC-ELLIP, are in agreement with the tables published in these references. Chapter 4 is devoted to a description of this interactive program, PC-ELLIP, in more detail. Input and output examples are included in Appendix A.

CHAPTER 2

MAGNITUDE APPROXIMATION

THE ELLIPTIC CHARACTERISTIC

2.1. THE ELLIPTIC RESPONSE

The elliptic low-pass voltage transfer function for a two-port network is defined by:

$$N(s) = \frac{V_2(s)}{V_1(s)}, \quad (2.1)$$

where $V_2(s)$ is the voltage at port 2, and $V_1(s)$ is the voltage at port 1 as shown in Fig. 2-1. The quantity s is the complex frequency defined by $s = \sigma + j\omega$, where σ and ω are real numbers. For $s = j\omega$ the magnitude squared of $N(s)$ can be approximated by

$$\left| N(j\frac{\omega}{\omega_c}) \right|^2 = \frac{H_n}{1 + \epsilon^2 F_n^2(\frac{\omega}{\omega_c})}, \quad H_n \geq 0 \quad (2.2)$$

where

$$F_n(\frac{\omega}{\omega_c}) = sn \left[\frac{nK_1}{K} sn^{-1}(\frac{\omega}{\omega_c}, k), k_1 \right] \quad (2.3a)$$

for n odd, and

$$F_n(\frac{\omega}{\omega_c}) = sn \left[K_1 + \frac{nK_1}{K} sn^{-1}(\frac{\omega}{\omega_c}, k), k_1 \right] \quad (2.3b)$$

for n even. A typical plot for (2.2) is shown in Fig. 2-2. The quantity ϵ in Eq. (2.2), called the ripple factor, controls the peak-to-peak ripple in the passband and is given by

$$\epsilon^2 = 10^{0.1A_{max}} - 1$$

where A_{max} is the passband ripple (measured in dB).

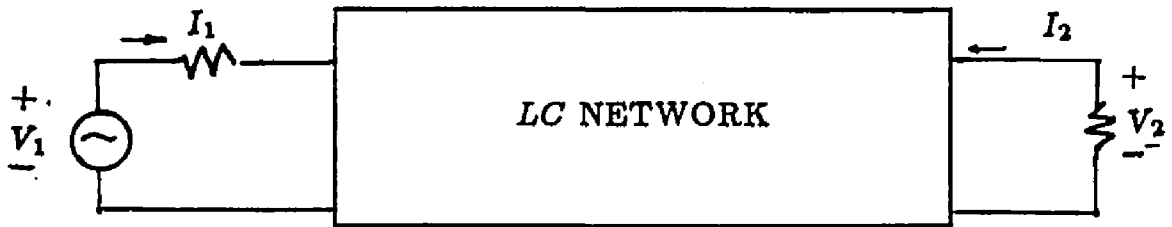
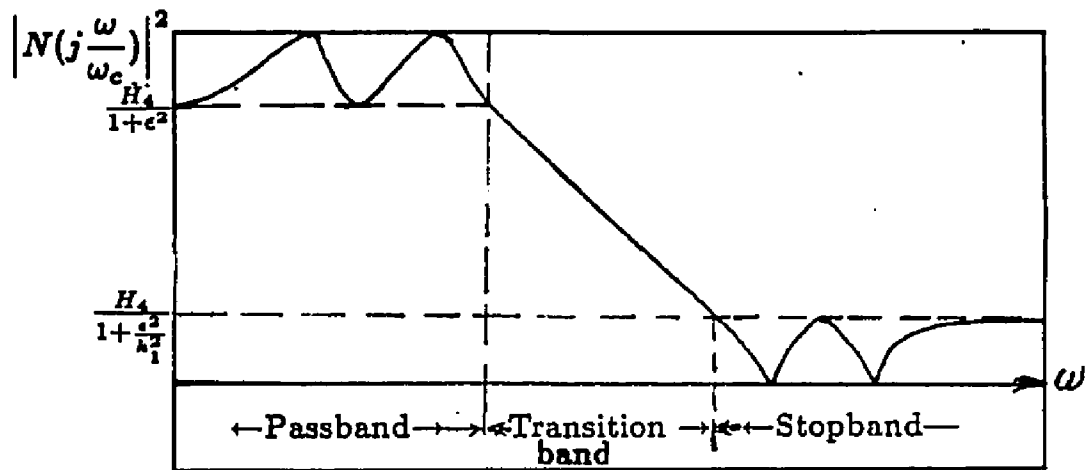


Fig 2-1 Double terminated two-port network.



$$n = 4$$

Fig. 2-2 Typical response of equiripple lowpass filter (Transducer power gain).

The constant H_n is the D. C. gain which may be greater than, equal to, or less than unity. The parameter ω_c is the cut-off frequency. To simplify the calculations, the cut-off frequency, ω_c , will be set to unity. Equations (2.3) are written in terms of the Jacobian elliptic sine, function $sn(u, k)$, and the complete elliptic integrals, $K = K(k)$, $K_1 = K(k_1)$, with moduli k , and k_1 , respectively. Those are now introduced very briefly.

Elliptic functions

The elliptic integral of the *first kind* is defined as

$$u \equiv F(\phi, k) = \int_0^x \frac{dx}{\sqrt{(1-x^2)(1-k^2x^2)}} = \int_0^\phi \frac{d\phi}{\sqrt{1-k^2 \sin^2 \phi}} \quad (2.4)$$

where $0 \leq k \leq 1$ is called the modulus of the elliptic integral and

$$x \equiv \sin \phi.$$

The upper limit ϕ is considered as a function of $u = F(k, \phi)$ and k . Thus ϕ is written as

$$\phi = am(u, k), \quad (2.5)$$

which reads ϕ is the *amplitude* of u of modulus k . Thus, we have

$$\sin \phi = \sin am(u, k).$$

To simplify this expression, we write

$$\sin \phi \equiv sn(u, k),$$

which is called the *Jacobian elliptic function of modulus k* . If $\phi = \pi/2$, the integral (2.4) is known as the *complete elliptic integral of the first kind*. Thus

$$K\left(\frac{\pi}{2}, k\right) = \int_0^{\frac{\pi}{2}} \frac{d\phi}{\sqrt{1-k^2 \sin^2 \phi}} \quad (2.6a)$$

The parameter k is defined by $k = \omega_c/\omega_s$ which is a measure of the steepness of the gain characteristic in the transition band, and is referred to as the selectivity factor. The reciprocal of the selectivity factor is called the steepness. Here, ω_s is the stopband frequency.

The complementary elliptic integral of the first kind is denoted by

$$K_1\left(\frac{\pi}{2}, k_1\right) = \int_0^{\frac{\pi}{2}} \frac{d\phi}{\sqrt{1 - k_1^2 \sin^2 \phi}} \quad (2.6b)$$

in which the parameter k_1 is given by

$$k_1^2 = \frac{10^{0.1A_{max}} - 1}{10^{0.1A_{min}} - 1},$$

where A_{min} is the minimum attenuation in the stopband with respect to the maximum value at the passband (measured in dB).

The Jacobian elliptic functions are defined as follows:

$$\begin{aligned} \text{elliptic sine} & \quad sn(u, k) \equiv \sin \phi, \\ \text{elliptic cosine} & \quad cn(u, k) \equiv \cos \phi, \\ \text{elliptic tan} & \quad tn(u, k) = \frac{sn(u, k)}{cn(u, k)} \equiv \tan \phi, \\ \text{and} & \quad dn(u, k) = \sqrt{1 - k^2 \sin^2 \phi}, \end{aligned} \quad (2.7)$$

where all arguments of Jacobi's elliptic functions in (2.7) are real. It easy to show that

$$\begin{aligned} sn^2(u, k) + cn^2(u, k) &= 1, \\ dn^2(u, k) + k^2 sn^2(u, k) &= 1. \end{aligned} \quad (2.8)$$

The Jacobi's functions with imaginary arguments

By introducing the transformation

$$\sin \theta = j \tan \phi,$$

we can show that the function $sn(ju, k)$ is imaginary while $cn(ju, k)$ and $dn(ju, k)$ are real. Let us write

$$\cos \theta = (1 - \sin^2 \theta)^{1/2} = (1 + \tan^2 \phi)^{1/2} = \sec \phi,$$

$$\sin \phi = \cos \phi \tan \phi = -j \tan \theta,$$

$$d\theta = j \sec^2 \phi \sec \theta d\phi = j \sec \phi d\phi.$$

Thus

$$\begin{aligned} F(k, \theta) &= \int_0^\theta \frac{d\theta}{(1 - k^2 \sin^2 \theta)^{1/2}} = \int_0^\phi \frac{j \sec \phi d\phi}{(1 + k^2 \tan^2 \phi)^{1/2}} \\ &= j \int_0^\phi \frac{d\phi}{(1 - k'^2 \sin^2 \phi)^{1/2}} = jF(k', \phi) = ju, \end{aligned} \quad (2.9)$$

where $k' = (1 - k^2)^{1/2}$ is called the *complementary modulus of k* . Using the notation given in (2.5), we can write

$$\phi = am(u, k'), \quad (2.10)$$

$$\theta = am(ju, k).$$

Now we can write

$$\begin{aligned} \sin \theta &= j \frac{\sin \phi}{\cos \phi}, \\ sn(ju, k) &= j \frac{sn(u, k')}{cn(u, k')} = j tn(u, k'). \end{aligned} \quad (2.11)$$

Similarly, one can show that

$$\begin{aligned} cn(ju, k) &= 1/cn(u, k'), \\ dn(ju, k) &= \frac{dn(u, k')}{cn(u, k')}. \end{aligned} \quad (2.12)$$

Thus, the elliptic functions with imaginary argument are written in terms of those with real arguments, whose modulus is the complement of the original modulus.

In practice, it is convenient to express k as

$$k = \sin \theta,$$

and to specify the stopband frequency in terms of the modular angle θ (rad), as

$$\omega_s = 1/\sin \theta.$$

We shall return now to the discussion of the parameters in (2.3). The notation $sn^{-1}(u, k)$ denotes an inverse elliptic function which is defined as if $y = sn(u, k)$ then $u = sn^{-1}(y, k)$, where $sn(u, k)$ is the Jacobian elliptic sine function. It should be noted that the real parameters ϵ , k , and k_1 , are all bounded between 0 and 1, and are to be determined from the filter specifications. However, the filter order n , and the quantities k , and k_1 are related to each other by the equation

$$\frac{K'_1}{K'} = \frac{nK_1}{K}, \quad (2.13)$$

where

$$K'_1 = K(k'_1),$$

$$K' = K(k'),$$

and

$$k'_1 = (1 - k_1^2)^{1/2},$$

$$k' = (1 - k^2)^{1/2},$$

are the complementary moduli of k_1 and k , respectively. This restriction is required to insure that the function $F_n(\omega)$ is a rational function [1,7], which is dictated by the desire to obtain formulas that are simple and give the equiripple characteristic in both the passband and the stopband.

2.2. THE CHARACTERISTIC FUNCTION $F_n(\omega)$

The general behavior of the rational function $F_n(\omega)$ (i.e., the ratio of two polynomials in ω), is shown in Fig. 2-3. Some of its features can be summarized as follows[7]:

- (a) $F_n(\omega)$ has all its n zeros in the interval $|\omega| < 1$ and all its poles outside this interval;
- (b) $F_n(\omega)$ oscillates between the value ± 1 in the interval $|\omega| < 1$;
- (c) $F_n(1) = 1$;
- (d) $1/F_n(\omega)$ oscillates between $\pm 1/k_1$ in the interval $|\omega| > \omega_s$.

From these requirements, i.e., all points in the $|\omega| < 1$ and $|\omega| > \omega_s$ where $|F_n(\omega)| = 1$ and $1/k_1$, the function $F_n(\omega)$ must be maximum or minimum points

$$\left. \frac{dF_n}{d\omega} \right|_{|F_n(\omega)|=1} = 0 \quad \text{except at } |\omega| = 1 \quad (2.14)$$

$$\left. \frac{dF_n}{d\omega} \right|_{|F_n(\omega)|=1/k_1} = 0 \quad \text{except at } |\omega| = \omega_s.$$

Hence $F_n(\omega)$ satisfies the differential equation

$$\frac{dF_n(\omega)}{d\omega} = A \sqrt{\frac{(1 - F_n^2(\omega))(1 - k_1^2 F_n^2(\omega))}{(1 - \omega^2)(1 - \omega^2/\omega_s^2)}}, \quad (2.15)$$

where A is constant. Alternatively, we can write

$$\frac{dF_n(\omega)}{\sqrt{(1 - F_n^2(\omega))(1 - k_1^2 F_n^2(\omega))}} = A \frac{d\omega}{\sqrt{(1 - \omega^2)(1 - \omega^2/\omega_s^2)}},$$

or

$$\int_0^F \frac{dx}{\sqrt{(1 - x^2)(1 - k_1^2 x^2)}} = A \int_0^\omega \frac{dy}{\sqrt{(1 - y^2)(1 - k_1^2 y^2)}}.$$

These are elliptic integrals of the first kind, and they can be put in the more convenient form shown in (2.4).

The solution of this equation is given by (2.3) in terms of the complete elliptic integrals and the inverse sin-Jacobian function.

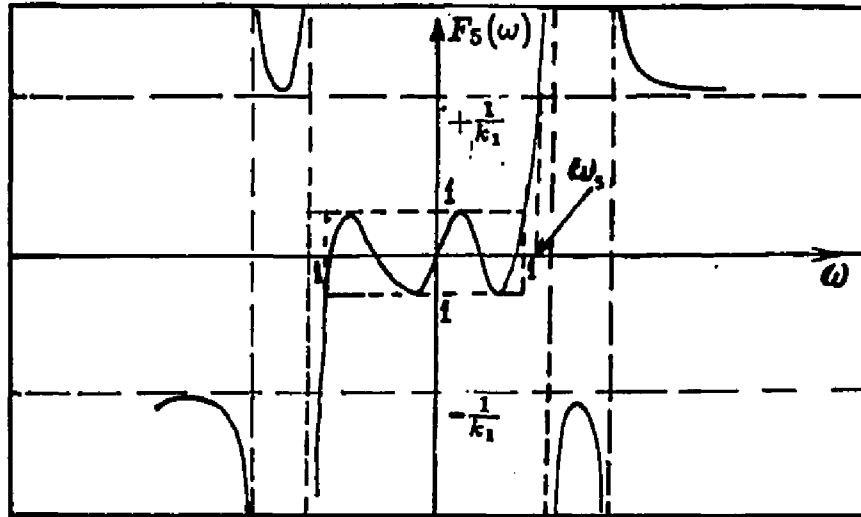


Fig 2-3 Typical plot of the behavior of $F_n(\omega)$, $n = 5$.

As in (2.3), two cases are to be considered, (n even or odd). These will be introduced separately next.

A. n odd

In this case, the normalized characteristic function $F_n(\omega)$ becomes

$$F_n(\omega) = H_o \prod_{m=1}^{(n-1)/2} \left\{ \frac{\omega(\omega_m^2 - \omega^2)}{1 - k^2 \omega_m^2 \omega^2} \right\}, \quad (2.16)$$

where $F_1(\omega) = H_o \omega$, and for $n > 1$,

$$\omega_m = sn(2mK/n, k), \quad m = 1, 2, \dots, q, \quad q = \frac{1}{2}(n-1) \quad (2.17)$$

and H_o is a real constant given by

$$H_o = \left(\frac{k^n}{k_1} \right)^{1/2}. \quad (2.18)$$

The quantity k_1 can be determined by the formula

$$k_1 = k^n \left[\frac{(1 - \omega_1^2)(1 - \omega_2^2) \cdots (1 - \omega_q^2)}{(1 - k^2 \omega_1^2)(1 - k^2 \omega_2^2) \cdots (1 - k^2 \omega_q^2)} \right]^2, \quad (2.19)$$

once k and n are known. The location for the distinct zeros of $F_n(\omega)$ are given by

$$\omega_{zm} = sn(2mK/n, k), \quad m = 0, \pm 1, \dots, \pm \frac{1}{2}(n-1). \quad (2.20)$$

Since $\omega_{zm} \leq 1$, all of the zeros lie within the passband. The location for the distinct pole of $F_n(\omega)$ are given by

$$\omega_{pm} = \frac{1}{k sn(2mK/n, k)}, \quad m = 0, \pm 1, \dots, \pm \frac{1}{2}(n-1) \quad (2.21)$$

For $m = 0$, the right-hand side of (2.21) is infinite, indicating that the function $F_n(\omega)$ has a pole at infinity. Since $sn(2mK/n, k) \leq 1$, all of the poles of $F_n(\omega)$ lie in the stopband.

B. n even

In this case, the normalized characteristic function becomes

$$F_n(\omega) = H_e \prod_{m=1}^{n/2} \left\{ \frac{(\omega_m^2 - \omega^2)}{(1 - k^2 \omega_m^2 \omega^2)} \right\}, \quad (2.22)$$

where

$$\omega_m = sn[(2m - 1)K/n, k], \quad m = 1, 2, \dots, \frac{1}{2}n, \quad (2.23)$$

and as in the case for n odd is real constant given by

$$H_e = \left(\frac{k^n}{k_1} \right)^{1/2} \quad (2.24)$$

and k_1 in this case can be determined by the formula

$$k_1 = k^n \left[\frac{(1 - \omega_1^2)(1 - \omega_2^2) \cdots (1 - k^2 \omega_{n/2}^2)}{(1 - k^2 \omega_1^2)(1 - k^2 \omega_2^2) \cdots (1 - k^2 \omega_{n/2}^2)} \right]^2, \quad (2.25)$$

once k and n are known. The location for the distinct zeros of $F_n(\omega)$ are given by

$$\omega_{zm} = sn[(2m - 1)K/n, k], \quad m = -\frac{1}{2}n + 1, -\frac{1}{2}n + 2, \dots, \frac{1}{2}n. \quad (2.26)$$

Again since $\omega_{zm} \leq 1$, all of the zeros lie within the passband. The location for the distinct pole of $F_n(\omega)$ are given by

$$\omega_{pm} = \frac{1}{k sn[(2m - 1)K/n, k]}, \quad m = -\frac{1}{2}n + 1, -\frac{1}{2}n + 2, \dots, \frac{1}{2}n - 1, \frac{1}{2}n. \quad (2.27)$$

Note that as $\omega \rightarrow \infty$, $F_n(\infty) = 1/k_1$, indicates that the function has neither a pole nor a zero at the infinity.

2.3. POLES AND ZEROS OF ELLIPTIC RESPONSE

To determine the poles and zeros of the elliptic transfer function, $N(s)$, we will replace ω by $-js$ and write $y = s/\omega_c$. The transducer power-gain characteristic is then given by

$$G(-s^2) = |N(s)|^2 = \frac{H_n}{1 + \epsilon^2 F_n^2(-jy)}, \quad (2.28)$$

where

$$F_n(-jy) = \operatorname{sn} \left[\frac{nK_1}{K} \operatorname{sn}^{-1}(-jy, k), k_1 \right] \quad (2.29)$$

for n odd, and

$$F_n(-jy) = \operatorname{sn} \left[K_1 + \frac{nK_1}{K} \operatorname{sn}^{-1}(-jy, k), k_1 \right] \quad (2.30)$$

for n even. The zeros of $G(-s^2)$ are the double poles of $F_n(-jy)$, which from (2.21) and (2.27) are located at

$$s_{zm} = j \frac{\omega_c}{k \operatorname{sn}[2mK/n, k]}, \quad m = 0, \pm 1, \dots, \pm \frac{1}{2}(n-1) \quad (2.31)$$

for n odd, and

$$s_{zm} = j \frac{\omega_c}{k \operatorname{sn}[(2m-1)K/n, k]}, \quad m = -\frac{1}{2}n + 1, -\frac{1}{2}n + 2, \dots, \frac{1}{2}n \quad (2.32)$$

for n even. Thus, all of its zeros are imaginary and lie on the segments $|\omega/\omega_c| > 1/k$.

The poles of $G(-s^2)$ are defined by the zeros of the equation

$$1 + \epsilon^2 F_n^2(-jy) = 0 \quad (2.33a)$$

or

$$F_n(-jy) = \pm j/\epsilon. \quad (2.33b)$$

There are two solutions for this equation, depend on whether n is even or odd. For n odd, the location of the poles of the transfer function are given by [1]

$$y_{pm} = jsn(2mK/n + ja, k), \quad m = 0, \pm 1, \dots, + \pm \frac{1}{2}(n-1) \quad (2.34)$$

and for n even,

$$y_{pm} = jsn\left[(2m-1)K/n + ja, k\right], \quad \text{for } m = -\frac{1}{2}n + 1, -\frac{1}{2}n + 2, \dots, \frac{1}{2}n \quad (2.35)$$

where

$$a = -j(K/nK_1)sn^{-1}(j/\epsilon, k_1) \quad (2.36)$$

is real. The denominator of $G(-s^2)$, apart from a constant, can be formed as

$$E(y)E(-y) = \begin{cases} \prod_{m=0}^{(n-1)/2} (y^2 - y_{pm}^2), & \text{if } n \text{ odd;} \\ \prod_{m=-n/2+1}^{n/2} (y^2 - y_{pm}^2), & \text{if } n \text{ even.} \end{cases} \quad (2.37)$$

where

$$E(y) = c_0 + c_1y + \dots + c_{n-1}y^{n-1} + c_ny^n = \sum_{m=0}^n c_my^m \quad (2.38)$$

with $c_n = 1$, is the Hurwitz polynomial of degree n formed by the LHS (left hand side) zeros of (2.33), which are given by (2.34) and (2.35).

In order to calculate the the Jacobian elliptic sine functions in (2.34) and (2.35) with the complex argument, we shall apply Jacobi's imaginary transformation $sn(ju, k) = jtn(u, k)$ to express them in term of functions of real arguments.

Doing so, we get[1]

$$sn(u, \pm jv, k) = \frac{sn(u, k)dn(v, k') \pm j cn(u, k)dn(u, k)sn(v, k')cn(v, k')}{cn^2(v, k') + k^2 sn^2(u, k)sn^2(v, k')}. \quad (2.39)$$

We note also that to evaluate the parameter a in Eq. (2.36), we must calculate $sn^{-1}(j/\epsilon, k_1)$. If we let

$$ju = sn^{-1}(j/\epsilon, k_1),$$

we would have

$$\begin{aligned} sn(ju, k_1) &= j/\epsilon \\ &= j tn(u, k'_1) \end{aligned}$$

or

$$tn(u, k'_1) = 1/\epsilon.$$

Now, using the identity

$$sn^2(u, k'_1) + cn^2(u, k'_1) = 1,$$

we can write

$$cn^2(u, k'_1) = \frac{\epsilon^2}{1 + \epsilon^2}.$$

Thus,

$$sn(u, k'_1) = (1 + \epsilon^2)^{-1/2} = \sin \phi,$$

or

$$u = F(\phi, k'_1). \quad (2.40)$$

CHAPTER 3

SYNTHESIS OF *LC* DOUBLY-TERMINATED NETWORKS

Synthesis is one of the most important subjects in modern electrical engineering. In network synthesis, we are primarily concerned with the design of networks that will meet a prescribed excitation-response characteristics. The subject of filter synthesis will not be treated here, (refer to [2 & 6] for more detail on the subject). Only some important relations are stated for *LC* networks.

Having obtained the gain function, (*cf.* Ch. 2), one can design the lossless two-port network to meet this gain characteristic, i.e.,

$$|S_{21}(j\omega)|^2 = G(\omega^2) = \frac{H_n}{1 + \epsilon^2 F_n^2(\omega)} \quad (3.1)$$

H_n as before being bounded between 0 and 1, and S_{21} is known as the transmission coefficient. The input reflection coefficient S_{11} is defined such that

$$|S_{11}(j\omega)|^2 + |S_{21}(j\omega)|^2 = 1 \quad (3.2)$$

Therefore, letting $s = j\omega$ and $y = s/w_c$, we can write

$$\begin{aligned} S_{11}(s)S_{11}(-s) &= 1 - G(-s^2) \\ &= (1 - H_n) \frac{(1 + \hat{\epsilon}^2 F_n(-jy))}{(1 + \epsilon^2 F_n^2(-jy))}, \end{aligned} \quad (3.3)$$

where

$$\hat{\epsilon} = (1 - H_n)^{-1/2} \quad (3.4)$$

Since S_{11} is devoid of poles in the closed right half plane, RHP (the right-half of the s -plane, including the $j\omega$ -axis), we must assign all of the left half plane (LHP) of (3.3) to $S_{11}(s)$, resulting in a unique decomposition of the denominator polynomial. However, the zeros of $S_{11}(s)$ may lie in the RHP, so that in general a number of different numerator are possible for $S_{11}(s)$. Unless the $S_{11}(s)$ is restricted to be a minimum phase, (i.e., the zeros can only occur in the LHP or on the $j\omega$ -axis), a unique decomposition is not permissible. Thus, the minimum-phase decomposition of (3.3) can be written as

$$S_{11}(s) = \pm \frac{R(y)}{E(y)} \quad (3.5)$$

with

$$R(y) = \sum_{m=0}^n b_m y^m,$$

$b_n = 1$, being the Hurwitz polynomial formed by the LHP zeros of the equation $1 + \hat{\epsilon}^2 F_n^2(-jy) = 0$, (or the zeros of $F_n(\omega)$, ω_{xm} , if $H_n = 1$), and $E(y)$ is a polynomial formed by the LHP poles of the transfer function, $N(s)$, which is given by Eq. (2.37). Then $S_{11}(s)$ is a bounded-real reflection coefficient.

Denoting the even and odd parts of the polynomials $R(s)$ and $E(s)$ by the subscripts e and o , we can write

$$R(s) = R_e + R_o, \quad (3.6a)$$

and

$$E(s) = E_e + E_o. \quad (3.6b)$$

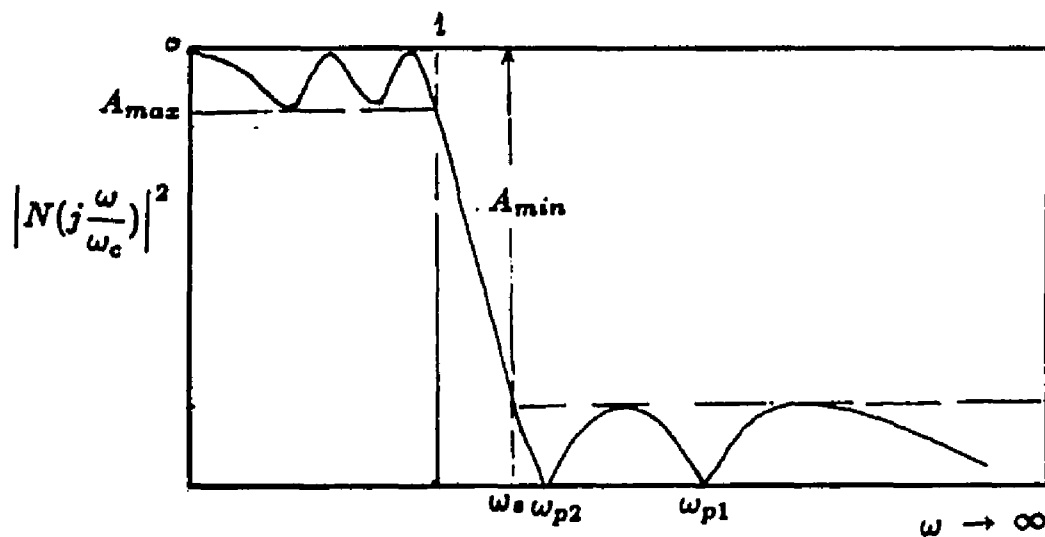
The input impedance can be formed by using Eq. (3.5) as follows

$$Z_{in} = R_1 \frac{R_e + E_e}{R_o - E_o} \quad (3.7)$$

or according to one of the formulas in Table 3-1.

Table 3-1 Design Impedance Equations for Doubly Terminating Filters [9].

(n even)	(n odd)
$Z_{11} = R_1 \left[\frac{R_e(s) - E_e(s)}{R_o(s) + E_o(s)} \right]$	$Z_{11} = R_1 \left[\frac{R_o(s) - E_o(s)}{R_e(s) + E_e(s)} \right]$
$Z_{22} = R_2 \left[\frac{R_e(s) + E_e(s)}{R_o(s) + E_o(s)} \right]$	$Z_{22} = R_2 \left[\frac{R_o(s) + E_o(s)}{R_e(s) + E_e(s)} \right]$
$Y_{11} = R_1 \left[\frac{R_o(s) - E_o(s)}{R_e(s) + E_e(s)} \right]$	$Y_{11} = R_1 \left[\frac{R_e(s) - E_e(s)}{R_o(s) + E_o(s)} \right]$
$Y_{22} = R_2 \left[\frac{R_o(s) - E_o(s)}{R_e(s) - E_e(s)} \right]$	$Y_{22} = R_2 \left[\frac{R_e(s) - E_e(s)}{R_o(s) - E_o(s)} \right]$

Fig 3-1 Frequency response for odd network lowpass filter, $n = 5$.

Once these immittances are evaluated, the element values of the filter circuits, can be evaluated by successive complete and partial-pole removals from these reactances corresponding to transmission zeros, ω_{pi} , of the transfer function $N(s)$.

For n odd, a ladder network can be realized with equal terminating resistances ($R_1 = R_2$). The frequency response for n odd is shown in Fig. 3-1. The modular angle of the Jacobian elliptic functions, is related to the stopband frequency ω_s by

$$\theta = \frac{180}{\pi} \arcsin\left(\frac{1}{\omega_s}\right). \quad (3.8)$$

For even n , the frequency response is shown in Fig. 3-2(a). This lowpass filter is termed case a and Eq. (3.8) also holds. Because of the finite non-zero attenuation, $|N(j\omega)| \neq 0$, at $\rightarrow \infty$, these filters cannot be realized as LC circuits without coupled coils [2,4]. To avoid the use of coupled coils, a frequency transformation is required to shift the highest transmission zero, ω_{p1} , (i.e., the 1st pole ω_{p1} of $F_n(\omega)$) to infinity. This will result in an LC ladder network with unequal terminating resistances, ($R_1 \neq R_2$), and the filter is termed case b. The frequency response for this case is shown in Fig. 3-2(b). The following frequency transformation is used

$$\tilde{S}^2 = \frac{(\omega_{p1}^2 - 1)S^2}{\omega_{p1}^2 + S^2}, \quad (3.9)$$

where S and \tilde{S} are the complex frequencies of cases a and b, respectively, and ω_{p1} is the highest transmission zero frequency of the lowpass filter case a. Obviously, the stopband frequency ω_s (case a) has been shifted to a new value, ω'_s , given by $\omega'_s = \omega_s/\omega'_c$, where ω'_c is the new cut-off frequency that is given by $\omega'_c = \omega_{zn}/2$.

Elliptic function lowpass filters

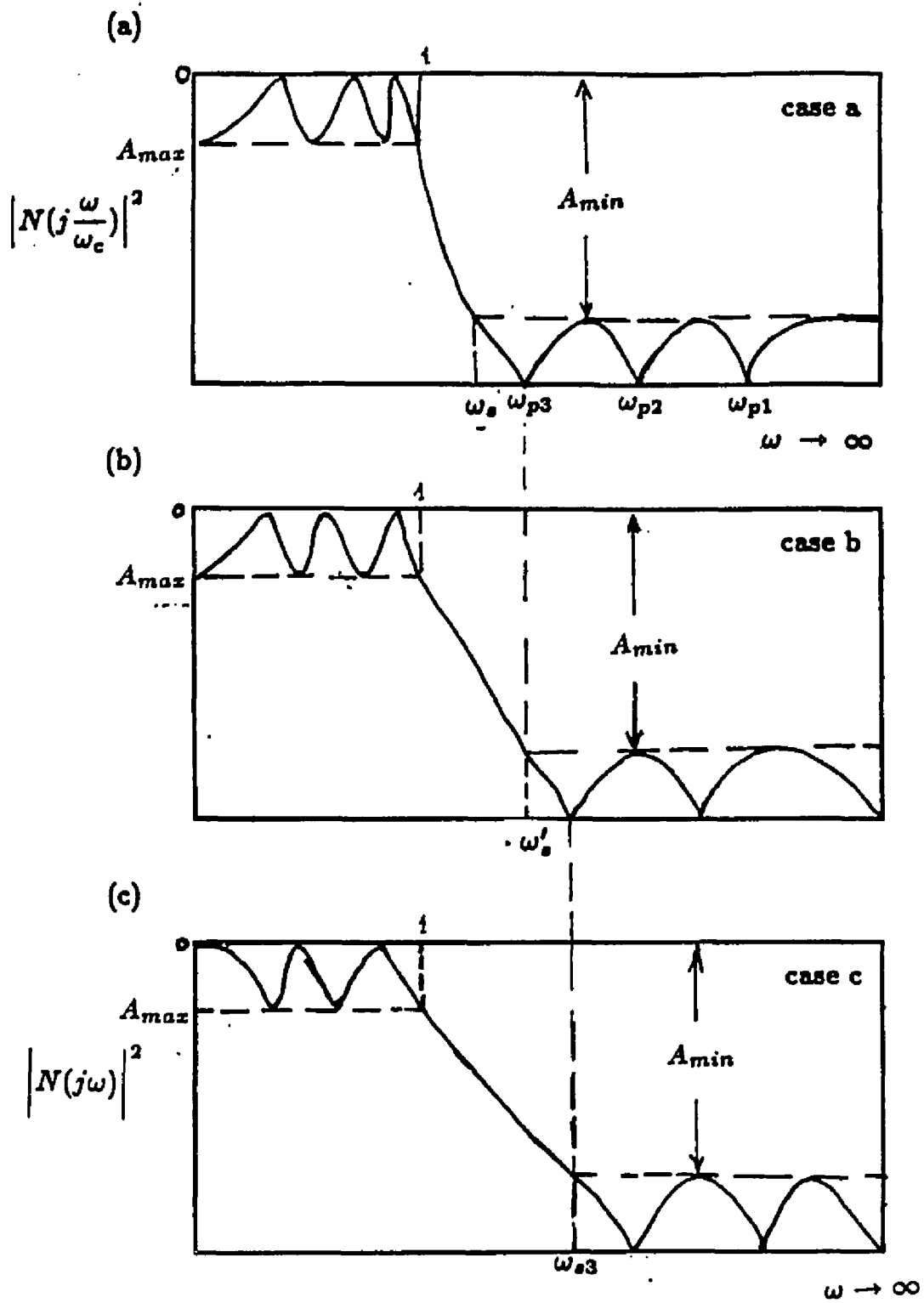


Fig. 3.2 Frequency responses of elliptic lowpass filters, case a, case b and case c of degree $n=6$, ($\omega_{s3} > \omega'_s > \omega_s$).

In addition, if the terminating resistances are to be equal, i.e., $R_2 = R_1$, in accordance with odd degree filters, a second frequency transformation is required to shift the first zero, (ω_{x1} of $F_n(\omega)$), to the origin. This lowpass filter type is termed case c. The transformation for this case is given by

$$\tilde{S}^2 = \frac{\tilde{S}^2 + \omega_{x1}^2}{1 - \omega_{x1}^2} \quad (3.10)$$

which gives zero attenuation at $\omega = 0$. Here, \tilde{S} and \tilde{S} are the complex frequencies of the filters case b, and case c, respectively, and ω_{x1} is the zero next to the origin. As in case b, the stopband frequency, ω_s (case a), is shifted to a new position given by $\omega_{s3} = \omega_s/\omega_c^2$. The frequency response for even Case c is shown in Fig. 3-2(c).

LC realization procedure for circuits without coupled coils

To realize filter circuits as ladder network, we will use the successive complete and partial-pole removal method. The realization procedures are outlined in Table 3-2, considering the lowpass filter circuit Fig. 3-3 as an example.

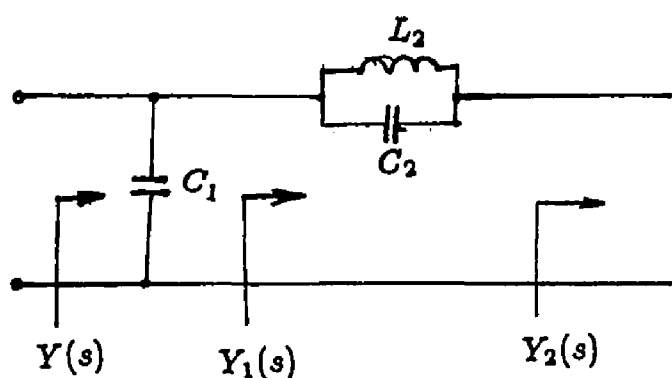


Fig. 3-3 Elliptic lowpass ladder network.

Table 3-2. Element values and remainder functions of lowpass filters.

The pole removal process starts from

$$Y(s) = \frac{N(s)}{M(s)}$$

by partial pole removal at $s \rightarrow \infty$ for the first transmission zero ω_{p1} . This yields the value of the shunt capacitor C_1 :

$$C_1 = \frac{Y(s)}{s} \Big|_{s^2 = -\omega_{p1}^2} = \frac{N(s)}{sM(s)} \Big|_{s^2 = -\omega_{p1}^2}$$

The remainder function, $Y_1(s)$, is given by

$$Y_1(s) = Y(s) - C_1 s = \frac{N(s)}{M(s)} - C_1 s = \frac{N_1(s)(s^2 + \omega_{p1}^2)}{M(s)}$$

The complete pole removal at $s^2 = -\omega_{p1}^2$ yields the value of C_2 and L_2 as follows:

$$C_2 = \frac{sY_1(s)}{s^2 + \omega_{p1}^2} \Big|_{s^2 = -\omega_{p1}^2}, \quad L_2 = \frac{1}{C_2 \omega_{p1}^2}$$

The remainder function, $Y_2(s)$, is calculated as follows:

$$\frac{1}{Y_2(s)} = \frac{1}{Y_1(s)} - \frac{\frac{1}{C_2} s}{s^2 + \omega_{p1}^2} = \frac{M(s) - \frac{1}{C_2} s N_1(s)}{N_1(s)(s^2 + \omega_{p1}^2)} = \frac{M_1(s)}{N_1(s)}$$

Thus

$$Y_2(s) = \frac{N_1(s)}{M_1(s)}$$

Obviously, the degree of the reactance has been reduced by two. These steps are repeated for ω_{p2} , ω_{p3} , and so on.

It is important to notice that we have used in Table 3-2 repeatedly division, subtraction, and multiplication of different orders polynomials. As a result, we may anticipate round off errors in our calculation. To avoid the accumulation of these errors, the calculations of the circuit-element values are performed from both ports of the ladder circuit, and stop when we reach the central branch of the ladder.

The values of the circuit-elements are in general dependent on the sequence of the transmission zeros, i.e., they depend on the way we start the substitution of these zeros in the steps outlined in Table 3-2. For example, the transmission of zeros of the transfer function of $n = 10$ is given by

$$\begin{aligned}\omega_{p1} &= 2.1122466606 & \omega_{p2} &= 1.4397415579 \\ \omega_{p3} &= 1.2358587261 & \omega_{p4} &= 1.1687177388\end{aligned}$$

(see output in App. A. Note that $wz1 = \omega_{p1}$, $wz2 = \omega_{p2}$, and so on.) These were printed by the following loop

For i := 1 to n div 2 do write ('wz',i, $\omega_p[i]$);

As one might see, we could have started the substitution in any of the following sequences, say,

$$\{\omega_{p1}, \omega_{p2}, \omega_{p3}, \omega_{p4}\} \quad \text{OR} \quad , \{\omega_{p3}, \omega_{p1}, \omega_{p2}, \omega_{p4}\}, \dots, \text{etc.}$$

The result from each sequence is different, and we may end up with negative element values. Thus, we must follow a certain rule for guidance to avoid these negative values. The rule that we shall follow is that we shall choose the sequence that (1) makes the transmission properties of the circuit as an insensitive as possible with respect to element variations [8], (2) makes the element values centered about unity, and (3) the sequence must be chosen in order to get suitable positive element values, up to the largest possible positive value of θ (or the largest possible range

of frequency in the passband). The sequence that has been used in our program is $\{\omega_{p1}, \omega_{p3}, \omega_{p4}, \omega_{p2}\}$. The Procedure "Zero-Seq()" is capable of reordering such a sequence.

CHAPTER 4

USER GUIDE: DETAILED PROGRAM DESCRIPTION

PC-ELLIP is a self-instructional program. It is written in Turbo Pascal to be used interactively on Microcomputers. It allows the user to synthesize Low-pass normalized elliptic filters as passive ladder structures. Most of the procedures and functions are written with notion that follows the theoretical treatment developed in Chapter 2 and Chapter 3. The program is written to utilize an efficient algorithm and to incorporate all the user friendly input/output characteristics, and error-handling capabilities. Procedures and Functions names are chosen as to represent what they stand for. Some of them are listed below:

- | | |
|------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Procedure EvenFn() | This procedure calculates the zeros of the even characteristic function $F_n(\omega)$. |
| Procedure EvenRoots() | This procedure calculates the roots (poles) of the transfer function when the filter order is even. |
| Procedure OddFn() | This procedure performs the same function as the EvenFn() except that the filter order is odd. |
| Procedure OddRoots() | This finds the poles of the transfer function, $N(s)$, when the filter order is odd. |
| Function Phi() | This function returns the value of the angle ϕ that will be used to calculate the Jacobi's elliptic functions. |
| Function Fk1() | This function returns the value of k_1 . |

- Function F()** This function calculates the complete elliptic integral of the first kind.
- Procedure Ellip_Coef()** This procedure returns the coefficient values of the Numerator or Denominator Polynomials of the transfer function, and the coefficients of the quadratic polynomial ($s^2 + a_1s + a_2$).
- Function ArcSn()** This function returns the value of the elliptic inverse sine function, $sn^{-1}(u, v)$.
- Procedure Jac_Fun()** This procedure calculates the Jacobi's elliptic functions.
- Calculate_Max_Order()** This procedure calculates the filter order, n .
- Procedure Even_NetWork**
This procedure calculates the element values for even networks.
- Procedure Odd_NetWork**
This procedure calculates the element values for odd networks.
- Print_Out_To_Dev()** This procedure prints the output into a device (file or printer).

How to Run the Program

To run the program simply type **PC-ELLIP**

The user will be asked if he would like to run the program on a monochrome or color monitor. The respond is yes (for color) or no (for monochrome). A new window will be displayed and offers the following descriptions:

This Program will find the Elliptic Network Function, and its Ladder Realization. The Elliptic network function is described by the following parameters:

1. The ripples in the passband, A_{max} ;
2. The minimum attenuation in the stopband, A_{min} ;
3. The order of the filter, n ;
4. The stopband frequency, ω_s ;
5. The case for even order functions, (A, B, or C).

With any two of the first three parameters and the stopband frequency, one can determine completely the Elliptic Network Transfer Function according to the following three possible combinations:

- 1) Combination A (The order n is unknown),
- 2) Combination B (A_{min} is unknown),
- 3) Combination C (A_{max} is unknown).

At the bottom of the screen you will notice the line command

Enter the option (1, 2, 3, or H(elp)).

Hit H(or h) for help on the definitions of the above combinations. If help is needed a new window will show on the top of the main window with the following descriptions:

Combination A

- 1) Stopband frequency (rad/s), or the Modular angle θ (in deg.)
- 2) Ripples in the Passband (A_{max} (db), or the ref. coef, R_o);
- 3) Attenuation in the stopband (A_{min} (db)).

Combination B

- 1) Stopband frequency (ω_s);
- 2) Ripples in the Passband A_{max} (dB);
- 3) The order of the filter (n).

Combination C

- 1) Stopband frequency ω_s .
- 2) Attenuation in the stopband, A_{min} (dB);
- 3) The order of the filter (n).

To get out of the help window, hit any key.

Now select any of the options (1, 2, or 3). For any of these three options the following window will show on the screen.

Stopband frequency:

- 1) ω_s (r/s)
- 2) Angle (deg)

At this stage, two options for the stopband frequency are shown. Choose 1 if you want to use the normalized stopband frequency, ω_s , or 2, if the modular angle θ is preferred. Note: Nulls are not acceptable values.

Depending on which combination is used, the passband ripples or the stopband attenuation will be prompted. (see examples in App. A). When the three values (say ω_s , A_{max} , and the order, n) are specified, a new window will show to the right of the input window and with all inputs that the user has specified. This makes it convenient for the user to correct or to modify the given values.

1) Angle =
 2) Ro =
 3) Max_order =
 Is it OK ? (Y/N):

Check your input to see if they were entered correctly. If the answer to the above question is N(o), you will be asked to enter a number corresponding to a value that is wrong or that you want to modify. If your answer is Y(es), (i.e. the input values are correct) a new fresh screen with a window that has the following command

Enter (1, 2, 3) for Even Cases (A, B, C) or Q(uit)=>

Note: The above command will appear only if the order is even.

For the even case A, (no shifting to the poles or zeros), the element values are not calculated for a passive ladder network, because the realization requires the

use of coupling coils. For case B, the right most transmission zero of $N(s)$, (i.e the right most pole of the characteristic function, $F_n(\omega)$), is shifted to infinity. This makes a passive ladder network with unequal terminating resistances realizable. For case C, the first zero, (next to the origin), of the characteristic function, $F_n(\omega)$, is shifted to the origin. The right most pole is also shifted to infinity. In this case a passive ladder network is realizable with equal resistances.

The last prompt is:

The output can be directed to:

1. Screen
2. Text File
3. Printer

Enter 1, 2, 3 =>

This is a convenient way to show the output on the screen, save it into a file, or even send it to the printer.

CHAPTER 5

SUMMARY AND RECOMMENDATIONS

This thesis presents an instructional program, PC-ELLIP, that can be used to synthesize Low-pass normalized elliptic filters as passive ladder structures. The mathematical model for the program, PC-ELLIP, was developed in Chapter 2. Explicit analytical expression for equiripple rational function, $F_n(\omega)$, poles and zeros of this rational function were given in terms of the Jacobian elliptic functions. The basic properties of the Jacobian elliptic functions were stated. The rational function, $F_n(\omega)$ is then used to construct the transfer function, $N(s)$, with equiripple properties in the passband and stopband regions. Formulas were derived to calculate the pole and zero locations of the filter transfer function.

Chapter 3 outlined the basic rules of partial and complete pole removal techniques that were used in PC-ELLIP (see Table 3-2). Special considerations were given when the filter order is even in order to make the filter realizable without transformers. The study was concentrated on the synthesis of two-port networks with double terminating resistances. No attempt was made to realize networks with single terminating resistances.

RECOMMENDATIONS

Due to the fact that elliptic filters provide the optimum amplitude responses for a given degree, the normalized design low-pass element values are available in the form of an extensive sets of tables (see for example references 2, 4 and 5).

Reference 4 and 5 give the specifications in terms of the reflection coefficient, ρ , (for the ripples in the passband, A_{max}) and the modular angle θ (for the stopband frequency, ω_s). Reference 2, however, gives the specifications in terms of the physical quantities, the frequency ω_s and the maximum ripples in the passband, A_{max} . The program, PC-ELLIP, combines the two methods and gives the user the options to choose one (see Chapter 4).

Mentioned in Chapter 3 was the fact that as we transform the even Case A to Case B or Case C the stopband frequency $\omega_s(\text{case a}) < \omega_s(\text{case b}) < \omega_s(\text{case c})$. For this reason, PC-ELLIP and Ref. (4 & 5) report the design values for Case B and Case C at the shifted stopband frequency ω_s (the attenuation is at the input frequency, i.e. A_{min} is that of Case A). However, Ref. 2 reports the design values for these two cases at a prescribed stopband frequency, the input frequency, (A_{min} is not that of Case A). This is a desirable feature for some application. The author would like to see such an addition to be implemented in the program PC-ELLIP. Also the addition of some procedures to the program to calculate the element values for networks with single terminated resistance will add to the effectiveness of the program.

APPENDIX A
INPUT/OUTPUT EXAMPLES

Table A-1 Third Order Filter.

InputTheta 30 Ro = 20 % n = 3

Output

n = 3 Ws = 2.00000 r/s
A_max = 0.17729 db A_min = 26.52839 dbActual attenuation in the stopband = 26.5283866470 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-9.5125175317E-01		
-3.4364160633E-01 + j	1.1453842929E+00	6.8728321267E-01
		1.4299947321E+00

Coefficients of the polynomial r(s) for the DENOMINATOR
Associated with the Elliptic Response.r(s) = a0 + a1*s + a2*s**2 + ... + a_(n-1)*s**(n-1) + s**n
-----a0 = 1.3602849959E+00 a1 = 2.0837740931E+00 a2 = 1.6385349658E+00
a3 = 1.0000000000E+00

Zeros of the transfer function

Wz1 = 2.2700680863E+00

Quadratic factors of $s^{**2} + a1*s + a2$	NUMERATOR	
	a1	a2
	0.0000000000E+00	5.1532091165E+00

Coefficients of the polynomial r(s) for the NUMERATOR
Associated with the Elliptic Response.r(s) = a0 + a1*s + a2*s**2 + ... + a_(n-1)*s**(n-1) + s**n
-----a0 = 5.1532091165E+00 a1 = 0.0000000000E+00 a2 = 1.0000000000E+00
a3 = 0.0000000000E+00
-----R1 = 1.000 R2 = 1.0000000000E+00 Const. H = 2.6396852237E-01

k	C(2k-1)	L(2k)	C(2k)
1	1.0512464141E+00	9.6123884859E-01	2.0187889505E-01
2	1.0512464373E+00		

Theat = 30.0 Ro = 20.00 % Case Odd

Table A-2 Third order Filter.

Input

Ws = 1.2 rad/s A_max = 0.01 % n = 3

Output

n = 3 Ws = 1.20000 r/s
 A_max= 0.01000 db A_min=1.34497 db

Actual attenuation in the stopband = 1.3449720881 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-4.7153366456E+00		
-8.3964162232E-02 +j	1.2775125330E+00	1.6792832446E-01
		1.6390882526E+00

 Coefficients of the polynomial r(s) for the DENOMINATOR
 Associated with the Elliptic Respnse.

$r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{*(n-1)} + s^{**n}$

 a0 = 7.7288529027E+00 a1 = 2.4309268347E+00 a2 = 4.8832649701E+00
 a3 = 1.0000000000E+00

Zeros of the transfer function

Wz1= 1.3036936410E+00

Quadratic factors of $s^{**2} + a1*s + a2$	NUMERATOR	
	a1	a2
	0.0000000000E+00	1.6996171097E+00

 Coefficients of the polynomial r(s) for the NUMERATOR
 Associated with the Elliptic Respnse.

$r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{*(n-1)} + s^{**n}$

 a0 = 1.6996171097E+00 a1 = 0.0000000000E+00 a2 = 1.0000000000E+00
 a3 = 0.0000000000E+00

 R1 = 1.000 R2 = 1.0000000000E+00 Const. H = 4.5474082713E+00

k	C(2k-1)	L(2k)	C(2k)
1	2.1207396424E-01	2.0490465431E-01	2.8714223888E+00
2	2.1207400416E-01		

 Theat =56.4 Ro =4.80 % Case Odd

Table A-3 Fourth Order filter Case B.

 Input

Theta 30 Ro = 15 % n=4 case b

 Output

n = 4 Ws = 2.14319 r/s
 A_max= 0.09883 db A_min=41.39555 db

Actual attenuation in the stopband = 41.3955457940 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-6.6349116557E-01 +j 5.1349285442E-01	1.3269823311E+00	7.0389543833E-01
-2.2943640930E-01 +j 1.1200835200E+00	4.5887281860E-01	1.3072281577E+00

 Coefficients of the polynomial r(s) for the DENOMINATOR
 Associated with the Elliptic Respse.

$r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$

 a0 = 9.2015193705E-01 a1 = 2.0576671518E+00 a2 = 2.6200397185E+00
 a3 = 1.7858551497E+00 a4 = 1.0000000000E+00

 Zeros of the transfer function

Wz1= 2.3310696140E+00

Quadratic factors of $s^{**2} + a1*s + a2$	NUMERATOR	
	a1	a2
	0.0000000000E+00	5.4338855451E+00
	0.0000000000E+00	0.0000000000E+00

 Coefficients of the polynomial r(s) for the NUMERATOR
 Associated with the Elliptic Respse.

$r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$

 a0 = 5.4338855451E+00 a1 = 0.0000000000E+00 a2 = 1.0000000000E+00
 a3 = 0.0000000000E+00 a4 = 0.0000000000E+00

 R1 = 1.000 R2 = 7.3913043478E-01 Const. H = 1.6933590695E-01

k	C(2k-1)	L(2k)	C(2k)
1	9.7000294753E-01	1.1167820754E+00	1.6478629094E-01
2	1.6608555400E+00	8.2775810659E-01	

 Theat =30.0 Ro =15.00 % Case B

Table A-4 Fourth Order Filter Case C.

InputTh_ta 30 Ro = 15 % n= 4 case c

Output

n = 4 Ws = 2.29663 r/s

A_{max} = 0.09883 db A_{min} = 41.39555 dbActual attenuation in the stopband = 41.3955457930 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-8.0726069659E-01 +j 5.0205746535E-01	1.6145213932E+00	9.0373153078E-01
-2.6694831328E-01 +j 1.1452050417E+00	5.3389662656E-01	1.3827559894E+00

Coefficients of the polynomial r(s) for the DENOMINATOR
Associated with the Elliptic Respnse.r(s) = a0 + a1*s + a2*s**2 + ... + a_(n-1)*s**(n-1) + s**n

a0 = 1.2496401870E+00	a1 = 2.7149883420E+00	a2 = 3.1484750455E+00
a3 = 2.1484180197E+00	a4 = 1.0000000000E+00	

Zeros of the transfer functionWz1 = 2.5048973161E+00
-----Quadratic factors of $s^{**2} + a1*s + a2$ NUMERATOR

a1	a2
0.0000000000E+00	6.2745105644E+00
0.0000000000E+00	0.0000000000E+00

Coefficients of the polynomial r(s) for the NUMERATOR
Associated with the Elliptic Respnse.r(s) = a0 + a1*s + a2*s**2 + ... + a_(n-1)*s**(n-1) + s**n

a0 = 6.2745105644E+00	a1 = 0.0000000000E+00	a2 = 1.0000000000E+00
a3 = 0.0000000000E+00	a4 = 0.0000000000E+00	

R1 = 1.000 R2 = 1.0000000000E+00 Const. H = 1.9916138066E-01

k	C(2k-1)	L(2k)	C(2k)
1	8.1364160082E-01	1.2416985704E+00	1.2835239122E-01
2	1.3589745251E+00	9.3091753170E-01	

Theat = 30.0 Ro = 15.00 % Case C

Table A-5 Fifth Order Filter.

InputTheta 30 Ro = 15 % n= 5

Output

n = 5 Ws = 2.00000 r/s
A_max= 0.09883 db A_min=58.84918 dbActual attenuation in the stopband = 58.8491753230 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$		
	a1	a2	
-5.9246078737E-01			
-4.3012950071E-01 +j	7.2184271023E-01	8.6025900143E-01	7.0606828570E-01
-1.3921820336E-01 +j	1.0740976896E+00	2.7843640671E-01	1.1730675549E+00

Coefficients of the polynomial r(s) for the DENOMINATOR
Associated with the Elliptic Response.r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n
-----a0 = 4.9071500654E-01 a1 = 1.5426177134E+00 a2 = 2.4609619464E+00
a3 = 2.7932956439E+00 a4 = 1.7311561955E+00 a5 = 1.0000000000E+00

Zeros of the transfer function

Wz1= 3.2508048748E+00 Wz2= 2.0892465023E+00

Quadratic factors of $s^{**2} + a1*s + a2$	NUMERATOR	
	a1	a2
	0.0000000000E+00	1.0567732334E+01
	0.0000000000E+00	4.3649509474E+00

Coefficients of the polynomial r(s) for the NUMERATOR
Associated with the Elliptic Response.r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n
-----a0 = 4.6127633263E+01 a1 = 0.0000000000E+00 a2 = 1.4932683281E+01
a3 = 0.0000000000E+00 a4 = 1.0000000000E+00 a5 = 0.0000000000E+00

Table A-5 -- Continued.

R1 = 1.000 R2 = 1.0000000000E+00 Const. H = 1.0638200398E-02			
k	C(2k-1)	L(2k)	C(2k)
1	1.0847737527E+00	1.2934482959E+00	7.3159229404E-02
2	1.7912862798E+00	1.1433104343E+00	2.0038096575E-01
3	9.7440587638E-01		

Theat =30.0 Ro =15.00 % Case Odd

Table A-6 Sixth Order Filter Case A.

InputTheta = 30 Ro = 25 % n=6 case a

Output

n = 6 Ws = 2.00000 r/s

A_max= 0.28029 db A_min=80.92154 db

Actual attenuation in the stopband = 80.9215419030 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-3.6052317525E-01 +j 3.0012643498E-01	7.2104635051E-01	2.2005283687E-01
-2.3622818741E-01 +j 7.8195822062E-01	4.7245637483E-01	6.6726241532E-01
-7.7633665334E-02 +j 1.0218395551E+00	1.5526733067E-01	1.0501830623E+00

Coefficients of the polynomial r(s) for the DENOMINATOR

Associated with the Elliptic Respse.

r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n

a0 = 1.5420151638E-01	a1 = 6.3725259410E-01	a2 = 1.5272806407E+00
a3 = 2.0291537387E+00	a4 = 2.4634732416E+00	a5 = 1.3487700560E+00
a6 = 1.0000000000E+00		

Zeros of the transfer function

Wz1= 7.2358027092E+00 Wz2= 2.7320508076E+00

Wz3= 2.0611053269E+00
-----Quadratic factors of $s^{**2} + a1*s + a2$ NUMERATOR

a1	a2
0.0000000000E+00	5.2356840847E+01
0.0000000000E+00	7.4641016153E+00
0.0000000000E+00	4.2481551688E+00

Coefficients of the polynomial r(s) for the NUMERATOR

Associated with the Elliptic Respse.

r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n

a0 = 1.6601653623E+03	a1 = 0.0000000000E+00	a2 = 6.4492542625E+02
a3 = 0.0000000000E+00	a4 = 6.4069097631E+01	a5 = 0.0000000000E+00
a6 = 1.0000000000E+00		

Table A-7 Sixth Order Filter Case B.

InputTheta = 30 Ro = 25 % n =6 case b

Output

n = 6 Ws = 2.06111 r/s

A_max= 0.28029 db A_min=80.92154 db

Actual attenuation in the stopband = 80.9215419030 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-3.5753896182E-01 +j 2.9639428313E-01	7.1507792364E-01	2.1568368029E-01
-2.3798470110E-01 +j 7.7774363852E-01	4.7596940221E-01	6.6152188522E-01
-7.9242372151E-02 +j 1.0220953760E+00	1.5848474430E-01	1.0509583111E+00

Coefficients of the polynomial r(s) for the DENOMINATOR

Associated with the Elliptic Respse.

 $r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$

a0 = 1.4995017986E-01 a1 = 6.2764767328E-01 a2 = 1.5135245271E+00

a3 = 2.0204044230E+00 a4 = 2.4572819193E+00 a5 = 1.3495320702E+00

a6 = 1.0000000000E+00

Zeros of the transfer function

Wz1= 2.9221322621E+00 Wz2= 2.1295487674E+00
-----Quadratic factors of $s^{**2} + a1*s + a2$ NUMERATOR

a1	a2
0.0000000000E+00	8.5388569572E+00
0.0000000000E+00	4.5349779529E+00
0.0000000000E+00	0.0000000000E+00

Coefficients of the polynomial r(s) for the NUMERATOR

Associated with the Elliptic Respse.

 $r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$

a0 = 3.8723528044E+01 a1 = 0.0000000000E+00 a2 = 1.3073834910E+01

a3 = 0.0000000000E+00 a4 = 1.0000000000E+00 a5 = 0.0000000000E+00

a6 = 0.0000000000E+00

Table A-7 -- Continued.

R1 = 1.000 R2 = 6.0000000000E-01 Const. H = 3.8723274307E-03			
k	C(2k-1)	L(2k)	C(2k)
1	1.3908870545E+00	1.2343229013E+00	9.4879300638E-02
2	2.1289740090E+00	1.2250434680E+00	1.8000033816E-01
3	2.0610917932E+00	8.8919488539E-01	
Theat =30.0 Ro =25.00 % Case B			

Table A-8 Sixth Order Filter Case C.

InputTheta = 30 Ro= 25 % n =6 case c

Output

n = 6 Ws = 2.12408 r/s

A_max= 0.28029 db A_min=80.92154 db

Actual attenuation in the stopband = 80.9215419030 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$		
	a1	a2	
-4.3879613890E-01 +j	2.6110292104E-01	8.7759227779E-01	2.6071678689E-01
-2.6262770762E-01 +j	7.6194968486E-01	5.2525541524E-01	6.4954063507E-01
-8.5502439588E-02 +j	1.0241220422E+00	1.7100487918E-01	1.0561366244E+00

Coefficients of the polynomial r(s) for the DENOMINATOR

Associated with the Elliptic Respse.

 $r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$

a0 =	1.7885266840E-01	a1 =	7.7562094308E-01	a2 =	1.7384353201E+00
a3 =	2.4230584628E+00	a4 =	2.6672479429E+00	a5 =	1.5738525722E+00
a6 =	1.0000000000E+00				

Zeros of the transfer functionWz1= 3.0249873471E+00 Wz2= 2.1958599083E+00

Quadratic factors of $s^{**2} + a1*s + a2$	NUMERATOR	
	a1	a2
	0.0000000000E+00	9.1505484503E+00
	0.0000000000E+00	4.8218007371E+00
	0.0000000000E+00	0.0000000000E+00

Coefficients of the polynomial r(s) for the NUMERATOR

Associated with the Elliptic Respse.

 $r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$

a0 =	4.4122121262E+01	a1 =	0.0000000000E+00	a2 =	1.3972349187E+01
a3 =	0.0000000000E+00	a4 =	1.0000000000E+00	a5 =	0.0000000000E+00
a6 =	0.0000000000E+00				

Table A-8 -- Continued.

R1 = 1.000 R2 = 1.0000000000E+00 Const. H = 4.0535827219E-03			

k	C(2k-1)	L(2k)	C(2k)

1	1.1972965749E+00	1.4275960011E+00	7.6550415514E-02
2	1.7097667011E+00	1.6382843728E+00	1.2659059827E-01
3	1.4295841139E+00	1.2707670562E+00	

Theat =30.0 Ro =25.00 % Case C			

Table A-9 Seventh Order Filter.

InputTheta = 30 Ro = 15 % n =7

Output

n = 7 Ws = 2.00000 r/s

A_max= 0.09883 db A_min=93.75705 db

Actual attenuation in the stopband = 93.7570543600 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$		
	a1	a2	
-4.0959528672E-01			
-3.5100546620E-01 +j	5.0219533573E-01	7.0201093241E-01	3.7540499253E-01
-2.1762510932E-01 +j	8.6254108092E-01	4.3525021865E-01	7.9133780448E-01
-7.1283447608E-02 +j	1.0374085418E+00	1.4256689522E-01	1.0812978125E+00

Coefficients of the polynomial r(s) for the DENOMINATOR

Associated with the Elliptic Respse.

r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n

a0 =	1.3157162321E-01	a1 =	6.5697791497E-01	a2 =	1.6354536207E+00
a3 =	2.8756824797E+00	a4 =	3.2708900641E+00	a5 =	3.2399383483E+00
a6 =	1.6894233330E+00	a7 =	1.0000000000E+00		

Zeros of the transfer functionWz1= 4.3544339261E+00 Wz2= 2.4903366117E+00
Wz3= 2.0445153147E+00
-----Quadratic factors of $s^{**2} + a1*s + a2$ NUMERATOR

a1	a2
0.0000000000E+00	1.8961094817E+01
0.0000000000E+00	6.2017764396E+00
0.0000000000E+00	4.1800428720E+00

Coefficients of the polynomial r(s) for the NUMERATOR

Associated with the Elliptic Respse.

r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n

a0 =	4.9154157064E+02	a1 =	0.0000000000E+00	a2 =	2.2277435174E+02
a3 =	0.0000000000E+00	a4 =	2.9342914129E+01	a5 =	0.0000000000E+00
a6 =	1.0000000000E+00	a7 =	0.0000000000E+00		

Table A-9 -- Continued.

R1 = 1.000 R2 = 1.0000000000E+00 Const. H = 2.6767140577E-04			

k	C(2k-1)	L(2k)	C(2k)

1	1.1463142985E+00	1.3802251284E+00	3.8210846609E-02
2	1.9178499179E+00	1.3529583923E+00	1.7682139927E-01
3	1.8543022161E+00	1.2705313141E+00	1.2691078191E-01
4	1.0644396334E+00		

Theat =30.0 Ro =15.00 % Case Odd			

Table A-10 Tenth Order Filter Case B.

InputTheta 60 ro = 20 % n = 10 case b

Output

n = 10 Ws = 1.16152 r/s
A_max= 0.17729 db A_min=80.81012 dbActual attenuation in the stopband = 80.8101185320 db

POLES	Quadratic factors of $s^{**2} + a1*s + a2$	
	a1	a2
-3.0520424939E-01 +j 2.2661838647E-01	6.1040849877E-01	1.4450552693E-01
-2.2341056741E-01 +j 6.0661444880E-01	4.4682113482E-01	4.1789337113E-01
-1.2818802242E-01 +j 8.4211159630E-01	2.5637604483E-01	7.2558410972E-01
-6.0202018257E-02 +j 9.6045252174E-01	1.2040403651E-01	9.2609332951E-01
-1.7153020768E-02 +j 1.0072503350E+00	3.4306041535E-02	1.0148474635E+00

Coefficients of the polynomial r(s) for the DENOMINATOR

Associated with the Elliptic Response.

 $r(s) = a0 + a1*s + a2*s^{**2} + \dots + a_{(n-1)}*s^{(n-1)} + s^{**n}$

a0 = 4.1180653091E-02	a1 = 2.3928001938E-01	a2 = 8.2660923881E-01
a3 = 1.8258521014E+00	a4 = 3.4594691060E+00	a5 = 4.4100875551E+00
a6 = 5.6545856257E+00	a7 = 4.2932864496E+00	a8 = 3.9800741938E+00
a9 = 1.4683157565E+00	a10 = 1.0000000000E+00	

Zeros of the transfer functionWz1= 2.1122466606E+00 Wz2= 1.4397415579E+00
Wz3= 1.2358587261E+00 Wz4= 1.1687177388E+00

Quadratic factors of $s^{**2} + a1*s + a2$	NUMERATOR	
	a1	a2
	0.0000000000E+00	4.4615859550E+00
	0.0000000000E+00	2.0728557534E+00
	0.0000000000E+00	1.5273467908E+00
	0.0000000000E+00	1.3659011531E+00
	0.0000000000E+00	0.0000000000E+00

Table A-10 -- Continued.

 Coefficients of the polynomial $r(s)$ for the NUMERATOR

Associated with the Elliptic Respnse.

$$r(s) = a_0 + a_1*s + a_2*s^{**2} + \dots + a_{(n-1)}*s^{**(n-1)} + s^{**n}$$

 $a_0 = 1.9293689012E+01$ $a_1 = 0.0000000000E+00$ $a_2 = 4.0389588691E+01$
 $a_3 = 0.0000000000E+00$ $a_4 = 3.0240188896E+01$ $a_5 = 0.0000000000E+00$
 $a_6 = 9.4276896522E+00$ $a_7 = 0.0000000000E+00$ $a_8 = 1.0000000000E+00$
 $a_9 = 0.0000000000E+00$ $a_{10} = 0.0000000000E+00$

 $R_1 = 1.000$ $R_2 = 6.6666666667E-01$ Const. $H = 2.1344105352E-03$

k	C(2k-1)	L(2k)	C(2k)
1	1.1917132071E+00	1.2120611641E+00	1.8492098562E-01
2	1.6369484920E+00	8.8328839276E-01	7.4124165977E-01
3	1.2686920634E+00	7.8299148755E-01	9.3502600095E-01
4	1.4090634895E+00	1.0556694334E+00	4.5698608038E-01
5	1.7567029245E+00	9.0806989400E-01	

 Theat =60.0 Ro =20.00 % Case B

APPENDIX B
PROGRAM LISTING

Program Elliptic;

Label 10, 15, 20, 25, 40, 45, 50, 60, la ;
const Lim = 25;

Type

Complex = RECORD
 re, im : real;
End;

Type

text80 = string[80];
str4 = string[4];
str40 = string[40];

array25 = Array[0..lim] of real;
Carray25 = array[1..lim] of Complex;

array25x25 = array[0..lim,0..lim] of real;
array2 = Array[1..2] of real;

var

j,i,Max_ord,n,raw,col,np_2,n_zeros,count,
ii,Elord,n_of_el, Ws_ty,A_max_ty : integer;
eps,k,kk,k1,kk1,fi,elin,Ws,n1,a,u,v,R1,R2,
ro,lamda,m,kkm,m1,kkm1,A_max,A_min,alpha,Angle,
Theta,ph,p0,wc,re_part,Im_part,Wpoz,Woz ,Argu,Magp : real;

case_a, case_b, case_c,oK,Screen : Boolean;

xk,argp,Wz,Wz1,Wpz,pr,qi,pz,
coef,Zcoef,Fa,E,Ele_val : array25;
pcoef,pc : array25x25;
Za : Carray25;
out_flag,Even_case : str4;
x : text80;
H : array2;
ScreenType, Ans, Ch, Choice : Char;
Out_Dev : Text;

{=====}
{ \$I Miscell.inc } (Include Misc. Procedures, Functions.)
 { parameters, and FastWrite routine }
{ \$I Help.txt } (Include some discription)
{=====}

```

{SI Read.in}
{SI Ellip.inc}
{SI MathPoly.inc}
{SI Rtrans.inc}
{SI Main_cal.inc}
{SI Sysodd.inc}
{SI SysEven.inc}
{SI print.scr}
{SI printo.dev}
(=====)

BEGIN          { main }

  ClrScr;
  InitMonoSpfx;
  MakeTitle('ELLIPTIC',3);
  MakeTitle('FILTER',11);
  Gotoxy(32,19);Bright('WRITTEN BY');
  Gotoxy(29,20);Bright('Mohamed AL-ZARIEY');
  Gotoxy(24,21);Bright('Supervised by Dr. L. P. HUELSMAN');
  Gotoxy(28,22);write('University of Arizona, Sep. 1987');
  ScreenType := ' ';
  Ans      := ' ';
  box(2,1,79,25,23);
  Gotoxy (10,24);
  Bright('Would '); Write ('you like to have this program run in Color? ');
  Blink;reverse(' (Y/N) '); Gotoxy(wherex-1,wherey);

  While not (ScreenType in ['Y', 'N']) do
    Begin
      Read (KBD, ScreenType);
      ScreenType := UpCase (ScreenType);
    End; CursOn;
    ScrType(ScreenType);
  (-----)

10:  ClrScr;
      Wopen(9);      ClrScr;
      gotoxy(2,1);Write('Date: ',date,' Time: ',Time);

      Gotoxy (10,3);
      Write ('This Program will find the Elliptic Network Function. ');
      Gotoxy (15,4);Write(' and its Ladder Realization. ');

      Gotoxy (2,6); Bright('The Elliptic network function is described');
      Bright(' by the following parameters. ');
      raw :=2; col := 7;

```

```

FastWrite(raw,col,$07,
          '1. The ripples in the passband, A_max      ');
FastWrite(raw,col+1,$07,
          '2. The minimum attenuation in the stopband, A_min ');
FastWrite(raw,col+2,$07,
          '3. The order of the filter, n            ');
FastWrite(raw,col+3,$07,
          '4. The Stopband frequency, ws          ');
FastWrite(raw,col+4,$07,
          '5. The Case for even order functions, (A, B, or C) ');

col := 13; Gotoxy(raw,col);
Bright('With');WriteLn(' any two of the first three',
          ' parameters and the stopband frequency.');
```

15: FastWrite(raw,col+1,\$09,
'One can determine completely the Elliptic Network Transfer Function');

x := 'according to the following three possible Combinations:';

```

FastWrite(raw,col+2,$09,x);
FastWrite(3,17,$17,'1) Combination A (The order n is Unknown) ');
FastWrite(3,18,$17,'2) Combination B ( A_min is Unknown) ');
FastWrite(3,19,$17,'3) Combination C ( A_max is Unknown) ');

Gotoxy(8,23); Bright(' E');Write('nter option (1, 2, 3, or)');
Bright(' H');Write('(elp) or ');
Bright('E');Write('(xit) ) ');
reverse(' ');Gotoxy(wherex-2,wherey);
```

```

15: Read(KBD,Ans);
Ans := UpCase (Ans);
```

```

Case Ans of
  'H' : Begin
        CursOFF;Help(2,1); WClose;
        CursON; Goto 15;
      end;
  '1'..'3': Begin
        Wopen(7); ClrScr;
        Wopen(6); ClrScr;
        col :=2; raw :=2;
        Case Ans of
          '1' : begin
                Choice := '1';
                StopBand_Frequency(Ws,Theta);
                col := col+ 4;
                PassBand_Ripple(A_max,ro);
                col := col +4;
                StopBand_Attenuation(A_min);
                Max_ord := 0;
              end;
          '2' : begin
                Choice := '2';
                StopBand_Frequency(Ws,Theta);
                col :=col +4;
                PassBand_Ripple(A_max,ro);
                col := col +4;
                Filter_Order(Max_ord);
                A_min := 0;
              end;
          '3' : begin
                Choice := '3';
                StopBand_Frequency(Ws,Theta);
                col := col +4;
                Filter_Order(Max_ord);
                col := col +4;
                StopBand_Attenuation(A_min);
                A_max := 0; eps := 0;
              end;
        end;
        end; { Casa of Combinations }
        Check_input; { and modify if necessary }
        Wclose; { Close Window(6) }
        Theta := ArcSin(1/Ws);
        Wclose; { Close Window(7) }
        Wclose; { Close Window(9) }
        Goto 20;
      end;
  'E' : begin Wclose; Goto 25; end

```

```

else
  Begin
    Click;
    goto 15;
  end;
end;

20:  ClrScr;                { main calculations  }
                                { Initialize Variables }
    case_a := false;        case_b := false;
    case_c := false;        Even_case := ' Odd Case ';
    R1 := 1;                R2 := 1;    m := 0;
    H[1] := 0;              H[2] := 1;    Elord := 0;
    Wc := 1;                k := Wc/Ws; count := 1;
    raw := 4;               col := 1;
    ch := ' ';

    eps := power(10,0.1*A_max) -1;    eps := sqrt(eps);
(-----)
    Wopen(7); ClrScr;
40:  If A_min <> 0 then
    m := eps/sqrt(power(10,0.1*A_min) -1);
    m1 := sqrt(1 - sqr(m));
    k1 := sqrt(1 - sqr(k));

    if (Max_ord = 0 ) then
    begin
      CalculateMax_order(Max_ord);
      Gotoxy(raw,col); write(' Filter"s Order (n) = ',max_ord);
    end;

    if (count = 1) then
    if not odd(Max_ord) then
    begin
      col := col +2;
      Gotoxy(raw,col);
      Bright ('Enter (1, 2, 3) for Even Cases (A, B, C) or Q');
      Reverse('(uit) => ');
45:  read(KBD,ch); ch := Ucase(ch); write(ch);
      col := col +2;

```

```

    case ch of
      'Q': begin Click; Goto 10;
            end;
      '1': begin
            case_a := true;
            Even_case := ' A';
            end;
      '2': begin
            case_b := true;
            Even_case := ' B';
            end;
      '3': begin
            case_c := true;
            Even_case := ' C';
            end
    else
      begin
        Click; ch := ' ';
        Goto 45;
      end;
    end; { end cass }
  end; { end of even cases }
  (----- Get outpou devive -----)

```

```

  If count = 1 then
    begin
      Get_outPut_dev(out_dev);
      if Screen then ch := #196 else ch := '-';
      Wclose; { close window(7) }
    end;
  (-----)

```

```

  if case_b then R2 := (1-ro)/(1+ro);
  if (R2 < 1000 ) then Find_H_eps;

```

```

  n := Max_ord;
  j := 0;
  Ellin(xk,elin,k,fi,j); kk := elin;
  kkm := F(90,m);

```

```

  if odd(n) then
    begin { Bigin Odd case }
      np_2 := (n-1) div 2;
      n_zeros := np_2;
      OddFn(n,Wz,Wpz);
    end

```

```

else
  begin
    { Even case }
    np_2 := n div 2;
    EvenFn(n,Wz,Wpz,Wpoz,Woz);
    if ( case_a ) then
      n_zeros := np_2
    else
      n_zeros := np_2 - 1;
  end;

  if eps <> 0 then
    alpha := 10* ln(1+sqr(eps/m))/ln(10)
  else
    begin
      eps := (A_min*Ln(10))/10 ;
      eps := m * sqrt(Exp(eps) - 1);
      A_max := 10* Ln(1+sqr(eps))/Ln(10);
      Ro := sqrt(1- exp(-A_max*Ln(10)/10));
      alpha := A_min;
      count :=count +1;
      if count <=2 then Goto 40;
    end;
    Writeln;
    { writeln(' A_min ',A_min,' alph ',alpha); }

  (----- )

  if (abs(alpha - A_min) <= 1.0E-7 ) then
    begin
      A_min :=alpha;
      Goto la;
    end
  else
    { With the new A_min }
    { recalculate Caure }
    { parameters }
    begin
      if count > 2 then goto la;
      if case_b then
        A_min := Alpha {+ H[1] or - A_max }
      else
        A_min := Alpha ;

      count := count +1;
      k := sin(Theta);   Ws := 1/k;
      Goto 40;          { Go back with the old values }
    end;
  la:  LowVideo; {WaitKey;}

  (----- )

  kk := F(90,k);
  kkm := F(90,m);

  a := kk/(n*kkm) * Arcsn(1.0/eps,m);      { m -> k! }

```

```

for i := 0 to n do pz[i] := 0;

  if odd(Max_ord) then
    begin
      OddRoots(max_ord,pcoef,coef,pr,qi);
      if Screen then print_out('Den') else print_out_toDev('Den');
      lamda := 1;
      R2 := 1; R1 := 1; H[2] := 1;
      Ellip_coef(pz,Wz,max_ord,pcoef,Zcoef); { Odd case }
      Goto 50;
    end
  else { Even Network function }
    begin
      EvenRoots(max_ord,pcoef,coef,pr,qi);

      if Screen then print_out('Den') else print_out_toDev('Den');
      lamda := sqrt( 1 - H[2]/(1 + sqr(eps/m)));

      if(H[2] >= 1) then
        begin
          Ellip_coef(pz,Wz,max_ord,pcoef,Zcoef);
          goto 50;
        end
      else
        begin
          for i := 0 to n do
            Fa[i] := Coef[i];
            eps := eps/sqrt(1 - H[2]);
            a := kk/(n*kkm) * Arcsn(1.0/eps,m); { m -> kl' }
            EvenRoots(max_ord,pcoef,coef,pr,qi);
            for i := 0 to n do
              E[i] := coef[i];
            goto 60;
          end
        end
      end;
    end;

50:  for i := 0 to n do
      begin
        fa[i] :=0;E[i] :=0;
        Fa[i] := Coef[i]; E[i] := Zcoef[i];
      end;

60:  Ellip_coef(pz,Wpz,max_ord,pcoef,coef); { calculate numerator }
      if (n =2) or (n = 1) then goto 10; { do not find the elements }

      if not case_a then
        { begin shifting the numerator coefficients }
        begin { those produced by pole-zero shifting }
          for i := 1 to n do coef[i-1] := coef[i]; coef[n] :=0;
          if( n mod 2 = 0 ) then begin

```

```

for i:= 1 to n-1 do coeff[i-1] := coeff[i];
      coeff[n-1] := 0; end;
end;
if Screen then print_out('Num') else print_out_toDev('Num');

for i := 1 to n_zeros do  Za[i].im := Wpz[i];
Theta := theta*180/pi ;      { back to deg }
H[1] := Fa[0]/coeff[0];      { Constant of the Transfer fn }
                                { (err n odd casea)}

lamda := 1;

for i:=0 to n do  writeln('num E ',E[i],' den F ',fa[i]); readln;
                )

if not case_a then
begin
  if odd(n) then
    Odd_NetWork(Max_ord,Za,fa,E,R1,R2,Elord,n_of_el,Ele_val)
  else
    Even_NetWork(Max_ord,Za,fa,E,R1,R2,lamda,Ro,Elord,
                 case_c, n_of_el,Ele_val );
  if Screen then
    Write_Element(n_of_el, Ele_val)
  else
    Write_ElementDev(n_of_el, Ele_val);

  gotoxy(3,wherey+2);
  write(out_dev,
        'Theat =',theta :-3:1,'      Ro =',Ro*100 :-3:2 , ' %');
  writeln(out_dev,'      Case ',Even_case);
  if Screen then
    begin  box(1,1,80,24,1);      WaitKey; end;
end;
close(out_dev);
Goto 10;
25: Window (1,1,80,25);
ClrScr; CursNorm;
Gotoxy (1,24);
TextColor(0);
TextBackground (15);
end.

```

(=====)

```

procedure TryAgain(flag : integer);
begin
    Wopen(10); ClrScr; gotoxy(2,2);
    if flag = 1 then
        write('Wrong Value Try Again ');
    else if flag = 2 then
        write('The value 0 is not allowed');
    else Write('Not valid Key ');

    Sound(100); delay(500);NoSound;
    delay(2000);Wclose;
end;

```

(-----)

(*Read.in*)

(*This procedure reads the input and check for errorr*)

Procedure Read_in(var value : real;i :integer);

label 10;

begin

```

10:    Repeat
        gotoxy(20,col+i);Write(' => ');
        {SI-} read(value) {SI+}; ok := (IOresult =0 );
        if not ok then
            begin
                gotoxy(20,col+i);ClrEol; TryAgain(1);
            end;
        Until ok;
        if value = 0 then
            begin
                TryAgain(2); goto 10;
            end;

```

end;

(-----)

```

Procedure Check_input;
var
  Temp   : array25;
  TempCh : array[1..25] of str40;

begin
  Angle := Theta;
  if Ws_ty = 1 then
    begin Temp[1] := Ws ;   TempCh[1] := 'Ws      = ' ; end
  else begin Temp[1] := Angle; TempCh[1] := 'Angle  = ' ; end;

  if Choice in ['1'..'2'] then
    if A_max_ty = 1 then
      begin Temp[2] := A_max; TempCh[2] := 'A_max  = ' ; end
    else
      begin Temp[2] := Ro;   TempCh[2] := 'Ro      = ' ; end;

  if Choice in ['1','3'] then
    begin Temp[3] := A_min; TempCh[3] := 'A_min  = ' ; end;

  if Choice in ['2'] then
    begin Temp[3] := Max_ord; TempCh[3] := 'Max_ord = ' ; end;
  if Choice in ['3'] then
    begin Temp[2] := Max_ord; TempCh[2] := 'Max_ord = ' ; end;

  Wopen(8); ClrScr;
  for i:=1 to 3 do
    begin
      gotoxy(2,i+1);write(i,') ',TempCh[i] ,Temp[i]:4:5);
    end;
  repeat
    gotoxy(2,6);ClrEol; write(' Is it ok ? (Y/N): ');
    repeat read(Kbd,Ans); Ans := Upcase(Ans);
    Until Ans in ['Y','N'];

  if Ans = 'N' then
    begin
      gotoxy(2,6);ClrEol;write(' Press the number : ');
      Repeat read(i); Until i in [1..3];
      gotoxy(2,i+1);ClrEol;write(i,') ',TempCh[i]); read(temp[i]);
    end;
  until Ans = 'Y'; Wclose; { Close window(8) }

  ( ----- Restore the values back ----- )

  if Ws_ty = 1 then Ws :=Temp[1]
  else begin Angle :=Temp[1];
          Ws := 1.0/sin((Angle*1.745329252E-2));
        end;

```

```

if Choice in ['1'..'2'] then
  if A_max_ty = 1 then
    begin
      A_max :=Temp[2]; Ro := sqrt(1- exp(-A_max*Ln(10)/10));
    end
  else
    begin
      Ro :=Temp[2];
      if (Ro >= 1) then Ro := Ro/100;
      A_max := -10*( Ln(1-Ro*Ro)/Ln(10) );
    end;

  if Choice in ['1','3'] then A_min :=Temp[3];
  if Choice in ['2'] then Max_ord :=Trunc(Temp[3]);
  if Choice in ['3'] then Max_ord :=Trunc(Temp[2]);
end;
(-----)

Procedure StopBand_Frequency(var Ws,Angle : real);

label start;
Var
  a      : Char;

begin
  Ws := 0; Angle := 0;
  Gotoxy(raw,col);Write(' Stopband frequency:');
  gotoxy(raw+2,col+1);Write(' 1) Ws (r/s) ');
  gotoxy(raw+2,col+2);Write(' 2) Angle (deg) ');
  repeat read(KBD,a); until a in ['1'..'2'];
  if (a = '1') then
    begin
      gotoxy(raw,col+2);clrEol;
      Read_in(Ws,1); Angle :=180/pi * Sin(1/Ws);
      Ws_ty := 1;
    end
  else
    begin
      gotoxy(raw,col+1); ClrEol;
      read_in(Angle,2); Ws_ty := 2;
      Ws := 1.0/sin(( Angle*1.745329252E-2));
    end;
end;
(-----)

```

```

Procedure PassBand_Ripple(var A_max,ro : real);

Var
  a      : Char;
begin
  Gotoxy(raw,col);Write(' Passband Ripple :');
  gotoxy(raw+2,col+1);Write(' 1) A_max (dB) ');
  gotoxy(raw+2,col+2);Write(' 2) Ro (%) ');
  repeat read(KBD,a); until a in ['1'..'2'];
  if (a = '1') then
    begin
      gotoxy(raw,col+2);ClrEol;
      read_in(A_max,1); A_max_ty := 1;
      Ro := sqrt(1- exp(-A_max*Ln(10)/10));
    end
  else
    begin
      gotoxy(raw,col+1); ClrEol;
      read_in(Ro,2); Ro := Ro/100;  A_max_ty :=2;
      A_max := -10*( Ln(1-Ro*Ro)/Ln(10) );
    end;
  end;
end;
(-----)
Procedure StopBand_Attenuation (var A_min : real);
begin
  Gotoxy(raw,col);Write(' Stopband Attenuation :');
  gotoxy(raw+2,col+1);Write('  A_min (dB) ');
  gotoxy(20,col+1);
  read_in(A_min,1);
end;
(-----)
Procedure Filter_order(var Max_ord : integer);
label 10;
begin
  Gotoxy(raw,col);
  Write(' Filter"s order (n):');
  gotoxy(raw+2,col+1);Write(' Max_ord ');
10:   Repeat
      gotoxy(20,col+1);Write(' => ');
      {$I-} read(Max_ord) {$I+}; ok := (IOresult =0 );
      if not ok then
        begin
          gotoxy(20,col+1);ClrEol; TryAgain(1);
        end;
      Until ok;
      if Max_ord = 0 then
        begin
          TryAgain(2); goto 10;
        end;
      end;
end;
(-----)

```

```

Procedure get_output_dev (var fp : text);
label again;
Const
    Terminal = 'TRM: ';
    Printer = 'LST: ';

Type
    Fname = String[14];

Var
    dest : Fname;
    (-----)

Function exists(filename : Fname):Boolean;
    Begin
        assign (fp, filename);
        {I-} Rewrite(fp) {I+} ;

        If ioresult <> 0 Then
            begin
                exists := False;
                Gotoxy(raw, col+1);
                Write(^G, 'Can't open 'Filename,' Try again');
            end
        Else
            exists := True;
    End;

Begin      { begin get the output file }

    Choice := ' ';      Screen := False;

    gotoxy(raw, col); write('The output can be directed to:');
    raw := 30; col := col + 1;
    gotoxy(raw, col+1); write(' 1.   Screen   ');
    gotoxy(raw, col+2); write(' 2.   Text File ');
    gotoxy(raw, col+3); write(' 3.   Printer  ');

    again: Gotoxy(raw, col+4); Write('Enter 1, 2, 3 => ');
    Repeat read(KBD, Choice); Write(choice);
    until choice in ['1'..'3'];

    col := col + 5;

```

```

Case Choice of
  '1': begin
        dest := Terminal;
        Screen := True;
        assign(fp,dest); Rewrite(fp);
      end;
  '2': Begin
        Repeat
          gotoxy(raw,col+1);
          Write('Enter OutPut File Name => ');
          Read(dest);
        Until exists(dest);
      end;
  '3': begin
        dest := Printer;
        assign(fp,dest); Rewrite(fp);
      end;
                                     ( go back to main )
Else
  goto again;
end;  ( End case )

End;

```

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

```
Procedure ELLIN(var xk: array25; var elin,x,fi :real;var j :
integer);
```

```
  ( Elliptic integral )
```

```
VAR
```

```
  i : integer;
```

```
  b : real;
```

```
BEGIN
```

```
  xk[1] := x ;
```

```
  fi := 1/2;
```

```
  elin := pi/2;
```

```
  j :=1;
```

```
  While ( xk[j] > 1.0E-18) do
```

```
    BEGIN
```

```
      b :=SQRT(1.0 - xk[j] * xk[j] ); j := j+1;
```

```
      xk [j] := (1.0 - b) / (1.0 + b);
```

```
      elin := elin * ( 1.0 + xk[j] );
```

```
      fi := 2*fi /(1.0 + xk[j]);
```

```
    end; j := j-1;
```

```
End;
```

```
Function ArcSin (y : real) : real;
```

```
begin
```

```
  ArcSin := arctan(y/sqrt(1 - y*y));
```

```
end;
```

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

```

Function Phi (var fi: real; argp,xk: array25; n,m: integer):real;
VAR
  amp : array25;
  i,ii,j : integer;
BEGIN
  amp [n] := argp[m] * fi;
  FOR i := 1 to n-1 do begin
    ii := n-i;
    j := ii+1;
    amp [ii] := (amp[j] + ArcSin(xk[j]*Sin(amp[j]))) / 2.0;
  end;
  phi := amp[ii];
end;
-----

```

```
Function F(ph,k : real) :real;
```

```
label 5, 10;
```

```
var
```

```
ck,ari,pim,sqgeo,geo : real;
```

```
angle,test,x,res,aari : real;
```

```
begin
```

```
angle := (ph*pi)/180;
```

```
x := sin(angle)/cos(angle); ck := sqrt(1-sqr(k));
```

```
if (x = 0) then
```

```
begin
```

```
res :=0;
```

```
goto 10; end;
```

```
if (ck = 0) then
```

```
begin
```

```
res := Ln(abs(x) + sqrt(1+x*x));
```

```
goto 10; end;
```

```
angle := abs(1/x);
```

```
geo := abs(ck);
```

```
ari := 1;
```

```
pim := 0;
```

```
5: sqgeo := ari * geo;
```

```
aari := ari;
```

```
ari := geo + ari;
```

```
angle := -sqgeo/angle + angle;
```

```
sqgeo := sqrt(sqgeo);
```

```
if (angle = 0) then
```

```
angle := sqgeo *1.0E-18; {replace zero by small number
}
```

```
test := aari * 1.0E-10;
```

```

if (abs(aari - geo) > test ) then
begin
  geo := sqgeo + sqgeo;
  pim := pim + pi;
  if ( angle < 0 ) then
    pim := pim + 3.1415927; { add pi to pim }
  goto 5;
end;

if (angle < 0 ) then
  pim := pim + pi;
res := (ArcTan(ari/angle) + pim)/ari;
10: if (x < 0 ) then
  res := - res;
  F := res;
end;

(-----)
function ArcSn(u,v :real) :real;
var
  w,x,k1 :real;

begin
  W := 1/sqrt(1 + sqr(1/u));
  k1 := sqrt(1 - sqr(v));
  x := ArcSin(w); { rad} x := (x*180)/pi; { deg }
  ArcSn := F(x,k1);
end;

(-----)

```

```

Procedure Jac_fun (u,v,k: real; var re_part,Im_part : real);

type
  arry2 = array[1..2] of real;
var
  fi,ph,k1,elin,den: real;
  j,i, n : integer;
  sign : string [2] ;
  xk,argp : array2S;
  sn,cn,dn,tn : array2;

begin
  k1 := sqrt(1 - sqr(k));
  argp[1] := u;
  argp[2] := v;

  if (u <> 0) then
    begin
      { Sn(u) ...etc.}
      i := 1;
      ellin(xk,elin,k,fi,j);
      ph := Phi (fi,argp,xk,j,i);
      sn[i] := sin(ph);
      cn[i] := cos(ph);
      dn[i] := sqrt(1 - sqr(k*sin(ph)));
      tn[i] := sn[i]/cn[i];
    end;

    if( v <> 0) then
      begin
        { sn (v) ... etc.}
        i := 2;
        ellin(xk,elin,k1,fi,j);
        ph := Phi (fi,argp,xk,j,i);
        sn[i] := sin(ph);
        cn[i] := cos(ph);
        dn[i] := sqrt(1 - sqr(k1*sin(ph)));
        tn[i] := sn[i]/cn[i];
      end;

      { Sn with complex argument }
      den := sqr(cn[2]) + sqr ( k * sn[1] * sn[2]);
      re_part := (sn[1]*dn[2])/den;
      Im_part := (cn[1]*dn[1]*sn[2]*cn[2])/den;

end;
{-----}

```

{elliptic coefficients}

```
procedure Ellip_coef( pr,qi : array25 ;np :integer;
                    var pcoef:array25x25;var coef: array25);
```

```
var
```

```
  i,np_2,k,j : integer;
```

```
begin
```

```
  pcoef [0,0] := 1;
```

```
  pcoef [1,0] := 0;
```

```
  pcoef [2,0] := 0;
```

```
(----- Coefficients -----)
```

```
  for i := 0 to np do
```

```
    coef [i] := 0;
```

```
  if odd (np) then
```

```
    np_2 := (np -1) div 2
```

```
  else
```

```
    np_2 := np div 2;
```

```
  For i := 1 to np_2 do
```

```
    begin
```

```
      pcoef [2,i] := 1;
```

```
      pcoef [1,i] := - 2* pr[i];
```

```
      pcoef [0,i] := pr[i]*pr[i] + qi[i] * qi[i];
```

```
    end;
```

```
(    for i :=1 to np_2 do
```

```
  for j:= 0 to 1 do
```

```
Gotoxy(40,col); write(' ',pcoef[j,i] ); )
```

```
(----- )
```

```
  if odd(np) then
```

```
    begin
```

```
      pcoef [0,0] := - pr[np_2+1 ]; {This is p0 }
```

```
      pcoef [1,0] := 1;
```

```
      pcoef [2,0] := 0;
```

```
      { for j:= 0 to 1 do
```

```
        writeln('pco ',pcoef[j,0]); }
```

```
    end;
```

```
(writeln(' s^2 + a_n-1*s^(n-1) .....+ao' ); )
```

```
for i := 1 to np_2 do
```

```
begin
```

```
  for j := 0 to 2 do begin
```

```
    for k := 0 to i*2 do
```

```
      coef [k+j] := pcoef[k,0] * pcoef[j,i] + coef[j+k];
```

```
    end; { end j}
```

```

    for k := 0 to i*2 + 2 do
    begin
        pcoef [k,0] := coef[k];
        if ( i < np_2) then
            coef[k] := 0;
        end;
    end;    { end i }

{
    for i := 0 to np-1 do
        write (":4,'a('i:1,')',coef[i]:10:7); }
end;
{-----}

Function power( x,y : real) : real;
begin
    if (x <> 0) then
        power := EXP( y*ln(x))
    else
        power := 0;
    end;
{-----}
{MulPoly.inc}
Procedure MulPoly(x,y : array25;var z:array25; xord,yord:integer;
    var zord : integer);
{ This procedure form the product of two polynomials
  Z = Resulting coefficient array, Z[0] is the constant term.
  zord = Order of Z(s) <= 20.
  X = Coefficient array for X(s).
  xord = Order of X(s).
  Y = Coefficient of array Y(s).
  yord = order of Y(s). }

var
    i,j,k : integer;
Begin
    if (xord*yord > 0 ) then
    begin
        Zord := xord + yord;
        for i := 0 to zord do Z[i] := 0;
        for i := 0 to xord do
            for j := 0 to yord do
            begin
                k := i + j;
                Z[k] := X[i]*Y[j] + Z[k];
            end;
        end
    end
end;
{-----}

```

{SubPoly.inc}

```

  Procedure Subtract(x,y : array25; var Z : array25; xord,yord : integer;
                    var zord : integer);
{ The Subtract is used to form the difference of two polynomials
as
   $Z(s) = X(s) - Y(s)$ 

  Z = Resulting polynomial array, Z(0) is the constant term
  zord = Order of Z(s) <= 20
  X = Coefficient array for X(s)
  xord = Order of X(s)
  Y = Coefficient array for Y(s)
  yord = Order of Y(s)
}

```

```

var
  i,k,largOrder : integer;

```

```

begin
  if( xord > yord ) then
    largOrder := xOrd

  else
    largOrder := yord;
  if(largOrder >0 ) then

    For i := 0 to largOrder do
      begin
        if ( i <= xord ) then
          if ( i <= yord ) then
            Z[i] := X[i] - Y[i]
          else
            Z[i] := X[i]
        else
          Z[i] := - y[i];
      end;
      zord := largOrder;

```

```

end;

```

{-----}

(Normalize Polynomial)

Procedure NormPoly (var x : array25; var xord : integer; tolerance : real);

{

The Procedure NormPoly is used to check the coefficients of a polynomial starting from the coefficient of the highest order and set them to zero if they do not exceed the " tolerance " until one which does exceed "tolerance " is found.

Definition of symbols

=====

xary = Coefficient of array xary(s), xary(0) is the constant term

xord = order of the polynomial

tolerance = Theshold of the search

NOTES

This procedure is used in connection with DevPoly to eliminate extraneous terms ; also to eliminate Zero coefficients

}

var i : integer;

Begin

if (xord >= 0) then
while (abs(x[xord]) <= tolerance) do
xord := xord - 1;

(for i:= 0 to xord do*
if (xary[i] < 0) then xary[i] := 0;
**)*

end;

(-----)

{Complex procedures}

(These procedures require a type definition as follows:*

TYPE complex = RECORD

re, im : real

END;

**)*

PROCEDURE Add (x, y : complex; VAR z : complex);

BEGIN

z.re := x.re + y.re;

z.im := x.im + y.im

END;

PROCEDURE Sub (x, y : complex; VAR z : complex);

BEGIN

z.re := x.re - y.re;

z.im := x.im - y.im

END;

PROCEDURE Mul (x, y : complex; VAR z : complex);

VAR q : complex;

BEGIN

*q.re := x.re*y.re - x.im*y.im;*

*q.im := x.re*y.im + x.im*y.re;*

z := q

END;

```

PROCEDURE Dvd (x, y : complex; VAR z : complex);
  VAR q : complex;
      d : real;
  BEGIN
    d := sqr(y.re) + sqr(y.im);
    if ( d=0) then d:=0.0000001;
    q.re := (x.re*y.re + x.im*y.im) / d;
    q.im := (x.im*y.re - x.re*y.im) / d;
    z := q
  END;

FUNCTION Mag (x : complex) : real;
  BEGIN
    Mag := sqrt(sqr(x.re) + sqr(x.im))
  END;

FUNCTION Arg (x : complex) : real;
  CONST pi = 3.141592653;
  VAR a : real;
  BEGIN
    WITH x DO
    IF re = 0.0 THEN
      IF im = 0.0 THEN
        a := 0.0
      ELSE BEGIN
        a := 0.5 * pi;
        IF im < 0.0 THEN
          a := -a
        ELSE
          END
      ELSE BEGIN
        a := arctan(im/re);
        IF re < 0.0 THEN
          IF im = 0.0 THEN
            a := pi
          ELSE
            IF im < 0.0 THEN
              a := a - pi
            ELSE
              a := a + pi
            ELSE
          END;
        Arg := a
      END;
    END;

PROCEDURE Expc (x : complex; VAR z : complex);
  VAR r : real;
  BEGIN
    r := exp(x.re);
    z.re := r * cos(x.im);
    z.im := r * sin(x.im)
  END;

```

Function Min(x,y : real) : real;

begin

if (x > y) then Min := y Else Min := x;

end;

(-----)

(calculates the value of a polynomial at complex frequency jw)

PROCEDURE ValPoly (n : integer; a : array25;

s : complex; VAR val : complex);

(Procedure for finding the (complex) value of a polynomial when evaluated at a (complex) value of its argument*

n - Degree of polynomial

a - Array of coefficients of polynomial

s - Complex value of argument of polynomial

value - Value of polynomial

The polynomial is assumed to have the form

*a[0] + a[1]*s + a[2]*s**2 + a[3]*s**3 + ...*

*Note: This procedure calls the procedures Add and Mul *)*

VAR

i : integer;

z : complex;

BEGIN

z.im := 0.0;

val.re := a[n];

val.im := 0.0;

i := n;

IF n > 0 THEN

REPEAT

BEGIN

i := i - 1;

z.re := a[i];

Mul(val, s, val);

Add(val, z, val)

END

UNTIL i = 0

END;

(-----)

(Reorder the order of the transmission zeros)

Procedure Zeros_seq(Za : Carray25; var Zn : Carray25; nd : integer);

var

l,i,la, idj,j : integer;

begin

l:= nd-2 ; i:=1;

Zn[nd] := Za[2];

Zn[1] := Za[1];

While (l > 0) do

begin

i :=i +1;

*Zn[i] := Za[2*i-1];*

l := l -2;

end;

if(l < 0) then

la := nd -1

else

la := nd;

While (la >= 2) do

begin

i := i+1;

Zn[i] := Za[la];

la := la-2;

end; {

for i:=1 to nd do writeln(' fix ',zn[i].im);}

end;

{-----}

(divided two polynomials)

Procedure DevPoly(x,y : array25;var z:array25; xord,yord:integer;

var zord : integer);

{ This procedure is used to form the quotient of two polynomials as

$$Z(s) = X(s)/Y(s)$$

Z = Resulting coefficient array, Z[0] is the constant term.

zord = Order of Z(s) <= 25.

X = Coefficient array for X(s).

xord = Order of X(s).

Y = Coefficient of array Y(s).

yord = order of Y(s).

tol = tolerance value; (0.0001);

Ier = Error indicator = 0 no error

= 1 error

}

```

label 10;
var
  ii,i,j,k,Ier : integer;

Begin
  Normpoly(y,yord,0.0001);
  if (yord <= 0 ) then
    begin
      Ier := 1;
      goto 10;
    end
  else
    Zord := xord - yord ;

    if (zord > 0 ) then
      begin
        xord := yord;
        i := zord;
        While(i >= 0 ) do
          begin
            ii := i + xord;
            Z[i] := x[ii]/y[yord];
            for k := 0 to xord do
              begin
                j := k + i;
                x[j] := X[j] - Z[i]*y[k];
              end;
            i := i-1;
          end;
          Normpoly(z,zord,0.0001);
          Ier := 0; zord := zord ;
          goto 10;
        end
      else
        zord := 0;
        Ier := 0;
      10: end ;

```

(-----)

(Exchange two Polynomials)

Procedure ExchangPoly(var A,B :array25;var a_ord,b_ord : integer);

(The procedure ExchangPoly is used to replace the array A with the array B;

*A = Array of A polynomial
a_ord = order of A(s)
B = Array of B polynomial
b_ord = Order of B(s)*

Note : Upon EXIT the a_ord of the array A will be replaced by the b_ord of the array B

)

var

*temp : array25;
larg,i : integer;*

(-----)

*PROCEDURE Switch_Order (VAR m : integer; VAR n : integer);
(* This procedure is to switch the values of integers m and n*

Additional Subprograms Required None

Input Arguments

*m The integer to be changed to n
n The integer to be changed to m*

Output Argument

*m The integer with the value of n
n The integer with the value of m *)*

VAR

temp,i : integer;

BEGIN

*temp := m;
m := n;
n := temp
END;*

```

begin { ExchangPoly }
  if (a_ord > b_ord ) then
    larg := a_ord
  else
    larg := b_ord;

  for i := 0 to larg do
  begin
    Temp[i] := A[i];
    A[i] := B[i];
    B[i] := Temp[i];
  end;

  Switch_Order( a_ord , b_ord);
end;

{-----}

{Hurwitz Polynomial }

PROCEDURE HrzPoly (n : integer; hrz : array25 ; VAR evn :array25;
  VAR od : array25; VAR neven : integer;
  VAR nodd : integer);
(* This procedure is to separate the Hurwitz polynomial into
  even and odd parts.
  Documentation
  n      The order of the Hurwitz polynomial
  hrz    The array of coefficients of Hurwitz polynomial
  evn    The array of coefficients of even part of Hurwitz
  polynomial
  od     The array of coefficients of odd part of Hurwitz
  polynomial
  neven  The order of even part of Hurwitz polynomial
  nodd   The order of odd part of Hurwitz polynomial

  Additional Subprogram  None

  User-defined Type
  array11 = ARRAY[0..10] OF real

  Input
  n
  hrz[0] hrz[1] .. hrz[n]
  Output
  evn[0] 0.0 evn[2] 0.0 .. evn[i] i<=n
  0.0 od[1] 0.0 od[3] 0.0 .. od[j] j<=n
  neven
  nodd *)

  var i : integer;

```

```

BEGIN
  FOR i:=0 TO n DO
    BEGIN
      IF Odd(i) = FALSE THEN
        BEGIN
          evn[i] := hrz[i];
          evn[i+1] := 0.0;
          neven := i
        END
      ELSE
        BEGIN
          od[i] := hrz[i];
          od[i-1] := 0.0;
          nodd := i
        END
      END
    END
  END;

  {-----}
  {Transfer the even case A into case B}

  procedure Transf_Root_ToCase_b( var pr,qi : array25);

  var
    p1,p2 : Complex ;
    Magb,Argb : real;

  begin
    { --- Numerator --- }
    p1.re := (sqr(Wpoz) -1)* sqr(Magp) * cos(2*argu);
    p1.im := (sqr(Wpoz) -1)* sqr(Magp) * sin(2*argu);
    { Denominator }
    p2.re := sqr(Wpoz) + sqr(Magp) * cos(2*Argu);
    p2.im := sqr(Magp)*sin(2*argu);
    Magb := sqrt( Mag(p1)/Mag(p2) );
    Argb := (Arg(p1) - Arg(p2))/2;

    { Returned values }
    pr[i] := - abs(Magb * cos(Argb));
    qi[i] := abs( Magb * sin(Argb));

  end;

  {-----}

```

{Transfer even case B to case C}

```
procedure Transf_Root_ToCase_c( var pr,qi : array25);
```

```
var
```

```
  p1 : Complex ;
```

```
  Magc,Argc : real;
```

```
begin
```

```
  p1.re := sqr(Wpoz) + sqr(Magp) * cos(2*Argu);
```

```
  p1.im := sqr(Magp)*sin(2*argu);
```

```
  Magc := sqrt( Mag(p1)/(1 - sqr(Wpoz)));
```

```
  Argc := Arg(p1)/2;
```

```
      { Returned values }
```

```
  pr[i] := -abs(Magc * cos(Argc));
```

```
  qi[i] := abs(Magc * sin(Argc));
```

```
end;
```

```
{=====}
```

{Calculates k1}

```
Function fkl( n :integer;k :real; Wz :array25) : real;
```

```
var
```

```
  i,n1 : integer;
```

```
  den,num : real;
```

```
begin
```

```
  if odd(n) then
```

```
    n1 := (n-1) div 2
```

```
  else
```

```
    n1 := n div 2;
```

```
  num :=1; den := 1;
```

```
  for i:= 1 to n1 do
```

```
    begin
```

```
      num := num * (1 - sqr(Wz[i]));
```

```
      if (Wpz[i] <> 0) then
```

```
        den := den * (1 - sqr(1.0/Wpz[i])) ;
```

```
    end;
```

```
  fkl := power(k,n)*sqr(num/den);
```

```
end;
```

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

{ Even function }

Procedure EvenFn(*n* : integer; var *Wz*,*Wpz* : array25;
var *Wpoz*,*Woz* : real);

var

i : integer;
Wz1 : array25;

begin

for *i*:=0 to *n* do

begin

Wz[*i*] :=0; *Wpz*[*i*] :=0;
Wz1[*i*] :=0; *Argp*[*i*] :=0;

end;

Wc := 1;

for *i*:= 1 to *n* div 2 do

begin

argp[*i*] := (2**i* -1)**kk*/*n* ;

ph := Phi(*fi*,*argp*,*xk*,*j*,*i*);

Wz[*i*] := sin(*Ph*); { *sn* = sin(*ph*) }

if(*Wz*[*i*] <> 0) then

Wpz[*i*] :=1/(*k***Wz*[*i*]);

end;

m := *fk1*(*n*,*k*,*Wz*);

{===== END OF CASE A =====}

if not case_a then

begin

Wc := *Wz*[*np_2*] ; *Ws* := *ws*/*Wc*;

WpOz :=*Wpz*[1];

(* { *Ws* :=1/(*k***Wz*[*np_2*]); (ok) *)

for *i*:=1 to *np_2* do { Transform the zeroes }

begin

Wz1[*i*] := ((*sqr*(*WpOz*) -1)* *sqr*(*Wz*[*i*]))/
(*sqr*(*WpOz*) - *sqr*(*Wz*[*i*]));

Wz1[*i*] := *sqr*(*Wz1*[*i*]);

end;

{ Transform the poles }

for *i*:=2 to *np_2* do

begin

Wpz[*i*] := ((*sqr*(*WpOz*) -1)* *sqr*(*Wpz*[*i*]))/
(*sqr*(*WpOz*) - *sqr*(*Wpz*[*i*]));

Wpz[*i*] := *sqr*(*Wpz*[*i*]);

end;

{===== END of CASE B =====}

```

if ( case_c) then
  begin
    Ws := Ws/Wc;
    WOz := Wz1[1];

    for i:=2 to np_2 do          { Transform the zeroe }
      begin
        Wz1[i] := (sqr(Wz1[i]) - sqr(WOz))/
          ( 1 - sqr(WOz));
        Wz1[i] := sqrt(Wz1[i]); Wz1[1] :=0;
      end;

    for i:=1 to np_2 do          { Transform the poles }
      begin
        Wpz[i] := (sqr(Wpz[i]) - sqr(WOz))/
          ( 1 - sqr(WOz));
        Wpz[i] := sqrt(Wpz[i]);
      end;
    end;

    for i:=1 to np_2 do
      begin
        Wz[i] := Wz1[i];
        Wpz[i] := Wpz[i+1];
      end;

    end;    { End case B & C }

{===== END of CASE C ===== }

    { for i := 1 to np_2 do
      writeln( ' Wz ',Wz[i],' wpz ',wpz[i]);
      writeln(' Ws ',Ws, ' k ',k,' 1/ws ',1/ws);
    readln;}
    end;
{-----}

```

(Calculates the poles of the transfer function)

```
Procedure EvenRoots( n: integer;var pcoef: array25x25;
                    var coef: array25;var pr,qi : array25);
```

```
var
```

```
  p      : complex;
  index  : integer;
  re_part,Im_part : real;
  sn     : array2;
```

```
begin
```

```
  index := 1;
```

```
  Max_ord := n;
```

```
  for i:=0 to n do
```

```
    begin
```

```
      qi[i] :=0; pr[i] :=0; coef[i] :=0; pcoef[1,i] :=0;
```

```
      pcoef[0,i] := 0;
```

```
    end;
```

```
  While (index <= (n div 2) ) do
```

```
    begin
```

```
      u := (2 * index - 1)*kk/n; v := a;
```

```
      jac_fun(u,v,k,re_part,Im_part);
```

```
      pr[index] := - Im_part;
```

```
      qi[index] := re_part;
```

```
      index := index +1;
```

```
    end;
```

```
(----- End Roots of Case A -----)
```

```
  if (not case_a) then
```

```
    begin
```

```
      { case B }
```

```
      for i := 1 to np_2 do
```

```
        begin
```

```
          p.re :=pr[i];  p.im :=qi[i];
```

```
          Magp := Mag(p);  Argu := Arg(p);
```

```
          Transf_Root_ToCase_b( pr,qi );
```

```
        end;
```

```
(----- End Roots of Case B -----)
```

```
  if (case_c) then
```

```
    { case C }
```

```
    begin
```

```
      Wpoz := Woz;
```

```
      for i := 1 to np_2 do
```

```
        begin
```

```
          p.re :=pr[i];  p.im :=qi[i];
```

```
          Magp := Mag(p);  Argu := Arg(p);
```

```
          Transf_Root_ToCase_c( pr,qi );
```

```
        end;
```

```
    end;
```

```
(----- End Roots of Case C -----)
```

```
  end;
```

```
  ellip_coef( pr,qi,Max_ord,pcoef,coef);
```

```
end;
```

```
(=====)
```

{Odd functions }

Procedure OddFn(n : integer;var Wz,Wpz :array25);

var

i : integer;

begin

for i:= 0 to n do

begin

Wz[i] :=0; Wpz[i] :=0;

Wz1[i] :=0; Argp[i] :=0;

end;

Wc := 1;

for i:=0 to (n-1) div 2 do

begin

*argp[i] :=(2*i *kk)/n;*

ph := Phi(fi,argp,xk,j,i);

Wz[i] := sin(Ph); {sn = sin(ph) }

if (Wz[i] <> 0) then

*Wpz[i] := 1/(k*Wz[i]);*

end;

m := fkl(n,k,Wz);

end;

(-----)

{Calculates the poles for transfer function (odd case)}

Procedure OddRoots(n: integer;var pcoef :array25x25;

var coef:array25; var pr,qi :array25);

var

index,i : integer;

sign : string[2];

re_part,Im_part : real;

begin

index := 1;

Max_ord :=n; sign := ' +';

argp[1] := a;

for i:=0 to n do

begin

qi[i] :=0; pr[i] :=0; coef[i] :=0;

pcoef[1,i] :=0; pcoef[0,i] := 0;

end;

ellin(xk,elin,k1,fi,j); {k1 -> k' }

ph := Phi(fi,argp,xk,j,1);

p0 := - sin(ph)/cos(ph) ;

pr[np_2+1] := p0; qi[np_2+1] :=0;

(-----)

```

While (index <= (n-1) div 2) do
  begin
    u := 2 * index * kk/n; v := a;
    jac_fun(u,v,k,re_part,Im_part);
    pr[index] := - Im_part;
    qi[index] := re_part;
    index := index +1;
  end;
  ellip_coef(pr,qi,Max_ord,pcoef,coef);

end;

(=====)
  (Calculates the order of the filter)

Procedure CalculateMax_order(var n : integer);
  var n1,fr : real;
  begin
    kkm := F(90,m);
    kkm1 := F(90,m1);
    kk := F(90,k);
    kk1 := F(90,k1);
    n1 := (kk * kkm1)/(kk1*kkm) ;      fr := frac(n1);
    if ( abs(fr) < 0.01) then fr := 0;
    if ( fr <> 0)then
      n := trunc(n1) +1
    else
      n := trunc(n1);
  end;

(=====)
  (calculates the constant H)

  procedure Find_H_eps;

    var
      epspl,Rr : real;

    begin
      if (R2 >= R1) then
        begin
          epspl := 1 + sqr(eps);
          Rr := sqr(R2/R1 +1)/(4*R2/R1);
          H[2] := epspl/Rr ;
        end
      else
        if (R2 <> 0) then
          begin
            epspl := 1 + sqr(eps);
            Rr := sqr(R1/R2 +1)/(4*R1/R2);
            H[2] := epspl/Rr ;
          end;
        end;
      end;

```

```

    if (H[2] > 1) then
      begin
        { Modefy eps }
        if (R2 = R1 ) then
          eps := sqrt(H[2] -1)
        else
          if (R2 > R1 ) then
            eps :=(R2/R1 - 1 )/(2*sqrt(R2/R1))
          else
            eps :=(R1/R2 - 1 )/(2*sqrt(R1/R2));
          H[2] := 1;
          A_max := 10 *Ln(1 + sqrt(eps))/Ln(10);
          Ro :=sqrt(1 - exp(-A_max*Ln(10)/10));
        end;

      if case_b then
        H[1] := - 10* Ln(Rr)/Ln(10)
          { value of transfere function at zero }
      else
        H[1] := 0;

      { H[1] := 0.5 *(sqrt(R2/R1) + sqrt(R1/R2));
        H[1] := - 20*Ln(H[1])/Ln(10);
        A_min := A_min + H[1]; another form }

      end;
    (-----)
    {calculates the elements values for odd network}

    procedure odd_NetWork(Max_ord:integer; Za : carray25; Fa,E : array25;
      R1,R2,Elord : real;var noel : integer;
      var El : array25);

    type
      array25x2 = array[0..25,0..2] of real;
      Str25 = array[1..25] of string[2];

    Var
      nh,i,m,ml,j,k,n_e,n_o,m_e,m_o,largn,largm : integer;
      Z,val_num,val_den : complex;

    label 10,20, 100;
    Var
      den,Anum,Anum1,Aden,Temp,f_e,f_o,E_e,E_o : Array25;
      Zn, Zom : Carray25;

      Nel1 : str25;
      Nel2 : array[1..25] of integer;
      nele,n,nd,ja,jan,nu,nn : integer;
      ab,cx,Yl : real;

```

```

Begin
  nh := Max_ord;
  noel := nh + nh div 2;
  nele := 1; n := nh; nd := (n-1) div 2;

  for i:=0 to n do
  begin
    Zn[i].re := 0; Zom[i].re :=0;
    Nell [i] :=' '; za[i].re :=0;
    El[i] := 0; Temp[i]:=0; Anum[i]:=0;den[i] :=0;
    f_e[i] :=0; F_o[i] :=0; E_e[i] :=0;E_o[i] :=0;
  end;

  if (Elord = 0 ) then Zeros_Seq(za,zn,nd);
  For i :=1 to nd do
  begin
    if (Elord = 1) then Zn[i] := Za[i];
    Zom [i].im := Zn[i].im* Zn[i].im;
  end;

  HrzPoly(n,fa,f_e,f_o,n_e,n_o);
  HrzPoly(n,E,E_e,E_o,m_e,m_o); largm := m_o;
  if ( m_o < n_o ) then largm := n_o; m:= largm;
  for i := 1 to m do
  Anum[i] := F_o[i] - E_o[i]; largn := m_e ;
  if ( m_e < n_e ) then largn := n_e; n := largn;

  for i := 0 to n do
  Den [i] := F_e[i] + E_e[i];

  NormPoly( Anum,m,0.0001);
  NormPoly(den,n,0.0001);
  if ( m < n ) then ExchangPoly( Anum,Den,m,n);
  nu := 0;
  for ja := 1 to nd do
  begin
    valPoly(m,Anum,Zn[ja],val_num);
    valPoly(n,den,Zn[ja],val_den);
    dvd (val_num,val_den,z);
    Dvd (Z,Zn[ja],z);
    nu := nu + 1;
    cx := Z.re;
    El [nele] := cx;
    Nell [nele] := 'C';
    Nel2 [nele] := nu;

    { Check for Large terminating resistance R2 }
    if (R2 >= 1000 ) then goto 10;
    if (nu >= noel div 2 -1) then goto 20;
  end;

```

```

10:      nele := nele + 1;
      Temp[0] :=0; Temp[1] := cx;
      MulPoly(Temp,den,Temp,1,n,j);
      Subtract(Anum,Temp,Anum,m,j,m);
      NormPoly(Anum,m,0.0001);

      ab := Zom [ja].im;
      Temp[0] := ab; Temp[1] :=0; Temp[2] :=1;
      Devpoly (Anum,temp,Anum1,m,2,m1);

      valPoly(m1,Anum1,Zn[ja],val_num);
      dvd (val_num,val_den,Z);
      cx := - Z.im * Zn[ja].im;
      nu := nu +1;
      Y1 := 1.0 /(cx*ab);
      El [nele] := Y1;
      Nel1 [nele] := 'L';
      Nel2 [nele] := nu;
      El [nele+1] := cx;
      Nel1 [nele+1] := 'C';
      Nel2 [nele+1] := nu;
      nele := nele +2;

      { Need Y2 = 1/Z2 }
      Temp[0] :=0; Temp[1] := 1.0/cx;
      MulPoly(Anum1,Temp,Aden,m1,1,j);
      Subtract(den,Aden,den,n,j,n);

      Temp[0] := ab; Temp[1] := 0; Temp[2] :=1;
      DevPoly (den,Temp,den,n,2,n);
      for i:=0 to m1 do
      Anum[i] := Anum1[i];
      m := m1;

      NormPoly(Anum,m,0.0001);
      NormPoly(den,n,0.0001);
      if (m < n) then ExchangPoly(Anum,Den,m,n);

      End; { end ja loop }

      El [nele] := Anum[1]/den[0]; {for the case where R2 >1000}
      Nel1 [nele] := 'C';
      Nel2 [nele] := nh;
      goto 100;

```

```

20:  nele := noel; nu := nh;

    for i:=0 to nh do
      begin
        Anum[i] :=0; Den [i] :=0;
        Anum1[i] :=0; Aden[i] :=0;
      end;
      n := largn; m := largm;
      for i := 1 to m do
        Anum[i] := (F_o[i] + E_o[i]);
        for i := 0 to n do
          Den [i] := R2*(F_e[i] + E_e[i]);

        NormPoly(Anum,m,0.0001);
        NormPoly(den,n,0.0001);
        if (m < n) then ExchangPoly(Anum,Den,m,n);

      for jan := 1 to nd do
        begin

          ja := - jan + nd +1;

          valPoly(m,Anum,Zn[ja],val_num);
          valPoly(n,den,Zn[ja],val_den);
          dvd (val_num,val_den,z);
          Dvd (Z,Zn[ja],z);
          cx := Z.re;
          El [nele] := cx;
          Nel1 [nele] := 'C';
          Nel2 [nele] := nu;

          nele := nele - 1;
          nu := nu -1;
          Temp[0] :=0; Temp[1] := cx;
          MulPoly(Temp,den,Temp,1,n,j);
          Subtract( Anum,Temp,Anum,m,j,m);

          NormPoly( Anum,m,0.0001);

          ab := Zom [ja].im;
          Temp[0] :=ab; Temp[1] :=0; Temp[2] :=1;
          DevPoly( Anum,Temp,Anum1,m,2,m1);

```

```

valPoly(m1,Anum1,Zn[ja],val_num);
( valPoly(n,den,Zn[ja],val_den); )
dvd (val_num,val_den,Z);
cx := - Z.im * Zn[ja].im;

```

```

Yl := 1.0 /(cx*ab);
El [nele-1] := Yl;
Nel1 [nele] := 'C';
Nel2 [nele] := nu;
El [nele] := cx;
Nel1 [nele-1] := 'L';
Nel2 [nele-1] := nu;
nele := nele -2;
nu := nu -1;
if ( nu <= ( noel div 2) ) then goto 100;

```

```

Temp[0] :=0; Temp[1] := 1.0/cx;
MulPoly(Anum1,Temp,Aden,m1,1,j);
Subtract(Den,Aden,Den,n,j,n);

```

```

Temp[0] := ab; Temp[1] := 0; Temp[2] := 1;
Devpoly (Den,Temp,Den,n,2,n);
For i:= 0 to m1 do
  Anum[i] := Anum1[i];
  m := m1;

```

```

NormPoly(Anum,m,0.0001);
NormPoly(den,n,0.0001);

```

```

End;      ( end jan loop )

```

```

100 : writeln; {
  For i := 1 to noel do
    Writeln(' ':2,Nel1[i],nel2[i],' = ',El[i]);
  }
End;      ( end odd_netWork)

```

```

{-----}

```

(Calculates the Elements values for Even networks)

```
procedure Even_NetWork( nh : integer; Za : Carray25; Fa,E : array25;
                       R1,R2,lamda,ro,Elord : real;case_c : Boolean;
                       var noel : integer; var El : array25);
```

```
type
  array25x2 = array[0..25,0..2] of real;
  Str25 = array[1..25] of string[2];
```

```
Var
  m,m1,k,n_e,n_o,m_e,m_o,largn,largm : integer;
  Z,val_num,val_den : complex;
```

```
label 10,20, 100;
```

```
Var
  den,Anum,Anum1,Aden,Temp,f_e,f_o,E_e,E_o : Array25;
  Zn, Zom : Carray25;
  Nel1 : str25;
  Nel2 : array[1..25] of integer;
```

```
nele,n,nd,j,ja,jan,i,nu,nn,nda,cnt : integer;
ab,cx,Yl,usq : real;
```

```
Begin
```

```
noel := nh + (nh-1) div 2;
nele := 1; n := nh; nd := n div 2; nda := nd-1;
```

```
for i:=1 to n do
```

```
begin
  Zn[i].re := 0; Zom[i].re :=0;
  Nel1 [i] :=' '; za[i].re :=0;
```

```
end;
```

```
for i:=0 to n do
```

```
begin
  El[i] := 0; Temp[i]:=0; Anum[i]:=0;den[i] :=0;
  f_e[i] :=0; F_o[i] :=0; E_e[i] :=0;E_o[i] :=0;
end;
```

```
if (Elord = 0 ) then Zeros_seq(za,zn,nda);
```

```
For i :=1 to nda do
```

```
begin
```

```
if (Elord = 1) then Zn[i] := Za[i];
  Zom [i].im := Zn[i].im* Zn[i].im;
```

```
end;
```

```
  HrzPoly(n,fa,f_e,f_o,n_e,n_o);
```

```
  HrzPoly(n,E,E_e,E_o,m_e,m_o); largm := m_e;
```

```
  if ( m_e < n_e ) then largm := n_e; m:= largm;
```

```
  for i := 0 to largm do
```

```
    Anum[i] := F_e[i] - lamda * E_e[i]; largn := m_o ;
```

```
  if ( m_o < n_o ) then largn := n_o; n := largn;
```

```

    for i := 1 to largn do
        Den [i] := F_of[i] + lamda * E_of[i];

Normpoly(Anum,m,1.0E-3);
Normpoly(den,n,0.001);
if ( m < n ) then Exchangpoly(Anum,Den,m,n);

nu := 0;
for ja := 1 to nda do
    begin
        valPoly(m,Anum,Zn[ja],val_num);
        valPoly(n,den,Zn[ja],val_den);
        dvd (val_num,val_den,z);
        Dvd (Z,Zn[ja],z);
        nu := nu + 1;
        cx := Z.re;
        El [nele] := cx;
        Nel1 [nele] := 'C';
        if((R2 > 1) and (R2 < 1000)) then Nel1 [nele] := 'L';
        Nel2 [nele] := nu;
        Nele := Nele + 1;

( Check for Large terminating resistance R2 )
        if (R2 >= 1000 ) then goto 10;
        if (nu >= nh div 2) then goto 20;

10:        Temp[0] :=0; Temp[1] := cx;
            MulPoly(Temp,den,Temp,1,n,j);
            Subtract(Anum,Temp,Anum,m,j,m);
            NormPoly(Anum,m,0.0001);

            ab := Zom [ja].im;

Temp[0] := ab; Temp[1] := 0; Temp[2] := 1;
DevPoly(Anum,Temp,Anum1,m,2,m1);

valPoly(m1,Anum1,Zn[ja],val_num);
dvd (val_num,val_den,Z);
cx := - Z.im * Zn[ja].im;
nu := nu +1;
Yl := 1.0 /(cx*ab);
El [nele] := Yl;
Nel1 [nele] := 'L';
if ((R2 > 1) and (R2 < 1000)) then Nel1[nele] := 'C';
Nel2 [nele] := nu;
El [nele+1] := cx;
Nel1 [nele+1] := 'C';
if ((R2 > 1) and (R2 < 1000)) then Nel1[nele+1] := 'L';
Nel2 [nele+1] := nu;

```

```

if (nh = 4) then goto 20;
nele := nele +2;

    { Need Y2 = 1/Z2 }
Temp[0] :=0; Temp[1] := 1.0/cx;
MulPoly(Anum1,Temp,Aden,m1,1,j);
Subtract(den,Aden,Den,n,j,n);

Temp[0] := ab; Temp[1] := 0; Temp[2] := 1;
Devpoly (Den,Temp,den,n,2,n);
For i := 0 to m1 do
    Anum[i] := Anum1[i];
    m := m1;

NormPoly(Anum,m,0.001);
NormPoly(den,n,0.001);
if (m < n ) then Exchangpoly(Anum,Den,m,n);

End:  { end ja loop }

```

```

    { For the case whetre R >= 1000 }
El [nele] := Anum[2]/den[1];
El [nele+1] := Den[1]/Anum[0];
Nel1 [nele] := 'C';
Nel1 [nele+1] := 'L';
Nel2 [nele] := nh-1;
Nel2 [nele+1] := nh;
goto 100;

```

```

20:     nele := noel; nu := nh;
        cnt :=1;
        for i :=0 to nh do
            begin
                Anum[i] := 0; Den[i] := 0;
                Anum1[i] :=0; Den[i] := 0;
            end;
        usq := (1-ro)/(1+ro);
        if (case_c) then
            begin
                usq := 1;
            end;
        if(usq > 1 ) then usq := 1.0/usq;
            n := largn; m := largm;
            for i := 0 to m do
                Anum[i] := (F_e[i] + lamda * E_e[i]);
            for i := 1 to n do
                Den [i] := (F_o[i] + lamda * E_o[i]);

Normpoly(Anum,m,0.0001);
Normpoly(den,n,0.0001);

```

```
if ( m < n ) then Exchangpoly( Anum,Den,m,n);
```

```
Yl := Anum[m]*usq/den[n] ;
EL[nele] :=YL;
Nel1 [nele] := 'L';
if (R2 > 1 ) then Nel1 [nele] := 'C';
Nel2 [nele] := nu;
nele := nele -1;
```

```
for i:=1 to m do begin { Remove pole at infinity }
  Anum[i] := Anum[i]*usq - Yl*den[i-1];
end;
  Anum[0] := usq*Anum[0];
  NormPoly( Anum,m,0.0001);
```

```
(=====)
```

```
if ( m < n ) then Exchangpoly( Anum,Den,m,n);
```

```
for jan := 1 to nda do
begin
```

```
  ja := nd - jan ;
  nu := nu - 1;
  valPoly(m,Anum,Zn[ja],val_num);
  valPoly(n,den,Zn[ja],val_den);
  dvd (val_num,val_den,z);
  Dvd (Z,Zn[ja],z);
  cx := Z.re;
  El [nele] := cx;
  Nel1 [nele] := 'C';
```

```
  if (R2 > 1 ) then Nel1 [nele] := 'L';
  Nel2 [nele] := nu;
  cnt := cnt +1;
  if (cnt >= (Noel div 2)) then goto 100;
  nu := nu -1;
  nele := nele - 1;
```

```
(=====)
```

```
Temp[0] :=0; Temp[1] := cx;
MulPoly(Temp,den,Temp,1,n,j);
Subtract( Anum,Temp,Anum,m,j,m);
```

```
ab := Zom [ja].im;
Temp[0] := ab; Temp[1] := 0; Temp[2] := 1;
DevPoly( Anum,Temp,Anum1,m,2,m1);
```

```
valPoly(m1,Anum1,Zn[ja],val_num);
valPoly(n,den,Zn[ja],val_den);
```

```
dvd (val_num,val_den,Z);
```

```
cx := - Z.im * Zn[ja].im;
Yl := 1.0 / (cx*ab);
El [nele] := cx;
Nel1 [nele] := 'C';
if (R2 > 1 ) then Nel1 [nele] := 'L';
Nel2 [nele] := nu;
Nele := Nele -1;
El [nele] := YL;
Nel1 [nele] := 'L';
if (R2 > 1 ) then Nel1[nele] := 'C';
Nel2 [nele] := nu;
cnt := cnt +2;
if (cnt >= ( Noel div 2)) then goto 100;
nele := nele -1;
```

```
Temp[0] :=0; Temp[1] := 1.0/cx;
MulPoly(Anum1,Temp,Aden,m1,1,j);
Subtract(den,Aden,Den,n,j,n);
```

```
Temp[0] := ab; Temp[1] := 0; Temp[2] := 1;
Devpoly (den,Temp,den,n,2,n);
for j:=0 to m1 do Anum[j] := Anum1[j];
m := m1;
```

```
NormPoly(Anum,m,0.001);
NormPoly(den,n,0.001);
```

```
End; { end jan loop }
```

```
100 : writeln; {Write_Elements;}
{ For i := 1 to noel do
  Writeln(' ':2,Nel1[i],nel2[i], ' = 'El[i]); }
  if ( not case_c ) then R2 := usq;
End; { end Even_netWork }
```

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

(Print the output to screen)

```

Procedure Print_out(out_flag : Str4);
label 10, 20;
var
  index,i,ni,nf,j : integer;
  sign             : string[2];
  text80          : string[80] ;

  procedure input_output;
  begin
    box(59,2,78,6,1);
    highvideo;
    gotoxy(65,2); Write( 'n = ',n);
    Gotoxy(60,3); Write('Ws = ',Ws:-2:5, ' r/s');
    gotoxy(60,4); Write('A_max= ',A_max:-3:5.' db');
    gotoxy(60,5); Write('A_min=',alpha:-5:5.' db');
  end;
begin
  Text80 :=' DENOMINATOR ' ;
  ClrScr; col :=2; raw := 4;

  if odd(n) then
    begin
      ni :=0; nf :=n-1;
      col := col +1
    end
  else
    begin
      ni := 1; nf := n;
      col := col +1;
    end ;
  if (out_flag = 'Den' ) then goto 10;
  Gotoxy(15,col);Bright(' Zeros of the transfer function ');
  col := col +1;
  for i :=ni to n_zeros do
    begin
      if (Wpz[i] <> 0 ) then
        if odd(i) then
          begin
            Gotoxy(raw,col); writeln(' Wz',i,'=',Wpz[i]);
            col := col -1;
          end
        else
          begin
            Gotoxy(30,col); writeln(' Wz',i,'=',Wpz[i]);
          end;
      col := col +1;
    end;
  if (out_flag = 'Num') then goto 20;

```

```

10:  col := col +3; input_output;
      Gotoxy(2,col );
      writeln('Actual attenuation in the stopband = ',alpha:-3:10,' db');
      if odd(n) then col := col + 1 Else col := col+2;

      index := 1; Max_ord :=n; sign :=' +';
      Gotoxy(2,col);for i:= 1 to 78 do write(#196);
      highvideo; col :=col +1; Gotoxy(40,col);
      Write('Quadratic factors of s**2 + a1*s + a2 ');
      col := col+1; Gotoxy(20,col);
      writeln(' POLES a1 a2');
      col :=col +1;
      Gotoxy(2,col);for i:= 1 to 78 do write(#196);
      col := col +1; lowvideo;
      if odd(n) then
        begin
          gotoxy(1,col); Writeln(pr[np_2+1]);
          col := col +1;
        end;
      While (index <= (nf div 2) ) do
        begin
          Gotoxy(1,col);
          Writeln(pr[index],sign,'j',abs(qi[index]));
          index := index +1; col := col+1;
        end;
20 :  if( out_flag = 'Num') then
        begin
          Text80 := ' NUMERATOR ';
          col := wherey+1 ;
          Gotoxy(2,col);for i:= 1 to 78 do write(#196);
          col := col +1; gotoxy(2,col);
          Write('Quadratic factors of s**2 + a1*s + a2 ');
          Gotoxy(40,col);
          writeln(' a1 a2');
          col := wherey +np_2; lowvideo;
        end;
      for i:=1 to n div 2 do
        begin
          gotoxy(40,col-np_2);write(abs(pcoef[1,i]));
          Gotoxy(60,col-np_2);write(abs(pcoef[0,i]));
          col := col +1;
        end;
      col := col - np_2+1; highvideo;
      Gotoxy(2,col-1);for i:= 1 to 78 do write(#196);
      {-----}
      if ( max_ord >=10 ) then
        begin
          box(1,1,79,25,1);
          col :=4; raw :=2; WaitKey;
          ClrScr;
        end;
      {-----}

```

```

Gotoxy(raw,col);
Write(' Coefficients of the polynomial r(s) for the '.text80);
col := col +1; Gotoxy(raw,col);
Write(' Associated with the Elliptic Respnse. ');
Gotoxy(12,col+1);
Write('r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n');
col := col +2;
raw := 2;
Gotoxy(raw,col);for i:= 1 to 78 do write(#196);writeln;
col := col +1; lowvideo;
gotoxy(raw,col);
j := 0;
for i :=0 to n do
  begin
    j := j +1;
    if (j <=3 ) then write(' a',i,' = ',coeff[i]);

    if (j=3) then
      begin
        j := 0; col := col +1;
        Gotoxy(raw,col);
      end;
  end;

box(1,1,79,25,1);
WaitKey;
ClrScr;
end;

{-----}

```

```

Procedure Write_Element(n_of_el : integer; Element : array25);

var
  i,raw,col,j,jj  : integer;

begin
  col :=2; raw := 3;
  ClrScr;

  { gotoxy(raw,col); for i:= 1 to 75 do write(#196); col := col +1;}
  gotoxy(raw,col);
  write(' R1 = ',R1:-3:3,' R2 = ',R2,' Const. H = ',H[1]);
  col := col +1;
  gotoxy(raw,col); for i := 1 to 75 do write(#196); col := col +1;

  gotoxy(raw-1,col);
  write(' k ' , ':10,'C(2k-1)',':15,'L(2k)',':20, 'C(2k)');
  col := col +1;
  gotoxy(raw,col); for i := 1 to 75 do write(#196); col := col +1;
  gotoxy(raw,col);
  i := 1;
  j := 0; jj := 1;
  while( i <= n_of_el) do
    begin
      if (i=1) then begin gotoxy(raw,col); write(1); end;
      j := j+1 ;
      if (j <=3) then Write(' ':5 ,Element[i]);

      if j= 3 then
        begin
          j := 0; jj := jj +1; col := col +1;
          gotoxy(raw,col); write(jj);
        end;
      i := i +1;
    end;
  gotoxy(raw,col+1); for i:= 1 to 75 do write(#196);

end;

(-----)

```

{Print output to a file or send it into the printer}

```

    Procedure Print_out_toDev(out_flag : Str4);
label 10, 20;
var
    index,i,ni,nf,j : integer;
    sign             : string[2];
    text80           : string[80] ;

    procedure input_output;
    begin
        write(out_dev, ' n = ',n);
        writeln(out_dev,' Ws = ',Ws,-2:5, ' r/s');
        write(out_dev, ' A_max= ',A_max,-3:5,' db');
        writeln(out_dev,' A_min=',alpha,-5:5,' db');
    end;

begin
    Text80 :=' DENOMINATOR ' ;
    if odd(n) then
        begin
            ni :=0; nf :=n-1;
        end
    else
        begin
            ni := 1; nf := n;
        end ;

    if (out_flag = 'Den' ) then goto 10;
    Writeln(out_dev);
    for i:= 1 to 80 do write(out_dev,ch); writeln(out_dev);
    writeln(out_dev);
    Writeln(out_dev,' Zeros of the transfer function ');
    writeln(out_dev); j:=0;
    for i :=ni to n_zeros do
        begin
            if (Wpz[i] <> 0 ) then
                if odd(i) then
                    write(out_dev,' Wz',i,'=',Wpz[i])
                else
                    if (j<=2) then
                        writeln(out_dev,' Wz',i,'=',Wpz[i]);
                    if(j=0) then j:=0;
                end;
            { end loop i }

        if (out_flag = 'Num') then goto 20;

10:    input_output;
        writeln(out_dev);
        writeln(out_dev,' :2,
'Actual attenuation in the stopband = ',alpha,-3:10,' db');

```

```

writeln(out_dev);

index := 1;    Max_ord :=n;    sign :=' +';
for i:= 1 to 80 do write(out_dev,ch); writeln(out_dev);

write(out_dev,' ':40,
'Quadratic factors of s**2 + a1*s + a2 ');

writeln(out_dev, ' ':20,
' POLES          a1          a2');

for i:= 1 to 80 do write(out_dev,ch); writeln(out_dev);
if odd(n) then writeln(out_dev,pr[np_2+1]);

While (index <= (nf div 2) ) do
begin
write(out_dev,pr[index],sign,'j',abs(qi[index]));
i := index;
write(out_dev,abs(pcoef[1,i]));
writeln(out_dev,abs(pcoef[0,i]));
index := index +1;
end;

20 : if( out_flag = 'Num') then
begin
writeln(out_dev);
Text80 := ' NUMERATOR ';
for i:= 1 to 80 do write(out_dev,ch); writeln(out_dev);
writeln(out_dev,
'Quadratic factors of s**2 + a1*s + a2 ',Text80);
writeln(out_dev,' ':40,
'          a1          a2');
writeln;

for i:=1 to n div 2 do
begin
write(out_dev,' ':40,abs(pcoef[1,i]));
writeln(out_dev,abs(pcoef[0,i]));
end;
end;
writeln(out_dev);
for i:= 1 to 80 do write(out_dev,ch); Writeln(out_dev);
(-----)

writeln(out_dev,
' Coefficients of the polynomial r(s) for the ',text80);

writeln(out_dev,' Associated with the Elliptic Respse. ');
writeln(out_dev, ' ':2,
'r(s)= a0 + a1*s+ a2*s**2 +...+ a_(n-1)*s**(n-1) + s**n');

```

```

for i:= 1 to 80 do write(out_dev,ch);writeln(out_dev);
  j := 0;
  for i :=0 to n do
    begin
      j := j +1;
      if (j <=3 ) then write(out_dev,' a',i,' = ',coeff[i]);
      if (j=3) then
        begin
          j := 0; writeln(out_dev);
        end;
    end;
  end;
end;
{----- write element values to device} -----}

Procedure Write_ElementDev(n_of_el : integer; Element : array25);

var
i,raw,col,j,jj : integer;

begin
  Writeln(out_dev); writeln(out_dev);
  for i := 1 to 80 do write(out_dev,ch); writeln(out_dev);
  writeln(out_dev,' R1 = ',R1:~3:3,' R2 = ',R2,' Const. H = ',H[1]);

  for i := 1 to 80 do write(out_dev,ch); writeln(out_dev);

  write(out_dev,
' k ' , ' :10,'C(2k-1)', ' :15,'L(2k)', ' :20, 'C(2k)');
  Writeln(out_dev);
  for i := 1 to 80 do write(out_dev,ch); writeln(out_dev);

  i := 1;
  j := 0; jj := 1;
  while( i <= n_of_el) do
    begin
      if (i=1) then begin write(out_dev,1); end;
      j := j+1 ;
      if (j <=3) then write(out_dev,' :5 ,Element[i]);

      if j= 3 then
        begin
          j := 0; jj := jj +1; writeln(out_dev);
          write(out_dev,jj);
        end;
      i := i +1;
    end;
  writeln(out_dev);
  for i:= 1 to 80 do write(out_dev,ch); writeln(out_dev);
end;
{-----}

```

{ Help }

Procedure Help(win_n ,Help_level : integer);

var

x :text80;

raw, col : integer;

Begin

Case Help_level of

1 : Begin

Wopen (win_n);

ClrScr;

raw := 7; col := 5;

FastWrite(25,col-2,\$09,' Combination A ');

x := ' 1) Stopband frequency (rad/s), or the Modular angle ';

FastWrite(raw,col,\$07,x); gotoxy(54,col-2);

write(chr(233),' (in deg.)');

x := ' 2) Ripples in the PassBand (A_max (db), or the ref. coef, Ro.');

FastWrite(raw,col+1,\$07,x);

x:= ' 3) Attenuation in the stopband (A_min (db).');

FastWrite(raw,col+2,\$07,x);

col := 10;

FastWrite(25,col-1,\$09,' Combination B ');

FastWrite(raw,col+1,\$07,' 1) Stopband frequency (Ws).');

fastWrite(raw,col+2,\$07,' 2) Ripples in the PassBand A_max (dB).');

fastWrite(raw,col+3,\$07,' 3) The order of the filter (n).');

col := 16;

FastWrite(25,col-1,\$09,' Combination C');

FastWrite(raw,col+1,\$07,' 1) Stopband frequency Ws. ');

FastWrite(raw,col+2,\$07,' 2) Attenuation in the stopband, A_min (dB).');

FastWrite(raw,col+3,\$07,' 3) The order of the filter (n).');

End;

2: Begin

Wopen (win_n);

ClrScr;

Gotoxy(2,2); Write('Note: this message here');

end;

End; { end case }

FastWrite(28,21,\$19,' Press Any Key To Continue ');

gotoxy(35,col+5); Repeat until keyPressed;

End;

{-----}

REFERENCES

- [1] W. K. Chen, *Theory and Design of Broadband Matching Networks*. New York: Pergamon Press, 1976, Chapter 3.
- [2] L. P. Huelsman and P. E. Allen, *Introduction to the Theory and Design of Active Filters*. New York: McGraw-Hill, 1980, Chapter 2.
- [3] D. J. Baez Lopez, *Sensitivity and Synthesis of Elliptic Functions*. Ph. D. dissertation, University of Arizona, Tucson, 1978.
- [4] R. Sall, *Handbook of Filter design*. AEG Telefunken, 1979.
- [5] Anatol I Zverev, *Handbook of Filter Synthesis*. New York: John Wiley & Sons, 1967.
- [6] H. Baher, *Synthesis of Electrical Networks*. New York: John Wiley & Sons, 1984.
- [7] J.D. Rhodes, *Theory of Electrical Filters*, New York: John Wiley & Sons, 1976.
- [8] R. Sall U. E. Ulbrich, " On the Design of filters by synthesis", *IRE Trans. Circuit theory* CT4 (1958) S. 284-327.
- [9] A. S. Sedra and P. O. Brackett, *Filter Theory and Design: Active and Passive*, Champaign, ILL: Matrix Publishers, 1978, Chapter 7.