

**Analysis and Software Implementation of
Two Approximation Methods
for Bandpass Filters**

by

Yiu Kuen Wong

A Thesis

*Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of*

MASTER OF SCIENCE

*Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba
Canada*

September, 1993

© Yiu Kuen Wong, 1993



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-86080-4

Canada

Name YIU KUEN WONG

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ELECTRICAL ENGINEERING

SUBJECT TERM

0544

U·M·I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0729
Art History	0377
Cinema	0900
Dance	0378
Fine Arts	0357
Information Science	0723
Journalism	0391
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0727
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0998
Physical	0523

Psychology	0525
Reading	0535
Religious	0527
Sciences	0714
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578

Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0332
Canadian	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	0317
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760

Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleoecology	0426
Paleontology	0418
Paleozoology	0985
Palynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0350
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575

Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences	
Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Science	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463

Applied Sciences

Applied Mechanics	0346
Computer Science	0984

Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0348
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0621
Behavioral	0384
Clinical	0622
Developmental	0620
Experimental	0623
Industrial	0624
Personality	0625
Physiological	0989
Psychobiology	0349
Psychometrics	0632
Social	0451



ANALYSIS AND SOFTWARE IMPLEMENTATION OF TWO
APPROXIMATION METHODS FOR BANDPASS FILTERS

BY

YIU KUEN WONG

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba
in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

© 1993

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA
to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to
microfilm this thesis and to lend or sell copies of the film, and LIBRARY
MICROFILMS to publish an abstract of this thesis.

The author reserves other publication rights, and neither the thesis nor extensive
extracts from it may be printed or other-wise reproduced without the author's written
permission.

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Yiu Kuen WONG

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Yiu Kuen WONG

Abstract

In this thesis, two approximation techniques which generate the characteristic functions of bandpass filters are investigated and implemented in Pascal language. The techniques described are not based on the usual frequency transformations from a lowpass filter. The first algorithm consists of two stages: an interpolation stage for the approximation by zeros, and a pole-shifting stage for obtaining a new position for the poles. The second algorithm uses two transformed variables to achieve a simple representation of the characteristic function. The efficient Remez algorithm is modified and applied in the optimization process.

Some examples which contain multi-level bounds in the stopbands as well as in the passband, have been used to test the algorithms. The results are considered acceptable and the degrees of the resulting functions are as low as possible.

These algorithms were implemented on a Macintosh II computer in double-precision form.

Acknowledgements

I would like to express my sincere gratitude to Professor G. O. Martens for his supervision and encouragement during the research of this thesis and throughout the M. Sc. program.

I would also like to thank my colleagues V. Cheng, M. F. Ng, and especially T. K. Chung for many informative discussions. Many thanks to all my friends for their supports all through these years, especially Buggle and Cecilia.

A heartfelt thanks to my parents, they have brought me to this day and taught me the lessons of life. They also give me the wisdom to overcome all the difficulties in this work.

Most of all, I wish to dedicate this thesis to my one and only love,

Jeanny Yin Ha MAK.

I realize that most of the time you did not know what I was doing or why it took so long but you have your faith on me. Without your continuous and unquestionable support, I will never complete this thesis.

Table of Contents

Abstract	iv
Acknowledgements	v
Table of Contents	vi
List of Figures	viii
Chapter	page
1 Introduction	1
2 Background Information	4
2.1 Feldtkeller Equation	4
2.2 Bandpass Filter	5
2.3 Remez Algorithm	8
3 Squared Magnitude Approximation	10
3.1 Specification	10
3.2 The Cohen Algorithm	14
3.3 The Interpolation Stage	16
3.4 Polynomial Interpolation	18
3.5 The Pole-Shifting Stage	19
3.6 Epilogue	22
4 Attenuation Approximation	24
4.1 Specifications	24
4.2 The Transformed Variable	26
4.3 The Devleeschouwer and Grenez Algorithm	29

5 Experimental Results and Discussion	37
5.1 Program Development of Cohen Algorithm.....	37
5.2 Program Development of Devleeschouwer and Grenez Algorithm.....	39
5.3 Application of Cohen Algorithm.....	43
5.4 Application of Devleeschouwer and Grenez Algorithm.....	53
5.5 Discussion.....	67
6 Conclusions	68
Appendix A: Program Listing of Cohen Algorithm	70
Appendix B: Program Listing of Devleeschouwer and Grenez Algorithm	123
References	140

List of Figures

Figure 2.1: A simple bandpass filter	6
Figure 2.2: Frequency transformation of lowpass to bandpass filter	6
Figure 2.3: Lowpass to bandpass filter with asymmetric requirements.....	7
Figure 2.4: Remez algorithm.....	9
Figure 3.1: $f(x)$ for a bandpass filter	11
Figure 3.2: Rational function resulting from Interpolation stage.....	14
Figure 3.3: Rational function resulting from Interpolation stage.....	15
Figure 3.4: Discrepancies (overshoots) resulting at the k^{th} interpolation :.....	17
Figure 3.5: Newton's Interpolation Algorithm	18
Figure 3.6: The Cohen Algorithm	23
Figure 4.1: x - y mapping for the bandpass filter.....	27
Figure 4.2: The iterative procedure	31
Figure 4.4: The Devleeschouwer & Grenez algorithm	36
Figure 5.1: Zeros outside stopbands.....	39
Figure 5.2: An arc crosses between two stopbands.....	40
Figure 5.3: Existence of x_0 , x_∞ , or both	41
Figure 5.4a: Stopband attenuation of the characteristic function from example 1.1	44
Figure 5.4b: Passband attenuation of the characteristic function from example 1.1	44
Figure 5.5a: Stopband attenuation of the characteristic function from example 1.2	46
Figure 5.5b: Passband attenuation of the characteristic function from example 1.2	46

Figure 5.6a: Stopband attenuation of the characteristic function from example 1.3	48
Figure 5.6b: Passband attenuation of the characteristic function from example 1.3	48
Figure 5.7a: Stopband attenuation of the characteristic function from example 2.1	50
Figure 5.7b: Passband attenuation of the characteristic function from example 2.1	50
Figure 5.8a: Stopband attenuation of the characteristic function from example 2.2	52
Figure 5.8b: Passband attenuation of the characteristic function from example 2.2	52
Figure 5.9a: Stopband attenuation of the characteristic function from example 3.1	54
Figure 5.9b: Passband attenuation of the characteristic function from example 3.1	54
Figure 5.10a: Stopband attenuation of the characteristic function from example 3.2	56
Figure 5.10b: Passband attenuation of the characteristic function from example 3.2	56
Figure 5.11a: Stopband attenuation of the characteristic function from example 3.3	58
Figure 5.11b: Passband attenuation of the characteristic function from example 3.3	58
Figure 5.12a: Stopband attenuation of the characteristic function from example 3.4	60
Figure 5.12b: Passband attenuation of the characteristic function from example 3.4	60
Figure 5.13a: Stopband attenuation of the characteristic function from example 3.5	62
Figure 5.13b: Passband attenuation of the characteristic function from example 3.5	62
Figure 5.14a: Stopband attenuation of the characteristic function from example 4.1	64
Figure 5.14b: Passband attenuation of the characteristic function from example 4.1	64
Figure 5.15a: Stopband attenuation of the characteristic function from example 4.2	66
Figure 5.15b: Passband attenuation of the characteristic function from example 4.2	66

Chapter 1

Introduction

The design of linear, time-invariant and stable lumped-element filters is one of the most important subjects in classical network theory. This kind of network design is normally based on the determination of real rational transfer functions. This requires the design specifications to be given in the form of prescriptions for the transfer behaviour of the network. For a fixed degree, the parameters of the transfer function (constant multiplier, zeros and poles) serve as the design parameters to fulfill the design specification. In order to guarantee the realizability of the network, the transfer function must be free of poles in the half-plane $\text{Re } s \geq 0$ including the imaginary axis of the complex s -plane.

A characteristic of the filter design problem is the kind of specification which always must be seen in view of realizability. Most of the time, a filter is designed on the basis of its attenuation characteristic only. It is specified in the form of a continuous, finite and non-negative function in a real frequency variable over the total frequency domain.

While ultimately a real rational transfer function is required, usually an approximation technique is used to first obtain a real rational characteristic function. The characteristic function has the maximum possible number of equal (or nearly equal) ripples in a passband and exceeds some frequency dependent requirement in two stopbands. The complete solution of this approximation problem has a general and a particular part: the description of the form of the characteristic function, which has given poles and is equal (or nearly equal) ripple in the passband, and the choice of the number and position of these poles to satisfy any given requirement. The ease with which the second part can be performed in a particular case should be an important factor in the choice of the form of the characteristic function [1].

In most filter applications, the synthesis of LC filters implies an important asymmetry between the number of capacitors and the number of inductors. Since inductors are larger, less easily implemented, more expensive, and more dissipative than capacitors, much

attention has been given to the problem of designing bandpass filters with minimum numbers of inductors. This minimum is achieved when a filter with an even number, $2N$, of degrees of freedom can be realized as a reactance ladder network containing N inductors. The degree of the approximating function is also an essential parameter. It must be chosen sufficiently high to meet the tolerance specifications. On the other hand, economic considerations dictate that it be as low as possible.

Traditionally, a bandpass filter function is obtained from the optimal lowpass function through a frequency transformation. This restricts the selection of the optimum to the set of functions obtainable through such a transformation. Also this transformation cannot be applied to the frequency-asymmetrical bandpass filter, i.e., a bandpass filter whose attenuation is not symmetrical on a log-frequency scale.

The objective of this thesis is to investigate two approximation techniques which generate the characteristic function of the bandpass filter without the frequency transformation. Also, the approximation techniques are implemented in a computer program. The classical approximation problem is described very briefly in [2], [3]. The discussion is limited to lowpass or bandpass filters with an equiripple or maximally flat passband response. Moreover, functions used in the design of filters have often been characterized as approximations to certain ideal curves known to be nonrealizable in practice [3], [4]. A proper characterization, however, would have been the selection of a function from a collection of allowable ones, fitting within certain bounds and resulting in a physical electric circuit of minimal complexity [5].

Both of the approximation techniques can be applied on any finite number of suitable bounds in the passband as well as in the stopbands. The well-known Remez algorithm [6] is used in modified form in both techniques for the iterative procedure. Chapter 2 will provide more detail about this algorithm. A discussion of the limitation of the frequency transformation will be provided as well. Also the general background of the characteristic function in filter design will be included.

Chapter 3 describes the squared magnitude approximation developed by Cohen [7]. This approximation combines the polynomial interpolation method and the pole shifting process to achieve an optimal characteristic filter function. Newton's Interpolation Algorithm is used for the first part of the interpolation problem.

Techniques based on linear programming [8], [9] or nonlinear programming [10], [11] have been proposed in order to remove the restriction on equiripple or maximally flat passband response. Chapter 4 describes the attenuation approximation developed by Devleeschouwer and Grenez [12]. The algorithm describes an iterative approximation process, somewhat similar to Remez's second method [13], for an arbitrary stopband specification in the Chebychev sense. A new transformed variable is introduced to achieve accuracy near the passband edge. At the end of each of these two chapters, a brief description of the algorithm flowchart is given.

All polynomials in this thesis are expressed in product form. This is because of its low error sensitivity compared with polynomials which are in coefficient form [14].

Throughout the interpolation process, polynomials are kept in product form by applying the zero-finding algorithm. The details of the algorithm are not discussed in this thesis.

References to this are given as required.

Subsequently, experimental results and discussion are provided in Chapter 5. Several examples are presented to illustrate the effectiveness of both approximation techniques. Finally, Chapter 6 summarizes the work presented in this thesis and presents topics for future research.

Chapter 2

Background Information

2.1 Feldtkeller Equation

A lossless two-port network is characterized by its wave scattering parameters. The scattering matrix

$$S = \frac{1}{g} \begin{bmatrix} h & \sigma f_* \\ f & -\sigma h_* \end{bmatrix}$$

where g , h , and f are the canonic polynomials of the complex frequency variable s and $\sigma = \pm 1$. The analytic continuation of Feldtkeller Equation is

$$gg_* = hh_* + ff_* \quad (2.1)$$

The lower asterisk ‘ $*$ ’ indicates Hurwitz conjugation. For real polynomials this corresponds to replacing s by $-s$ and is equal to the complex conjugation for $s = j\omega$; i.e., $f_*(s) = f(-s)$. According to [15–17], the polynomial g from (2.1) must be a Hurwitz polynomial; i.e., a polynomial with its zeros belonging to the left-half of the complex plane. Also, the polynomial f is monic as well, i.e., the leading coefficient of f is unity. The validity of (2.1) for lossless two-ports is proved in Williamson [14].

Divide (2.1) by ff_* to obtain

$$\frac{gg_*}{ff_*} = 1 + \frac{hh_*}{ff_*} \quad (2.2)$$

The reciprocal of (2.2) is

$$\frac{ff_*}{gg_*} = \frac{1}{1 + \frac{hh_*}{ff_*}} \quad (2.3)$$

which can be written in squared magnitude and attenuation form as

$$\left| \frac{f}{g} \right|^2 = \frac{1}{1 + \left| \frac{h}{f} \right|^2} \quad (2.4a)$$

$$\alpha = -10 \log_{10} \left| \frac{f}{g} \right|^2 = 10 \log_{10} \left(1 + \left| \frac{h}{f} \right|^2 \right) \quad (2.4b)$$

where α is the attenuation in dB, f/g is the transfer function and h/f is the characteristic function of the two-port network. Eq. (2.4) is normally used for the approximation of the transfer function. The polynomial g is obtained from the characteristic function h/f by using (2.1). Since g , h and f are real polynomials, all zeros of these polynomials are either real, or pairs of complex conjugates.

The expressions (2.4) show that the zeros of f and h are the attenuation poles and zeros, respectively. They are preferably located on the imaginary axis in order to create the maximum number of attenuation poles in the stopbands for the zeros of f and attenuation zeros in the passband for the zeros of h [18]. From (2.1), the degree of g is equal to the larger degree of the polynomials f and h , or possibly to the degree of both polynomials if they are the same. In case the degree of f is lower than that of h , there are in addition as many attenuation poles at infinity as there are units in the difference between these degrees.

2.2 Bandpass Filter

Mathematically, the frequency axis is divided into intervals called bands. One of them, the passband, is divided into sub-intervals with various upper-bound constraints on the value of the polynomial. Others, the stopbands, are divided into sub-intervals with lower-bound constraints. The passband and stopbands are separated by transition bands. A bandpass filter has a passband $[\omega_{-c}, \omega_{+c}]$, set between two stopbands $[0, \omega_{-s}]$ and $[\omega_{+s}, \infty)$, with $\omega_{-s} < \omega_{-c}$ and $\omega_{+c} < \omega_{+s}$. This is shown in Figure 2.1.

Traditionally, a bandpass filter is obtained through the frequency transformation from an optimal lowpass filter. This transformation leads to a distortion of the attenuation characteristic of the symmetrical lowpass filter. Figure 2.2 shows the nature of that distortion.

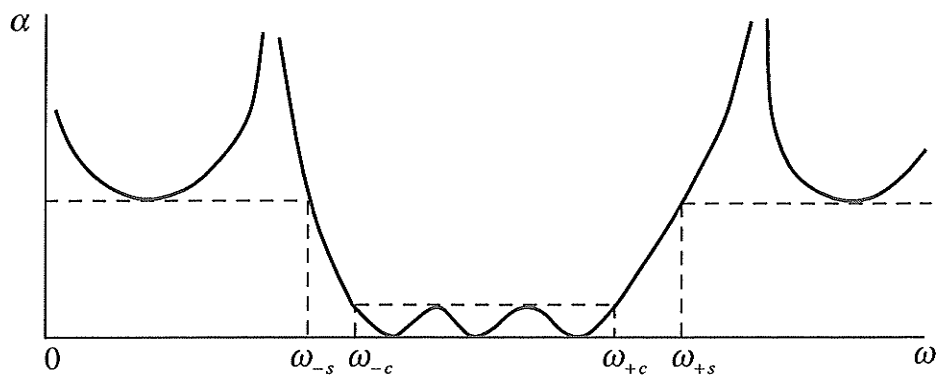


Figure 2.1: A simple bandpass filter

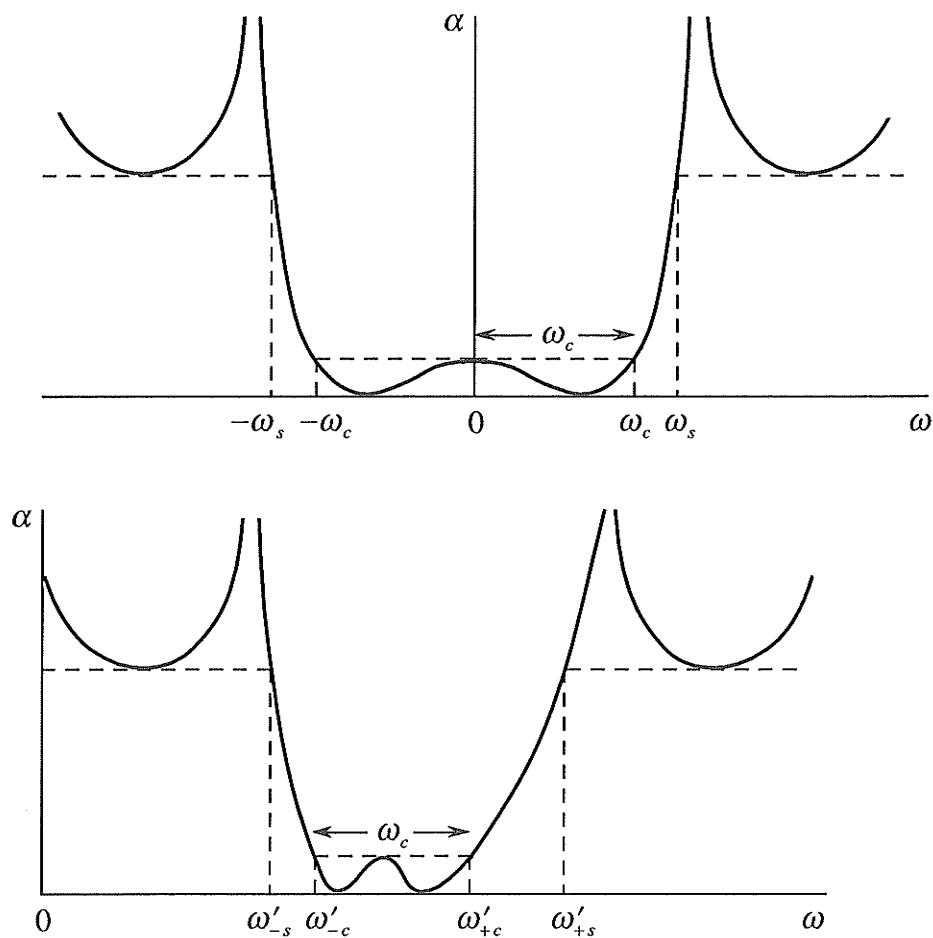


Figure 2.2: Frequency transformation of lowpass to bandpass filter

It can be seen that the upper transition band, from ω'_{+c} to ω'_{+s} , is wider than the lower transition band of the transformed bandpass filter, from ω'_{-s} to ω'_{-c} . Also, this transformation limits the bandpass filter to the symmetrical requirements of those of the lowpass filter. If the requirements are asymmetrical, they will be made symmetric (as shown in the dotted line of Figure 2.3) to those of the lower stopband after the transformation.

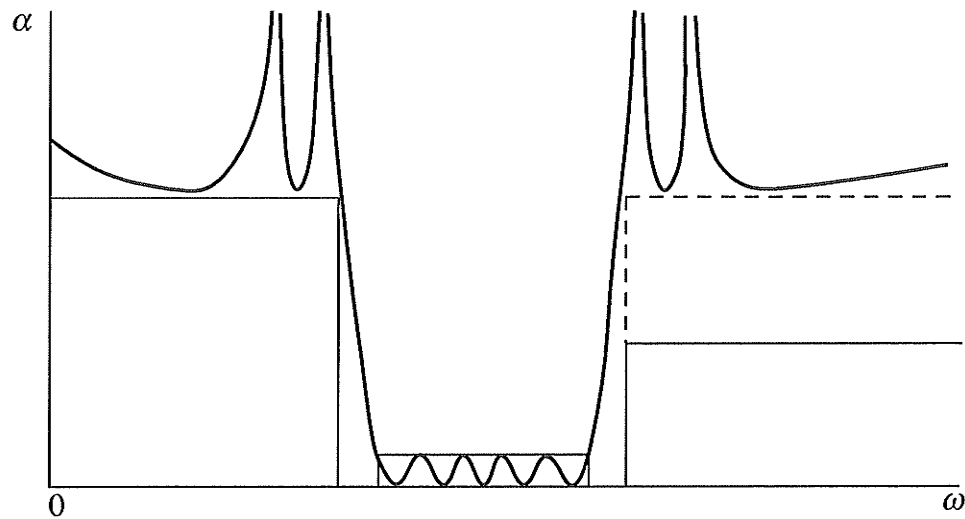


Figure 2.3: Lowpass to bandpass filter with asymmetric requirements

The frequency transformation can only be satisfactory when either the requirements are symmetrical, or the filter degree is not too high. In addition, the synthesis of a bandpass filter based on frequency transformation from a low pass filter is an over-simplified method whose cost is often higher due to extra elements in the circuit [18].

As a result, a unique approximation technique is required to generate bandpass filters directly. The attenuation requirements can be multi-level, asymmetric with respect to the upper and lower stopbands. The circuit will be less complex compared to those obtained by frequency transformation. This also makes no sacrifice of the degrees of freedom.

2.3 Remez Algorithm

In the investigation of the two approximation techniques, the Remez algorithm is used. It has been applied to the computation of polynomial rational approximations with considerable success. There are many possible variations of the algorithm. The original Remez algorithm can be found in [13] and other variants in [2], [6]. It is a method of exchange, but rather than exchanging one point at a time, it exchanges all points, in general, at one time. This algorithm consists in locating the $n + 1$ points (where n is the degree of the approximating function) and taking these points as the next set of points upon which the function is approximated.

A simple description of the Remez algorithm is presented to provide some insight of the approximations presented in this thesis. The Remez algorithm consists of following steps [18]:

- choose an initial set of $(n + 1)$ abscissas, x_j , in the interval $[a, b]$;
- construct a polynomial, $P_n(x)$, presenting errors, k_j which is defined as

$$k_j = P_n(x_j) - f(x_j)$$

These errors are identical in absolute value to a value k , at the abscissas;

- find the abscissas of the error local extrema in the interval;
- replace each abscissa x_j by the abscissa of the closest local extremum, whose error is of the same sign;
- iterate from the second step until k does not increase any more.

A simple example is shown in Figure 2.4 with the error curve corresponding to the polynomial $P_n(x)$ on the abscissas x_1, \dots, x_6 . After applying the Remez algorithm, x_2 , x_3 , and x_5 are replaced by x'_2 , x'_3 , and x'_5 , respectively.

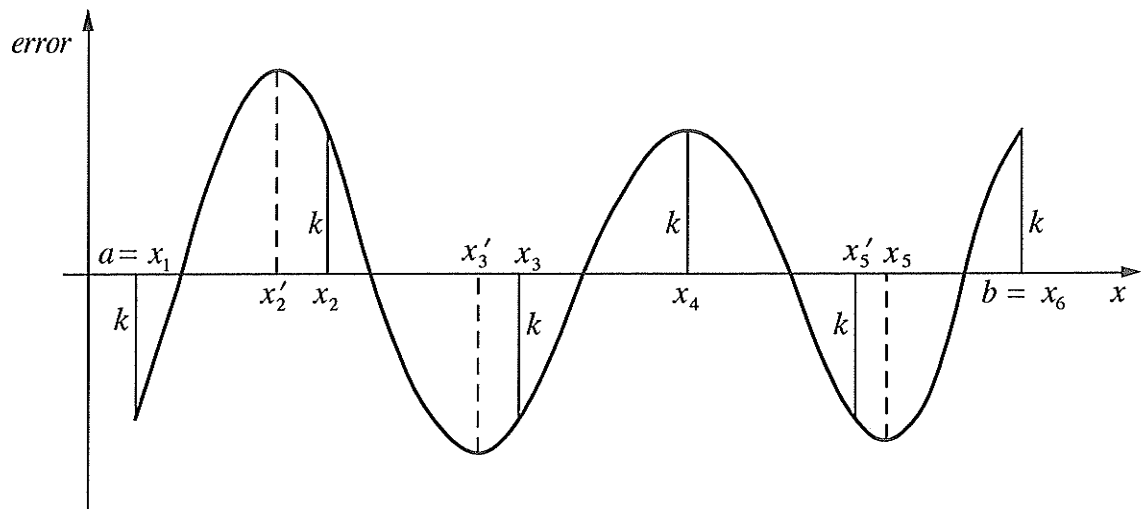


Figure 2.4: Remez algorithm

This algorithm is applied in modified form to approximate the transfer function by the algorithms described in the next chapters.

Chapter 3

Squared Magnitude Approximation

The primary intent in this chapter is to provide a brief description of the squared magnitude approximation for a rational bandpass filter function. The approximation is developed by Cohen [7]. This approximation is basically split into two parts: the first one uses the interpolation by a polynomial for the passband; in the second part, the use of pole shifting achieves the approximation to any specifications in the stopband. The attenuation specifications are general, multi-leveled, allowing for free attenuation poles.

3.1 Specification

A realizable approximation to the ideal magnitude-vs-frequency response of a filter transfer function is usually expressed as

$$|H(j\omega)|^2 = \frac{1}{1 + \varepsilon^2 f(\omega^2)} \quad (3.1)$$

The above equation is the same as in (2.4a) but with different notation. One important point is that the f here describes a function of the squared magnitude of the characteristic function is completely different from that in (2.4a). This f is a rational function of ω^2 and is expressed as a ratio of polynomials. Within this chapter, f refers to the function described above in (3.1). The above equation describes the squared magnitude of the transfer function $H(j\omega)$. ε is some constant. With a change of variable,

$$x = \omega^2 \quad (3.2)$$

$f(x)$ can be specified by the following conditions [5]:

$$f(x) \geq 0 \quad \text{for all } x \geq 0 \quad (3.3a)$$

$$f(x) \leq U_i \quad \text{for } x_{i-1} \leq x \leq x_i \quad i = 1, 2, \dots, m \quad (3.3b)$$

$$f(x) \geq Ll_i \quad \text{for } x_{-s-i} \leq x \leq x_{-s-i+1} \quad i = 1, 2, \dots, t \quad (3.3c)$$

$$f(x) \geq Lr_i \quad \text{for } x_{s+i-1} \leq x \leq x_{s+i} \quad i = 1, 2, \dots, v \quad (3.3d)$$

where $x_{-s-t} = 0$, $x_- = x_0$, $x_+ = x_m$, $x_{s+v} = \infty$;
 x_+ and x_- are the passband edges;
 x_{-s} and x_s are the left and right stopband edges respectively;
 $0 < U_i \leq 1$, with $U_i = 1$ for some i , are upper bounds of $f(x)$ specified in the passband;
 $Ll_i > 1$ are lower bounds specified in the left stopband;
 $Lr_i > 1$ are lower bounds specified in the right stopband.

This specification is illustrated in Figure 3.1, where the interval $[x_-, x_+]$ is the passband, $[0, x_{-s}]$ and $[x_s, \infty)$ are the left and right stopbands, respectively, while (x_{-s}, x_-) and (x_+, x_s) are the transition bands.

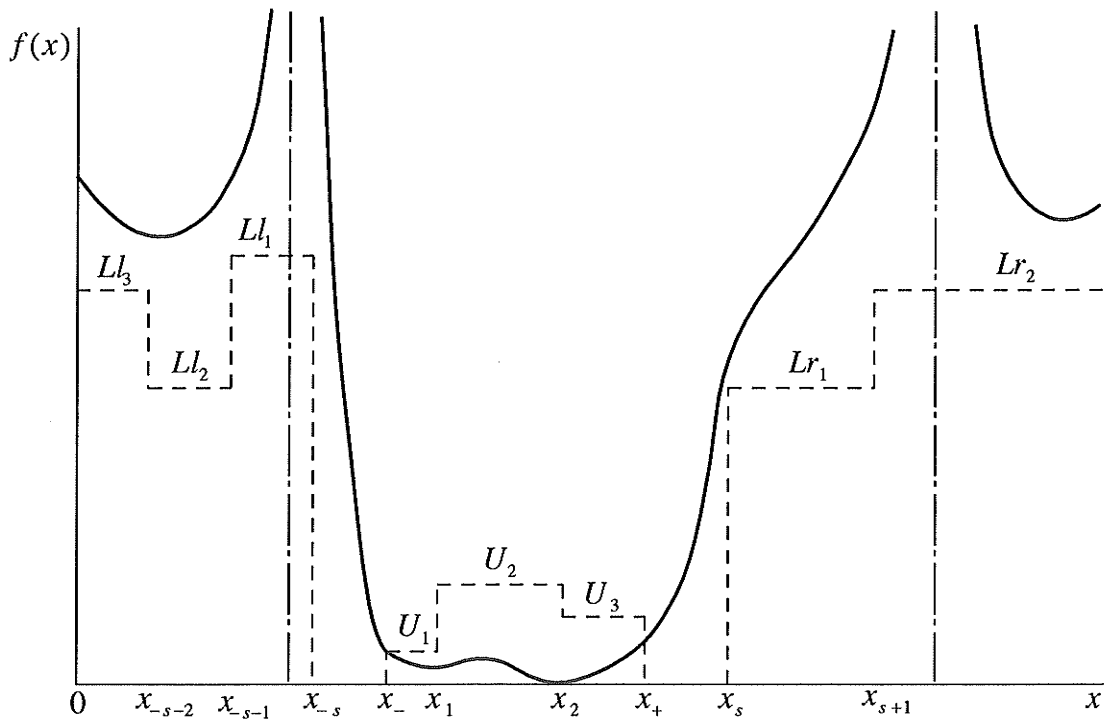


Figure 3.1: $f(x)$ for a bandpass filter

As can be seen in the above figure, there can be any finite number of steplike bounds in the passband and stopbands. This kind of characteristic behaviour can be specifically identified

by introducing two distinct discontinuous functions.

The “ceiling function” $C(x)$, used for the passband, is defined as follows:

$$C(x) = U_i \quad \text{for } x_{i-1} < x < x_i \quad i = 1, 2, \dots, m \quad (3.4a)$$

$$x_- = x_0 < x_1 < \dots < x_{m-1} < x_m = x_+$$

$$C(x_-) = C(x_0) = U_1 \quad (3.4b)$$

$$C(x_+) = C(x_m) = U_m \quad (3.4c)$$

$$C(x_i) = \min(U_i, U_{i+1}) \quad i = 1, 2, \dots, m-1 \quad (3.4d)$$

The “floor function” $F(x)$, used for the stopbands, is defined as follows:

for the left stopband:

$$F(x) = Ll_i \quad \text{for } x_{-s-i} < x < x_{-s-i+1} \quad i = 1, 2, \dots, t \quad (3.5a)$$

$$0 = x_{-s-t} < x_{-s-t+1} < \dots < x_{-s+1} < x_{-s} < x_-$$

$$F(x_{-s}) = Ll_1 \quad (3.5b)$$

$$F(0) = F(x_{-s-t}) = Ll_t \quad (3.5c)$$

$$F(x_{-s-i}) = \max(Ll_i, Ll_{i+1}) \quad i = 1, 2, \dots, t-1 \quad (3.5d)$$

for the right stopband:

$$F(x) = Lr_i \quad \text{for } x_{s+i-1} < x < x_{s+i} \quad i = 1, 2, \dots, v \quad (3.5e)$$

$$x_+ < x_s < x_{s+1} < \dots < x_{s+v} = \infty$$

$$F(x_s) = Lr_1 \quad (3.5f)$$

$$F(x_{s+i}) = \max(Lr_i, Lr_{i+1}) \quad i = 1, 2, \dots, v-1 \quad (3.5g)$$

As described in Equation (3.3), the basic electric filter design criteria thus become

$$0 \leq f(x) \leq C(x) \quad \text{for } x_- \leq x \leq x_+ \quad (3.6a)$$

$$f(x) \geq F(x) \quad \text{for } 0 \leq x \leq x_{-s} \text{ and } x \geq x_s \quad (3.6b)$$

As mentioned earlier, $f(x)$ is a rational function, which can be expressed as a ratio of polynomials:

$$f(x) = \frac{N_n(x)}{D_q(x)} \quad (3.7)$$

where $N_n(x)$ is a polynomial of degree n and $D_q(x)$ is a monic polynomial of degree q , and where q is restricted to be less than n . And the order of $f(x)$ is equal to n . Thus the filter transfer function has the order n . According to the conditions in (3.3a), any real zero of N in $[0, \infty)$ must be of even multiplicity; also it cannot be located in the stopband because of (3.6b). Because of (3.3a), any real zero of D in $[0, \infty)$ must be of even multiplicity if it lies in $(0, \infty)$ and it cannot be located in the passband because of (3.6a).

A rational function $f(x)$ with minimal order, n , is sought such that it will satisfy the constraints (3.6). Therefore, the degree of the numerator in $f(x)$ should be as small as possible and $f(x)$ should exhibit as large a value as possible in the transition bands (x_{-s}, x_-) and (x_+, x_s) . In other words, it maximizes $F(x_{-s})$ or $F(x_s)$, or both. This allows the realizability of a bandpass filter with minimal complexity.

The optimal rational function $f^*(x)$ has the pole-zero form

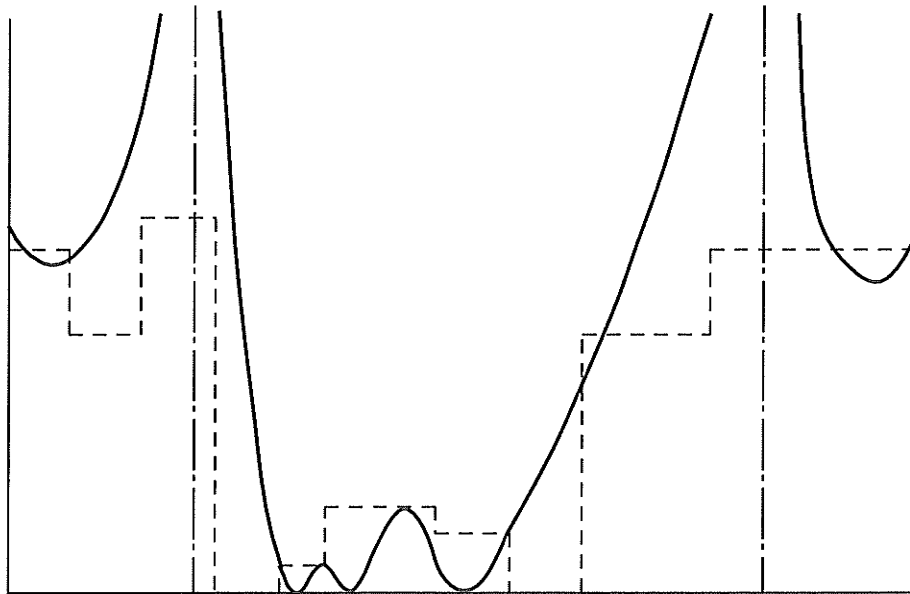
$$f^*(x) = a_n \frac{\prod_{i=1}^{n/2} (x - z_i)^2}{\prod_{i=1}^{q/2} (x - p_i)^2} \quad (3.8)$$

where all z_i and p_i are real, $x_- < z_i < x_+$, and p_i satisfies either $0 < p_i \leq x_{-s}$ or $p_i \geq x_s$. a_n is some real positive value which is actually the same as ϵ^2 in (3.1) and where n and q are even. Within the passband, from x_- to the left of the first minima and from the right of the last minima to x_+ , as well as in every interval between two adjacent minimas, this optimal function will reach its maximum allowable value $C(x)$ at at least one point. Thus, it has $n/2$ distinct zero minima and $n/2 - 1$ maxima in the passband. If the positions of the poles is anticipated, this optimal function will be completely characterized by its behaviour in the passband.

3.2 The Cohen Algorithm

The importance of this algorithm is to attempt to correct the maximum discrepancies of a rational function which does not satisfy the specifications indicated in Equation (3.3). The algorithm starts to interpolate the numerator of the rational function by keeping the denominator fixed. That is, with initial poles fixed, an interpolating process is performed which passes the curve through the extrema and the proper end values in the passband. After the first interpolation, the extrema of this newly obtained rational function will, in general, not satisfy the conditions (3.3a,b). Using these new extrema and end values, a second interpolation is carried out. This process is repeated until the interpolation points stabilize. This is called the *interpolation stage*.

The rational function resulting from the interpolation stage will, in general, not satisfy the conditions (3.3c,d). This situation is illustrated in Figure 3.2.



*Figure 3.2: Rational function resulting from Interpolation stage.
To be raised in stopbands in pole-shifting stage.*

The rational function is lower than the floor function in the stopbands. The next step is to raise the rational function over the stopband region. This is to be done in a *pole-shifting stage*. Using first-order approximations, a set of pole shifts is determined with the goal of raising the offending minima in the stopbands. This stage is somewhat similar to the global

exchange method proposed by Remez [6]. This is to achieve the equiripple approximation to any specification in the stopbands. This pole-shifting routine involves the solution of a set of linear equations.

Another case which may have occurred after the interpolation stage is illustrated in Figure 3.3.

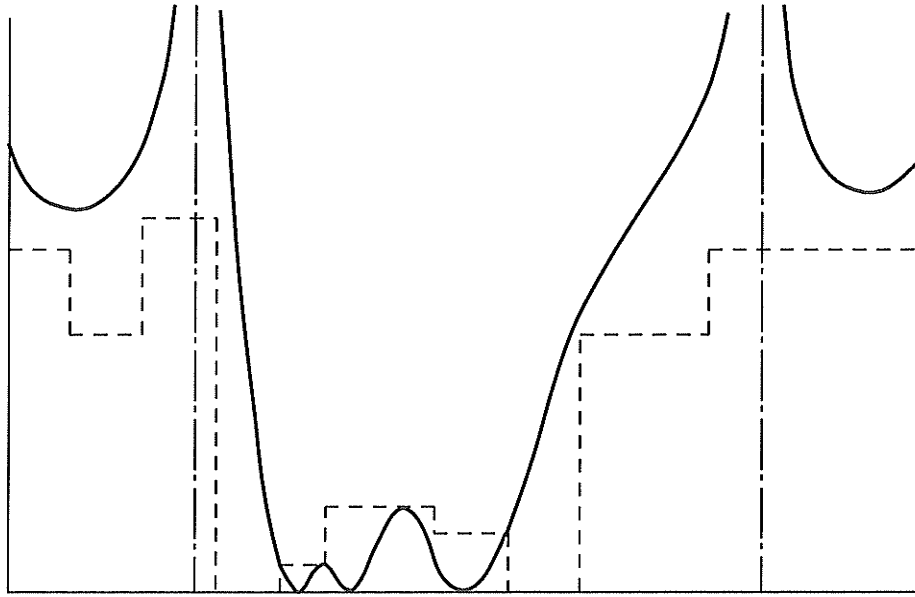


Figure 3.3: *Rational function resulting from Interpolation stage.
To be lowered in stopbands in pole-shifting stage.*

The rational function is larger than the specifications in the stopbands. In the pole-shifting stage, the rational function will be lowered until it equals the specified floor function to the left of the rightmost pole in the left stopband and to the right of the leftmost pole in the right stopband. This will result in raising the rational function in the transition band to yield an attenuation as large as possible. The above two aspects of the pole-shifting stage may be combined in a single process.

After the pole-shifting stage, if the resulting rational function still does not satisfy conditions (3.3), because the zeros no longer satisfy (3.3a,b), the whole algorithm is repeated starting with the interpolation stage. But this time the revised set of pole locations is used. This alternation of stages is repeated until the desired attenuation behaviour results, i.e., conditions (3.3) are satisfied.

3.3 The Interpolation Stage

Initially, if the positions of the poles are not fixed, the $q/2$ poles are randomly placed in any convenient location in the stopband. These pole positions are $p_i^{(0)}$, ($i = 1, 2, \dots, q/2$). Then the $n + 1$ interpolation abscissas are randomly selected from the passband, namely, the $x_i^{(0)}$, ($i = 0, 1, \dots, n$). The approximating rational function is in the form

$$f^{(0)}(x) = \frac{a_n \cdot \prod_{i=0}^n (x - z_i^{(0)})}{\prod_{i=1}^{q/2} (x - p_i^{(0)})^2} \quad (3.9)$$

with zeros $z_i^{(0)}$. A polynomial interpolation algorithm is then used to interpolate the points

$$f^{(0)}(x_i^{(0)}) = C(x_i^{(0)}) \quad \text{for } i \text{ even} \quad (3.10a)$$

$$f^{(0)}(x_i^{(0)}) = 0 \quad \text{for } i \text{ odd} \quad (3.10b)$$

thereby determining the $z_i^{(0)}$ and the constant multiplier, a_n . The interpolation algorithm and how it is applied is described in next section (Sec. 3.4). After getting, all the z_i 's, the locations of the minima of $f^{(0)}(x)$ are obtained next, namely, $y_1^{(0)}, y_3^{(0)}, \dots, y_{n-1}^{(0)}$. They all should be found in the interval (x_-, x_+) . Within each of the intervals (between two adjacent minima):

$$[x_-, y_1^{(0)}), \dots, (y_i^{(0)}, y_{i+2}^{(0)}), \dots, (y_{n-1}^{(0)}, x_+] \quad i = 1, 3, \dots, n-3$$

a point $y_i^{(0)}$, i even, is located such that $f(y_i^{(0)}) - C(y_i^{(0)})$ is maximized. A new set of interpolation abscissas is then determined as

$$x_i^{(1)} = y_i^{(0)} \quad i = 0, 1, \dots, n \quad (3.11)$$

The new abscissas $x_i^{(1)}$ are compared to abscissas $x_i^{(0)}$. If they are not equal, a second iteration is performed. This will produce $f^{(1)}(x)$ and the new abscissas $x_i^{(2)}$. The comparison is carried out again and so on. On the other hand, if they are equal, or $|x_i^{(k+1)} - x_i^{(k)}|$ is less than some chosen tolerance for all i , the interpolation procedure is stopped.

For clarity, the resulting rational function is then called $g^{(1)}(x)$. This process is referred to as the interpolation stage. The interpolating rational function $g^{(1)}(x)$ satisfies conditions (3.3a,b) but may not necessarily agree with (3.3c,d). This is mentioned earlier and illustrated in Figures 3.2 and 3.3. In the other case, if conditions (3.3c,d) are satisfied, it is a solution, but may not be the optimal one. The process is passed to the pole-shifting stage.

For the starting solution of the $n + 1$ interpolation abscissas, it is convenient to take equally spaced points on $[x_-, x_+]$:

$$x_i^{(0)} = x_- + \frac{i \cdot (x_+ - x_-)}{n} \quad i = 0, 1, \dots, n \quad (3.12)$$

It is also to be noted that the $y_i^{(k)}$, i even, are relatively easy to obtain. Each y_i must be located at either one of the break points of the bounds in the passband, i.e. $x_-, x_1, x_2, \dots, x_{m-1}, x_+$, or else at a maximum of $f^{(k)}(x)$ as illustrated in Figure 3.4.

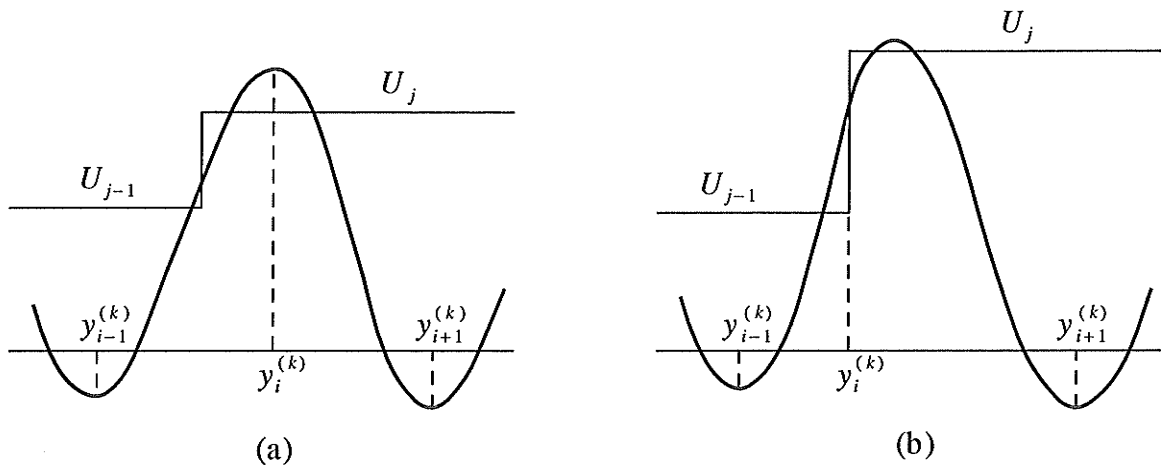


Figure 3.4: Discrepancies (overshoots) resulting at the k^{th} interpolation :
 (a) at minima and maximum of interpolation rational function $f^{(k)}(x)$; (b) at minima and break point;

3.4 Polynomial Interpolation

From the last section, in order to obtain z_i 's in (3.9), an ordinary polynomial interpolation is required. Since the poles, p_i 's are initially chosen, (3.9) can be written as

$$\prod_{i=0}^n (x - z_i^{(0)}) = f^{(0)}(x) \cdot \prod_{i=1}^{q/2} (x - p_i^{(0)})^2 \quad (3.13)$$

Applying the conditions in (3.10), the right hand side of (3.13) can be replaced by a constant, β_i , which is associated with the proper abscissa x_i . For clarification, the abscissa x_i is replaced by α_i . Therefore, a set of interpolation points (α_i, β_i) is formed. By a theorem in [19], if α_i 's are distinct, a unique polynomial $U(x)$ can be interpolated having degree n such that

$$U(\alpha_i) = \beta_i \quad i = 0, 1, \dots, n \quad (3.14)$$

where $U(x)$ corresponds to the polynomial of the left hand side of (3.13). The α_i are distinct since the abscissa x_i are all separated on the passband. Therefore the interpolation of $U(x)$ is accomplished by using Newton's Interpolation Algorithm [19].

The pseudo-code of Newton's Interpolation Algorithm is shown in Figure 3.5.

```

1.  begin
2.     $U(x) := \beta_0;$ 
3.     $M(x) := 1;$ 
4.    for  $k := 1$  until  $n$  do
5.      begin
6.         $M(x) := M(x) \times (x - \alpha_{k-1});$ 
7.         $c := [M(\alpha_k)]^{-1};$ 
8.         $\delta := [\beta_k - U(\alpha_k)] \times c;$ 
9.         $U(x) := U(x) + \delta \times M(x)$ 
10.     end
11. end

```

Figure 3.5: Newton's Interpolation Algorithm

where k is an integer iteration index; $U(x)$ is the expression for the polynomial in (3.14); α_k and β_k are the abscissas and ordinates of the interpolation points, respectively; $M(x)$ is the monic polynomial which constitutes all factors $(x - z_i)$ where $i \leq k$; and δ is the coefficient of highest power in $U(x)$.

It is noted that the interpolating polynomial in (3.13) is in product form. This is because the coefficient form polynomials are very sensitive to errors incurred during numerical processing [14]. Thus, $U(x)$ in Line 9 of the above algorithm is determined in product form. The zeros of $U(x)$ can be determined by a special zero-finding routine. On the other hand, $M(x)$ is already in product form which is shown in Line 6. The detail of the zero-finding routine will not be discussed in this thesis. They can be referred to in [20].

An important point has to be noted when this algorithm is applied. According to Williamson [14], the set of interpolation points (α_i, β_i) has to be arranged in ascending order based on the ordinate magnitudes. Otherwise critical cases may occur if the α_i values are clustered close together and the β_i values vary by many orders of magnitude. If the large-ordinate points are interpolated first, the entire interpolation process becomes insensitive to the comparatively very-small-ordinate points occurring later. As a result, some accuracy may be lost during this process and the interpolated polynomial may not accurately reflect the interpolation points from which it was derived. On the other hand, if the small-ordinate points are interpolated first, an accurate interpolation process is preserved.

3.5 The Pole-Shifting Stage

The approximation of $f(x)$ is continued with the rational function $g^{(1)}(x)$ in the pole-shifting stage. Let $p_v^{(0)}$ to be the rightmost pole position in the left stopband. An abscissa y_i , ($i = 1, 2, \dots, q/2$), is found within each of the intervals in the left stopband:

$$[0, p_1^{(0)}), (p_1^{(0)}, p_2^{(0)}), \dots, (p_{v-1}^{(0)}, p_v^{(0)})$$

and within each of the intervals in the right stopband:

$$(p_{v+1}^{(0)}, p_{v+2}^{(0)}), \dots, (p_{\frac{q}{2}-1}^{(0)}, p_{\frac{q}{2}}^{(0)}), (p_{\frac{q}{2}}^{(0)}, \infty)$$

Abscissa y_i is located such that $g^{(1)}(y_i) - F(y_i)$ is minimized. For obtaining the pole displacements $\Delta p_i^{(0)}$, ($i = 1, 2, \dots, q/2$), consider the distance functions

$$h(y_i, \mathbf{p}) = F(y_i) - g^{(1)}(y_i, \mathbf{p}^{(0)}) \quad (3.15)$$

where $\mathbf{p} = (p_1^{(0)}, p_2^{(0)}, \dots, p_{\frac{q}{2}}^{(0)})$. The $\mathbf{p}^{(0)}$ above is the one which was initially chosen arbitrarily. If this $\mathbf{p}^{(0)}$ happens to give the optimal rational functions, it will follow that $h(y_i, \mathbf{p}^{(0)}) = 0$. If this is not the case consider the possibility of an improvement by changing $\mathbf{p}^{(0)}$ by an amount $\Delta \mathbf{p}$ to a new \mathbf{p} . The Taylor series expansion of $h(y_i, \mathbf{p})$ around $\mathbf{p}^{(0)}$ gives

$$h(y_i, \mathbf{p}) \cong h(y_i, \mathbf{p}^{(0)}) + \left(\sum_{j=1}^{\frac{q}{2}} \Delta p_j \cdot \frac{\partial}{\partial p_j} \right) \cdot h(y_i, \mathbf{p}) \Big|_{\mathbf{p}=\mathbf{p}^{(0)}} \quad (3.16)$$

$i = 1, 2, \dots, q/2$

It is noted that only the first two terms of the expansion are retained. Since $h(y_i, \mathbf{p}) = 0$ is desired, impose this condition, i.e. require that

$$0 = h(y_i, \mathbf{p}^{(0)}) + \left(\sum_{j=1}^{\frac{q}{2}} \Delta p_j \cdot \frac{\partial}{\partial p_j} \right) \cdot h(y_i, \mathbf{p}) \quad i = 1, 2, \dots, q/2$$

Replace h by $F(y_i) - g^{(1)}(y_i)$ in the last equation to obtain

$$0 = F(y_i) - g^{(1)}(y_i, \mathbf{p}^{(0)}) - \left(\sum_{j=1}^{\frac{q}{2}} \Delta p_j \cdot \frac{\partial}{\partial p_j} \right) \cdot g^{(1)}(y_i, \mathbf{p}) \Big|_{\mathbf{p}=\mathbf{p}^{(0)}} \quad (3.17)$$

$i = 1, 2, \dots, q/2$

After the equation is rearranged and the designation of the parameter \mathbf{p} is dropped from $g^{(1)}$, it is written as

$$\sum_{j=1}^{\frac{q}{2}} \frac{\partial g^{(1)}(y_i)}{\partial p_j^{(0)}} \cdot \Delta p_j^{(0)} = F(y_i) - g^{(1)}(y_i) \quad i = 1, 2, \dots, q/2 \quad (3.18)$$

For the partial derivative of $g^{(1)}(y_i)$ with respect to p_j , assume $g^{(1)}$ has the form as in (3.8). Taking the natural logarithm of both sides, it becomes

$$\ln g^{(1)}(y) = \ln a_n + \sum_{i=1}^{n/2} 2 \ln |y - z_i| - \sum_{i=1}^{q/2} 2 \ln |y - p_i| \quad (3.19)$$

Taking the partial derivative with respect to p_j , it is then written as

$$\frac{1}{g^{(1)}(y)} \cdot \frac{\partial g^{(1)}(y)}{\partial p_j} = -2 \cdot \frac{(-1)}{y - p_j} \quad (3.20)$$

Therefore,

$$\frac{\partial g^{(1)}(y)}{\partial p_j^{(0)}} = \frac{2g^{(1)}(y)}{(y - p_j^{(0)})} \quad (3.21)$$

Combining (3.18) and (3.21), the pole displacements, $\Delta p_i^{(0)}$, are obtained by solving the system of $q/2$ linear equations:

$$\sum_{j=1}^{q/2} \frac{2g^{(1)}(y_i)}{y_i - p_j^{(0)}} \cdot \Delta p_j^{(0)} = F(y_i) - g^{(1)}(y_i) \quad i = 1, 2, \dots, q/2 \quad (3.22)$$

Thus the new pole positions are calculated as

$$p_i^{(1)} = p_i^{(0)} + \Delta p_i^{(0)} \quad i = 1, 2, \dots, q/2 \quad (3.23)$$

This ends the pole-shifting stage. This stage is different from the interpolation stage in that this process is not iterative. If by some chance $\Delta p^{(0)} = 0$, i.e. $p^{(1)} = p^{(0)}$, then $g^{(1)} = f^*$ is the optimal rational function sought. Otherwise, the whole algorithm is restarted. Only this time, the following rational function is used instead of (3.9).

$$f^{(0)}(x) = \frac{\text{numerator of } g^{(1)}(x)}{\prod_{i=1}^{q/2} (x - p_i^{(1)})^2} \quad (3.24)$$

From the $f^{(0)}(x)$ interpolation abscissas are obtained for the determination of the interpolation function $f^{(1)}(x)$ as described above. This interpolation process eventually leads to $g^{(2)}(x)$, which is then used in the pole-shifting stage, etc.

The rational function which results from this algorithm may not satisfy conditions (3.3c,d). Especially in the intervals $[p_v, x_{-s}]$ and $[x_s, p_{v+1}]$, the rational function may happen to be below the specification levels in the stopband edges. In that case, the poles may be redistributed differently between stopbands and the whole process started anew. If this still fails to satisfy (3.3c,d), there may be no rational function of numerator degree n which can satisfy (3.3c,d). The whole process is then restarted using rational functions of numerator degree $n + 2$.

If, however, the optimal rational function resulting from the algorithm satisfies all of (3.3), then it is “the” optimal function sought, and it can be shown to be unique [7].

3.6 Epilogue

Due to the linearization of the Taylor expansion and the shift of the essential frequencies, several iterations are necessary. It is expected that convergence can be guaranteed if the displacements adopted for each shift stage are kept small in magnitude, even if the solution of the set of linear equations results in large values. This suggests that after solving for the Δp_i 's, they are normalized by a certain constant in order to make them small. But the directions of the shifts are still the same. The method of the search for this normalizing constant will be explained in Chapter 5.

As mentioned earlier in (3.7), q is limited to be less than n . In the pole-shifting stage, when locating $y_{\frac{q}{2}}$ within the interval $(p_{\frac{q}{2}}, \infty)$, two situations may occur. When $n \leq q$, $f(x)$ will approach 0 as $x \rightarrow \infty$. In this case, $y_{\frac{q}{2}}$ will be located at ∞ ! And also, the rational function may not satisfy condition (3.6b). Thus, q is limited to satisfy $q < n$.

The whole process of this squared magnitude approximation is summarized as a flow chart shown as in Figure 3.6.

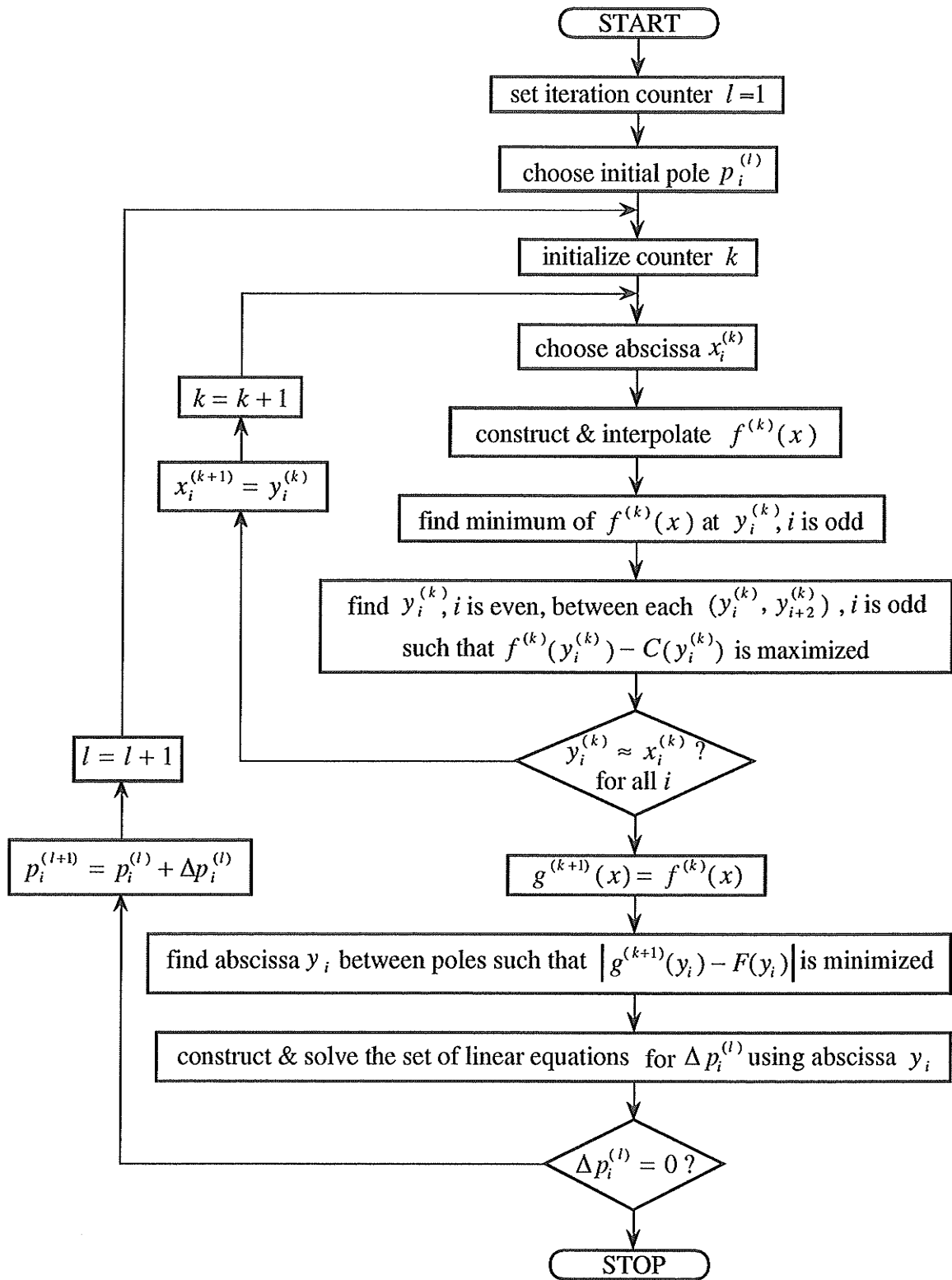


Figure 3.6: The Cohen Algorithm

Chapter 4

Attenuation Approximation

The previous chapter briefly described the squared magnitude approximation using an iterative interpolation process and a pole-shifting routine for the characteristic function of a bandpass filter. In this chapter, the concept of attenuation approximation is presented. This approximation technique was introduced by Devleeschouwer and Grenez [12]. It is described for the approximation in the Chebyshev sense of an arbitrary attenuation curve in the passband and in the stopbands by a real transfer function. The approximation process is reduced to a common optimization problem. It is then solved by the efficient Remez algorithm which is the original approach, and is extended to more general passband specifications. This method was first described in [22] for analog filters. In addition some of the transmission zeros or attenuation zeros can be fixed at prescribed frequencies. The applied optimization procedures are mainly based on linearization methods.

The approximation algorithm described here is originally designed for both analog and digital (recursive) filters. Only the analog bandpass filter design is described in this thesis. Transformed variables are used to obtain a unique formulation. In order to improve the computational accuracy, a second transformation is then used. This transformation gains its advantages for the optimization of the bandpass filter. For other filter designs, different transformations are employed. They can be found in [12].

4.1 Specifications

A transfer function $H(s)$ of a bandpass filter is to be determined which has a degree as low as possible. It must satisfy the following specifications:

$$\alpha(f) \geq \alpha_s(f) \quad \text{in the stopbands} \quad (4.1a)$$

$$0 \leq \alpha(f) \leq \alpha_s(f) \quad \text{in the passband} \quad (4.1b)$$

where $\alpha(f)$ is the attenuation function and is defined as

$$\alpha(f) = -10 \log |H(j2\pi f)|^2 \quad (4.2)$$

and $\alpha_s(f)$ is the given specification function. Since the amplitude of $H(s)$ should approximate unity in the passband, it is convenient to use the characteristic function $K(s)$, which is related to the transfer function $H(s)$ according to

$$H(s) \cdot H(-s) = \frac{1}{1 + K(s) \cdot K(-s)} \quad (4.3)$$

The approximation parameters are the zeros, poles and the constant factor of $K(s)$. The advantage of these parameters is due to the fact that the zeros and poles of $\alpha(f)$ are identical to those of $K(s)$ on the imaginary axis $s = j\omega$. This follows immediately from

$$\alpha(f) = -10 \log |H(j2\pi f)|^2 = 10 \log \left[1 + |K(j2\pi f)|^2 \right] \quad (4.4)$$

Because of this, it is possible without difficulty to choose initial values for the approximation parameters, namely, all the attenuation zeros (AZ) in the passbands and all the transmission zeros (TZ), or just poles, in the stopbands. The form

$$K(s) = C_k \cdot s^{-m_0} \cdot \prod_{i=1}^n (s^2 + \omega_i^2)^{-m_i} \quad \omega_i \neq 0, \quad i = 1, 2, \dots, n \quad (4.5)$$

is chosen for the characteristic function where positive integers of m_i ($i = 0, 1, \dots, n$) correspond to TZs and negative integers correspond to AZs. For the variable parameters C_k and ω_i , initial values must be assumed. These will be iteratively improved as described later. m_i is called the multiplicity of the singularity. m_i is normally equal to ± 1 . For multiple poles/zeros, $m_i \leq -2$ or $m_i \geq 2$. The multiplicity m_∞ of the singularity at infinity does not appear in (4.5) but is given by the difference between the numerator and the denominator degrees of the rational function $K(s)$. It is derived directly from (4.5) as

$$m_\infty = -m_0 - 2 \sum_{i=1}^n m_i \quad (4.6)$$

$k(s)$ is defined by

$$k(s) = K(s) \cdot K(-s) \quad (4.7)$$

Obviously, it is an even function of s . Therefore, the variable y is introduced according to

$$y = -s^2 \quad (4.8)$$

The specification originally given on the axis $s = j\omega$ is transferred to a specification on the real axis of the y -plane. Hence $k(s)$ can be written as

$$k(y) = C_y \cdot y^{-m_0} \cdot \prod_{i=1}^n (y - y_i)^{-2m_i} \quad (4.9)$$

where $C_y = C_k^2$ and $y_i = \omega_i^2$, $i = 1, 2, \dots, n$

For real frequencies, y is real, positive and increasing for increasing frequency.

4.2 The Transformed Variable

In order to improve the accuracy near the passband limits where the zeros tend to cluster, as well as to simplify the solution of the approximation problem [6], a new transformed variable will be used in the optimization procedure. The transformed variable x is related to the variable y by

$$x = \frac{y - t_2}{y - t_1} \quad (4.10)$$

where t_1 and t_2 are the y values corresponding to the passband limits with $t_1 =$ passband lower edge, and $t_2 =$ passband upper edge. The passband is mapped onto the negative real x -axis and the stopbands onto the positive real x -axis. The x - y mapping is illustrated in Figure 4.1.

References [24–26] give a detailed justification of this particular choice of transformation. To take the full advantage of the numerical properties of the transformation, the entire synthesis process must be performed with the x variable.

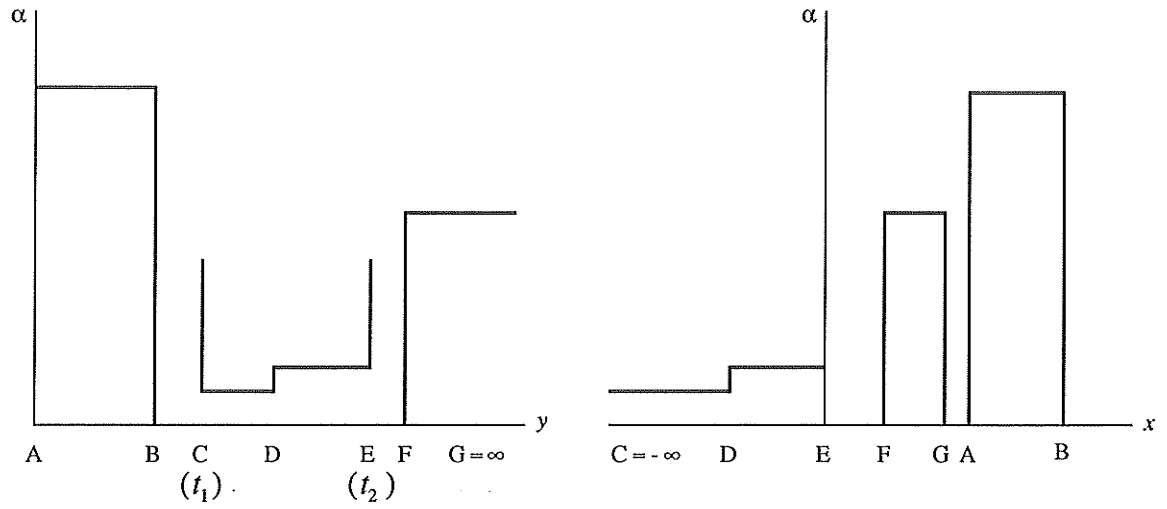


Figure 4.1: x - y mapping for the bandpass filter

It is noted that when y approaches ∞ , x is mapped to 1. And when y approaches 0, x is mapped to t_2/t_1 . Therefore, $G = 1$ and $A = t_2/t_1$ on the x -axis in Figure 4.1. It is assumed that no zero (TZ or AZ) is located on a passband edge. After some algebraic manipulations with the substitution of (4.10), (4.9) becomes

$$k(x) = C_x \cdot \prod_{i=1}^{n+2} (x - x_i)^{-m'_i} \quad (4.11)$$

with

$$x_i \neq 0 \quad i = 1, 2, \dots, n+2$$

$$x_i = \frac{y_i - t_2}{y_i - t_1} \quad m'_i = 2m_i \quad i = 1, 2, \dots, n$$

$$x_{n+1} = 1 \quad m'_{n+1} = m_\infty$$

$$x_{n+2} = \frac{t_2}{t_1} \quad m'_{n+2} = m_0$$

C_x is related to C_y as follows. From (4.10), y is written in terms of x as

$$y = \frac{t_1 x - t_2}{x - 1} \quad (4.12)$$

Upon substituting this into (4.9), it becomes

$$\begin{aligned}
 k(x) &= C_y \cdot \left(\frac{t_1 x - t_2}{x-1} \right)^{-m_0} \cdot \prod_{i=1}^n \left(\frac{t_1 x - t_2}{x-1} - \frac{t_1 x_i - t_2}{x_i - 1} \right)^{-2m_i} \\
 &= C_y \cdot \frac{(t_1 x - t_2)^{-m_0}}{(x-1)^{-m_0}} \cdot \prod_{i=1}^n \left(\frac{(t_2 - t_1)}{(x-1)(x_i - 1)} \cdot (x - x_i) \right)^{-2m_i} \\
 &= C_y \cdot t_1^{-m_0} \cdot \left(x - \frac{t_2}{t_1} \right)^{-m_0} \cdot (x-1)^{m_0} \cdot \prod_{i=1}^n (x-1)^{2m_i} \cdot \prod_{i=1}^n \left(\frac{t_2 - t_1}{x_i - 1} \right)^{-2m_i} \cdot \prod_{i=1}^n (x - x_i)^{-2m_i}
 \end{aligned}$$

According to (4.6),

$$k(x) = C_y \cdot t_1^{-m_0} \cdot \left(x - \frac{t_2}{t_1} \right)^{-m_0} \cdot (x-1)^{m_0} \cdot \prod_{i=1}^n \left(\frac{t_2 - t_1}{x_i - 1} \right)^{-2m_i} \cdot \prod_{i=1}^n (x - x_i)^{-2m_i} \quad (4.13)$$

Thus, comparing to (4.11)

$$C_x = C_y \cdot t_1^{-m_0} \cdot \prod_{i=1}^n \left(\frac{t_2 - t_1}{x_i - 1} \right)^{-2m_i} \quad (4.14)$$

Notice that the function $k(x)$ has no singularity at infinity.

For the optimization process, a logarithmic transformation converts $k(x)$ into a sum of terms, each of these terms corresponds to a zero (or pole) only. A function $\beta(x)$ is used which is defined by

$$\beta(x) = \ln|k(x)| = \ln k(x) \quad (4.15)$$

Since $k(x)$ is a magnitude squared function for any real frequency, it must be non-negative. Thus, $\beta(x)$ is a real function of x .

$$\beta(x) = C_\beta - \sum_{i=1}^{n+2} m'_i \cdot \ln|x - x_i| \quad (4.16)$$

where $C_\beta = \ln|C_x|$

Let $\beta_s(x)$ be the transformed function of $\ln \alpha_s(x)$. Thus, the filter specifications in (4.1) become

$$\beta(x) \geq \beta_s(x) \quad \text{in the left and right stopbands, and} \quad (4.17a)$$

$$\beta(x) \leq \beta_s(x) \quad \text{in the passband.} \quad (4.17b)$$

4.3 The Devleeschouwer and Grenez Algorithm

The passband and stopband weighted margins are defined by

$$M_{BP} = \text{minimum over the passband of } (\beta_s(x) - \beta(x))/W_{BP} \quad (4.18a)$$

$$M_{BS} = \text{minimum over the stopbands of } (\beta(x) - \beta_s(x))/W_{BS} \quad (4.18b)$$

where W_{BP} and W_{BS} are two positive constants. Some of the zeros x_i are defined to be movable and others are fixed (such as the one at $s = 0, \infty$ or at some frequencies prescribed in the specifications). The optimal bandpass filter will be obtained in the Chebychev sense if movable zeros and the constant C_β are found such that $D = \min(M_{BP}, M_{BS})$ is maximum.

An arc will be defined as the portion of the $\beta(x)$ curve between two adjacent movable zeros or between a movable zero and the adjacent band limit. This is the general definition of an arc, given by Fujisawa [23]. A fixed zero, whose frequency is prescribed in the specifications, does not limit an arc. (Evidently, zeros at $s = 0$ or ∞ are always considered as fixed zeros.) If there are n_{BP} free AZs and n_{BS} free TZs, there are $n_{BP} + 1$ and $n_{BS} + 1$ arcs in both passband and stopbands, respectively. All these zeros x_i are arranged in a specific order on the x -axis. The n_{BP} movable AZs are first placed on the negative side. Then the n_{BS} movable TZs are located on the positive side, and followed by the others, the fixed zeros. The optimality condition is then known to be: If the filter is optimal, then the (weighted) margins (4.18) under/over each arc are equal.

The iterative design procedure, based on the Remez exchange method, is now described. It is similar to that given in [2]. The procedure is carried out in the following steps:

- 1) The iteration index k is initialized, $k = 1$. Then an initial set of zeros $x_i^{(1)}$, ($i = 1, 2, \dots, q$) is selected randomly with

$$q = n_{BP} + n_{BS} \quad (4.19)$$

2) The transformed frequencies $x_j^{(k)}$, ($j = 1, 2, \dots, n_{BP} + 1$) are found such that the margin under each arc of the passband is minimum. Additional transformed frequencies $x_j''^{(k)}$, ($j = 1, 2, \dots, n_{BS} + 1$) are found such that the margin over each arc of the stopbands is minimum. An example of the locations of all the movable AZs and TZs, and the $x_j^{(k)}$, $x_j''^{(k)}$ is shown in Figure 4.2.

3) A new set of zeros $x_i^{(k+1)}$, ($i = 1, 2, \dots, q$) is found such that

$$\beta^{(k+1)}(x_j^{(k)}) = \beta_s(x_j^{(k)}) - W_{BP} \cdot D \quad (j = 1, 2, \dots, n_{BP} + 1) \quad (4.20)$$

$$\beta^{(k+1)}(x_j''^{(k)}) = \beta_s(x_j''^{(k)}) + W_{BS} \cdot D \quad (j = 1, 2, \dots, n_{BS} + 1) \quad (4.21)$$

where $\beta^{(k+1)}$ denotes the β function corresponding to $x_i^{(k+1)}$. This is a system of $q + 2$ nonlinear equations in $q + 2$ unknowns: the q free zeros, C_β and D . This system of equations is solved by linearization around the preceding position of the zeros $x_i^{(k)}$.

4) If all $x_i^{(k+1)} \approx x_i^{(k)}$, ($i = 1, 2, \dots, n$), then the optimal filter function is obtained. Otherwise, k is incremented by 1, $k := k + 1$. Step 2 is repeated. A new set of $x_j^{(k)}$ and $x_j''^{(k)}$ is found and so on. Note that the TZs can move easily from one stopband to the other. This is an important advantage of working with the x variable.

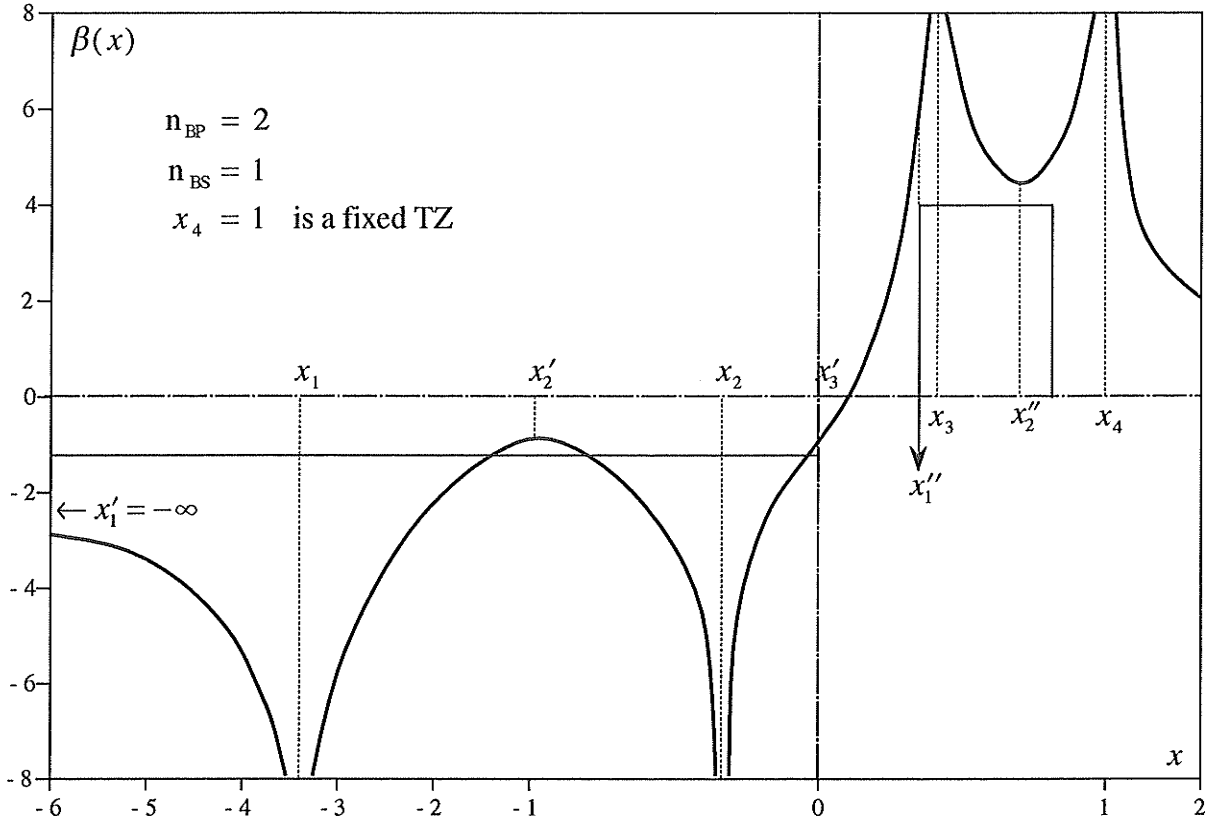


Figure 4.2: The iterative procedure

As described in Step 3, the system of equations have to be linearized in order to solve for the zeros. Substitute (4.16) into (4.20), which becomes

$$C_\beta - \sum_{i=1}^{n+2} m'_i \cdot \ln |x_j^{(k)} - x_i^{(k+1)}| = \beta_s(x_j^{(k)}) - W_{BP} \cdot D \quad (j = 1, 2, \dots, n_{BP} + 1) \quad (4.22)$$

Consider

$$F(x_i^{(k+1)}) = \ln |x_j^{(k)} - x_i^{(k+1)}| \quad (4.23)$$

Using the Taylor series expansion to linearize $F(x_i^{(k+1)})$ around the preceding position of the zeros, $x_i^{(k)}$, it becomes

$$\begin{aligned} F(x_i^{(k+1)}) &\cong F(x_i^{(k)}) + (x_i^{(k+1)} - x_i^{(k)}) \cdot F'(x_i^{(k)}) \\ &= \ln |x_j^{(k)} - x_i^{(k)}| - \frac{x_i^{(k+1)} - x_i^{(k)}}{x_j^{(k)} - x_i^{(k)}} \end{aligned}$$

$$F(x_i^{(k+1)}) = \ln|x_j^{(k)} - x_i^{(k)}| - \frac{x_i^{(k+1)}}{x_j^{(k)} - x_i^{(k)}} + \frac{x_i^{(k)}}{x_j^{(k)} - x_i^{(k)}} \quad (4.24)$$

Substitute (4.24) back into (4.22) to obtain

$$\sum_{i=1}^{n+2} m'_i \cdot \frac{x_i^{(k+1)}}{x_j^{(k)} - x_i^{(k)}} + C_\beta + W_{BP} \cdot D = \beta_s(x_j^{(k)}) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln|x_j^{(k)} - x_i^{(k)}| + \frac{x_i^{(k)}}{x_j^{(k)} - x_i^{(k)}} \right] \quad (j = 1, 2, \dots, n_{BP} + 1) \quad (4.25)$$

Similarly, Equation (4.21) becomes

$$\sum_{i=1}^{n+2} m'_i \cdot \frac{x_i^{(k+1)}}{x_j^{(k)} - x_i^{(k)}} + C_\beta - W_{BS} \cdot D = \beta_s(x_j^{(k)}) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln|x_j^{(k)} - x_i^{(k)}| + \frac{x_i^{(k)}}{x_j^{(k)} - x_i^{(k)}} \right] \quad (j = 1, 2, \dots, n_{BS} + 1) \quad (4.26)$$

As can be seen from Figure 4.2, there are two special cases: $j = 1$ and $j = n_{BP} + 1$; that is, $x'_1 = -\infty$ and $x'_{n_{BP}+1} = 0$, respectively. Consider $x'_j = -\infty$. From (4.25)

$$\frac{x_i^{(k+1)}}{x_j^{(k)} - x_i^{(k)}} = 0 \quad (4.27)$$

It follows that

$$C_\beta + W_{BP} \cdot D = \beta_s(x_j^{(k)}) + \sum_{i=1}^n m'_i \cdot \ln|x_j^{(k)} - x_i^{(k)}| + m'_{n+1} \cdot \ln|x_j^{(k)} - x_{n+1}^{(k)}| + m'_{n+2} \cdot \ln|x_j^{(k)} - x_{n+2}^{(k)}| \quad (4.28)$$

Then from (4.6) and (4.11), it follows that

$$m'_{n+1} = m_\infty = -m_0 - \sum_{i=1}^n m'_i \quad \text{and} \quad m'_{n+2} = m_0 \quad (4.29)$$

Hence, (4.28) becomes

$$C_\beta + W_{BP} \cdot D = \beta_s(x_j^{(k)}) + \sum_{i=1}^n m'_i \cdot \left[\ln |x_j^{(k)} - x_i^{(k)}| - \ln |x_j^{(k)} - x_{n+1}^{(k)}| \right] \\ + m_0 \cdot \left[\ln |x_j^{(k)} - x_{n+1}^{(k)}| - \ln |x_j^{(k)} - x_{n+2}^{(k)}| \right] \quad (4.30)$$

or it can be written as

$$C_\beta + W_{BP} \cdot D = \beta_s(x_j^{(k)}) + \sum_{i=1}^n m'_i \cdot \ln \left| \frac{x_j^{(k)} - x_i^{(k)}}{x_j^{(k)} - x_{n+1}^{(k)}} \right| + m_0 \cdot \ln \left| \frac{x_j^{(k)} - x_{n+1}^{(k)}}{x_j^{(k)} - x_{n+2}^{(k)}} \right| \quad (4.31)$$

As $x'_j = -\infty$, (4.31) will finally become

$$C_\beta + W_{BP} \cdot D = \beta_s(-\infty) \quad (4.32)$$

The other case is $x'_j = 0$. Equation (4.25) becomes

$$\sum_{i=1}^{n+2} m'_i \cdot \frac{x_i^{(k+1)}}{-x_i^{(k)}} + C_\beta + W_{BP} \cdot D = \beta_s(0) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln |x_i^{(k)}| - 1 \right] \quad (4.33)$$

After the linearization, there are $q + 2$ equations from (4.25) and (4.26) in the system which also include the special conditions (4.32) and (4.33). The matrix equation is shown in Figure 4.3. The $x_i^{(k+1)}$, C_β and D can be obtained by solving this matrix equation.

$$\begin{bmatrix}
 0 & \dots & 0 & 1 & W_{BP} \\
 \frac{m'_1}{x_2^{(k)} - x_1^{(k)}} & \dots & \frac{m'_{n+2}}{x_2^{(k)} - x_{n+2}^{(k)}} & 1 & W_{BP} \\
 \vdots & \ddots & \vdots & \vdots & \vdots \\
 \frac{m'_1}{x_{n_{BP}}^{(k)} - x_1^{(k)}} & \dots & \frac{m'_{n+2}}{x_{n_{BP}}^{(k)} - x_{n+2}^{(k)}} & 1 & W_{BP} \\
 \frac{m'_1}{-x_1^{(k)}} & \dots & \frac{m'_{n+2}}{-x_{n+2}^{(k)}} & 1 & W_{BP} \\
 \dots & \dots & \dots & \dots & \dots \\
 \frac{m'_1}{x_1^{(k)} - x_1^{(k)}} & \dots & \frac{m'_{n+2}}{x_1^{(k)} - x_{n+2}^{(k)}} & 1 & -W_{BS} \\
 \vdots & \ddots & \vdots & \vdots & \vdots \\
 \frac{m'_1}{x_{n_{BS}+1}^{(k)} - x_1^{(k)}} & \dots & \frac{m'_{n+2}}{x_{n_{BS}+1}^{(k)} - x_{n+2}^{(k)}} & 1 & -W_{BS}
 \end{bmatrix} \cdot \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_{n+2}^{(k+1)} \\ C_\beta \\ D \end{bmatrix} = \beta_s(-\infty) \begin{bmatrix} \beta_s(x_2^{(k)}) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln |x_2^{(k)} - x_i^{(k)}| + \frac{x_i^{(k)}}{x_2^{(k)} - x_i^{(k)}} \right] \\ \vdots \\ \beta_s(x_{n_{BP}}^{(k)}) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln |x_{n_{BP}}^{(k)} - x_i^{(k)}| + \frac{x_i^{(k)}}{x_{n_{BP}}^{(k)} - x_i^{(k)}} \right] \\ \beta_s(0) + \sum_{i=1}^{n+2} m'_i \cdot [\ln |x_i^{(k)}| - 1] \\ \vdots \\ \beta_s(x_1^{(k)}) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln |x_1^{(k)} - x_i^{(k)}| + \frac{x_i^{(k)}}{x_1^{(k)} - x_i^{(k)}} \right] \\ \vdots \\ \beta_s(x_{n_{BS}+1}^{(k)}) + \sum_{i=1}^{n+2} m'_i \cdot \left[\ln |x_{n_{BS}+1}^{(k)} - x_i^{(k)}| + \frac{x_i^{(k)}}{x_{n_{BS}+1}^{(k)} - x_i^{(k)}} \right] \end{bmatrix}$$

Figure 4.3: Matrix created by (4.25) and (4.26)

4.4 Determination of the Transfer Function

Combining (4.3) with (4.7), (4.8) and (4.10), the transfer function becomes

$$H(s) \cdot H(-s) \Big|_{s^2 = \frac{t_2 - x \cdot t_1}{x-1}} = \frac{1}{1 + k(x)}$$

The factorization is achieved by using the x variable to keep the accuracy. Some particular points have to be noticed.

- If there exists a singularity of $k(x)$ at infinity and it has a positive multiplicity, it will lead to a TZ at $s = 0$.
- The zero x_{n+1} (if $m_{n+1} > 0$) is transformed into $s = \infty$.
- In the case that x_i is in the gap between the two stopbands ($G < x_i < A$ on the x -axis of Figure 4.1), it indicates that the corresponding y_i is imaginary. The corresponding TZ is real. If the kind of filter is not expected, the zero must be fixed by either placing it on a gap edge or splitting it into two zeros of smaller multiplicity and placing each on a gap edge. The optimization procedure is then started again.
- Another case is when x_i falls outside the stopbands (i.e., $x_i > B$ or $E < x_i < F$ in Figure 4.1), which indicates that the corresponding y_i is in either transition bands. If this is not allowed, that zero will be shifted to the nearest edge. Details of this situation and the one before will be explained in the next chapter.
- In (4.18), W_{BP} and W_{BS} are introduced as two positive constants. They are used to control the margin width under/over the arcs. For example, in order to have maximum margins in the stopbands, they can be set to $W_{BP} = 0$ and $W_{BS} = 1$. It is important that both of them cannot be equal to 0 simultaneously. Otherwise, the matrix in Figure 4.3 will be singular. Thus it cannot be solved for a unique solution for each of the unknowns.

Figure 4.4 illustrate the flow of operations in this algorithm.

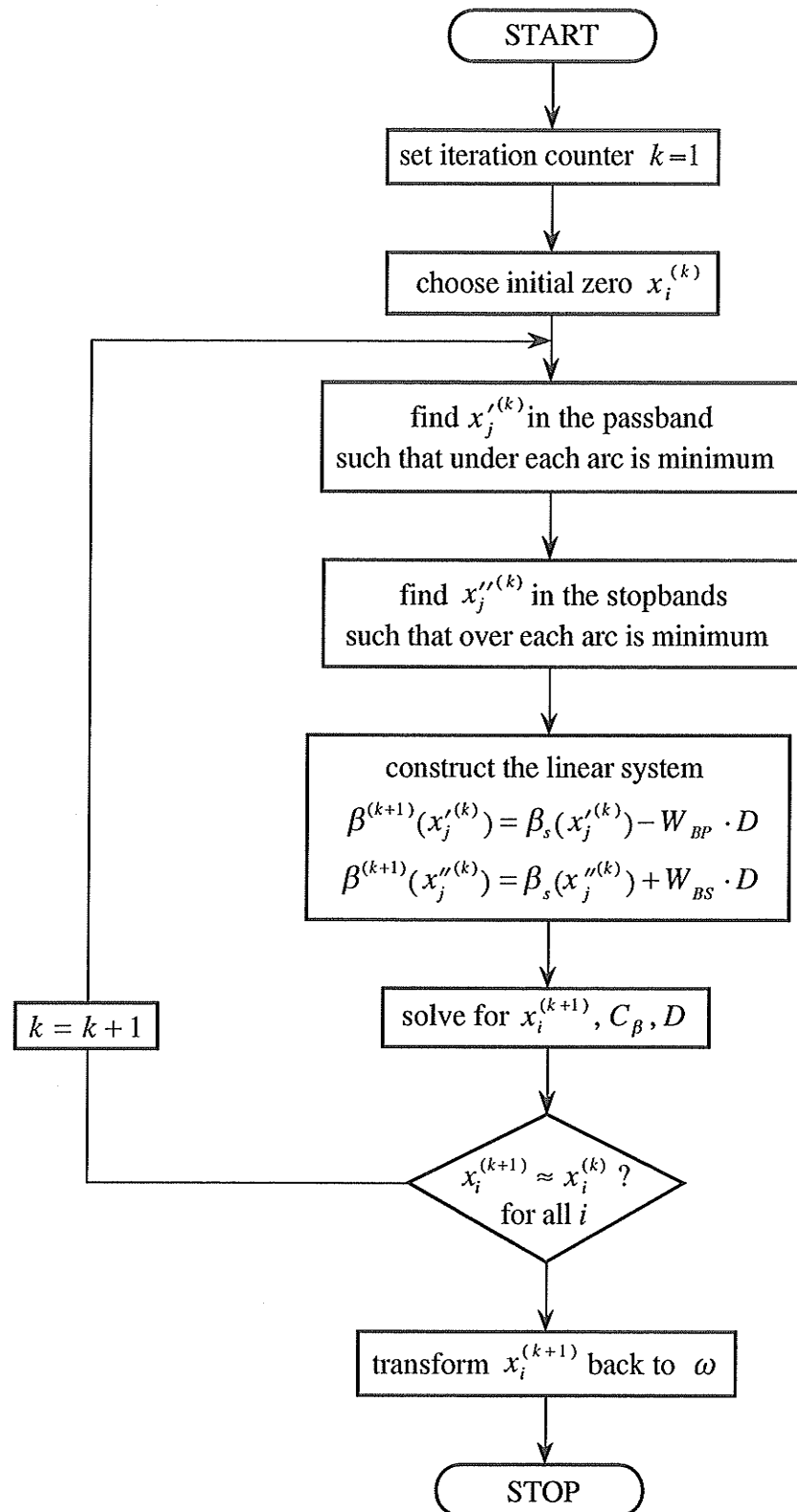


Figure 4.4: The Devleeschouwer & Grenez algorithm

Chapter 5

Experimental Results and Discussion

In this chapter, the approximation techniques described in this thesis are applied to several problems. Each example will be run on different sets of initial values in order to show the convergence of each technique. Both of the techniques are implemented in computer programs. The listings of the programs are given in Appendix A and B. The format of the inputs are also included. Some modifications are applied in order to improve the efficiency of the programs. The details of the modification will be discussed. All the extrema in both algorithms are found by applying the Golden Section Search [21].

5.1 Program Development of Cohen Algorithm

In the process of the pole-shifting stage, the pole displacement Δp_i is obtained by solving (3.21). In order to guarantee convergence, Δp_i will be kept small in magnitude. The search for the possibly “small” Δp_i is done by normalizing the Δp_i with a “scale” factor. Each time a new Δp_i is found in the pole-shifting stage, a temporary new pole position p'_i is obtained. It is calculated as follows

$$p'_i{}^{(j)} = p_i{}^{(k)} + \Delta p_i{}^{(k)} \cdot \frac{j}{S} \quad i = 1, 2, \dots, q/2 \quad (5.1)$$

where $p_i{}^{(k)}$ is the pole position at k^{th} iteration; $\Delta p_i{}^{(k)}$ is the pole displacement at k^{th} iteration. S is an integer scaling factor, and j is an index counter for the ratio of the scale factor, defined in the range $(1, 2, \dots, S)$. $p'_i{}^{(j)}$ is the temporary new pole position that corresponds to the j^{th} scale. (5.1) will give a new pole position with a (maybe “small”) shift in the same direction of $\Delta p_i{}^{(k)}$ for every i .

For every $p'_i{}^{(j)}$, the *Least-Square Distance* will be calculated only when the new rational function is under the floor function according to

$$\|d_{stop}^{(j)}\|^2 = \sum_{i=1}^{q/2} \|g_v^{(k)}(\gamma_i^{(j)}) - F(\gamma_i^{(j)})\|^2 \quad (5.2)$$

where $\gamma_i^{(j)}$ is the abscissa corresponding to the j^{th} scale ratio, which is the minimum point from $g_v^{(k)}(\gamma_i^{(j)}) - F(\gamma_i^{(j)})$. $d_{stop}^{(j)}$ is the least-square distance on the stopband associated with the j^{th} scale. $g_v^{(k)}(x)$ is the same as $g^{(k)}(x)$ in the pole-shifting stage except that the pole is replaced by $p_i'^{(j)}$. It is defined as

$$g_v^{(k)}(x) = a_n \cdot \frac{\prod_{i=1}^{n/2} (x - z_i^{(0)})^2}{\prod_{i=1}^{q/2} (x - p_i'^{(j)})^2} \quad (5.3)$$

The least-square distance, $d_{pass}^{(j)}$ is found the same way except that it is over the passband

$$\|d_{pass}^{(j)}\|^2 = \sum_{i=1}^{n/2} \|g_v^{(k)}(\varphi_i^{(j)}) - C(\varphi_i^{(j)})\|^2 \quad (5.4)$$

where $\varphi_i^{(j)}$ is the abscissa corresponding to the j^{th} scale and is located such that $g_v^{(k)}(\varphi_i^{(j)}) - C(\varphi_i^{(j)})$ is maximized. After a new pole-shifting, the rational function may not satisfy condition (3.6a). Any abscissa which is over the passband is used to calculate this least-square distance. Next, $d_{edge}^{(j)}$, the least-square distance of both the stopband edges, is defined as

$$\|d_{edge}^{(j)}\|^2 = \|g_v^{(k)}(x_{-s}) - F(x_{-s})\|^2 + \|g_v^{(k)}(x_s) - F(x_s)\|^2 \quad (5.5)$$

This is also true only if $g_v^{(k)}(x) < F(x)$ for each of the stopband edges. Otherwise 0 is assumed for the corresponding term. This is to ensure that the rational function is above the stopband edges. Finally, the total least-square distance, d , is defined as

$$d^{(j)2} = \|d_{stop}^{(j)}\|^2 + \|d_{pass}^{(j)}\|^2 + \|d_{edge}^{(j)}\|^2 \quad (5.6)$$

The minimum $d^{(j)}$ is chosen with the optimal j^* scale ratio such that the shift will result with the least-square distance on those abscissa which do not satisfy the specifications. The new pole position is then defined as

$$p_i^{(k+1)} = p_i^{(k)} + \Delta p_i^{(k)} \cdot \frac{j^*}{scale} \quad i = 1, 2, \dots, q/2 \quad (5.7)$$

If $\Delta p_i^{(k)} \approx 0$ for all i , the above process will be ignored. In order to obtain a very “small” $\Delta p_i^{(k)}$, scale factor, S , can be set to a large value ($S = 50$ in the program). On the other hand, this will slow down the whole pole-shifting stage.

5.2 Program Development of Devleeschouwer and Grenez Algorithm

As mentioned in Section 4.4, it is possible for an x_i to be in the gap between the two stopbands. Another case is that x_i is out of the stopbands; this corresponds to the transition bands in the y -axis. An example of this situation is shown in Figure 5.1.

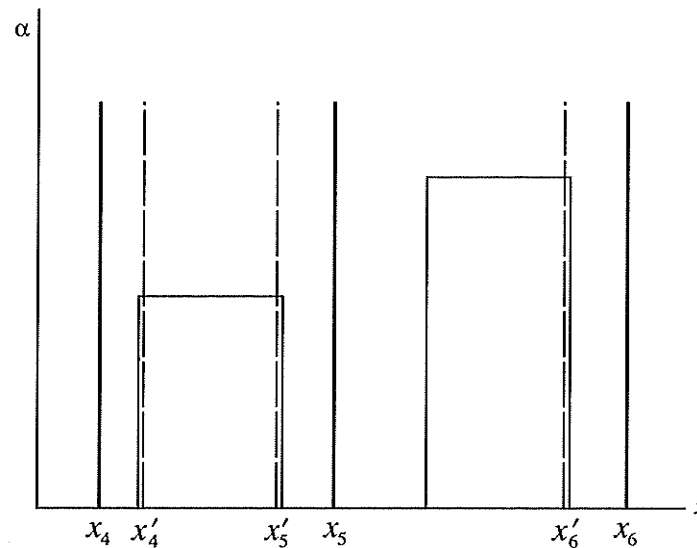


Figure 5.1: Zeros outside stopbands

If this situation is not allowed in the approximation, those zeros which are outside of stopbands, x_4 and x_6 in Figure 5.1, will be shifted to the nearest stopbands edges, i.e., they are shifted to x'_4 and x'_6 , respectively. But the new pole after this shift cannot be allowed to remain on the edge. In Step 2 of the algorithm, a minimum x''_j is found over each arc of the stopbands. If the zero is shifted to the edge, a minimum point will vanish. Therefore, the zero has to be shifted to the edge within a tolerance (0.001 is chosen in the program) in order to preserve the minimum point with that zero.

Similarly, when a zero is located between two stopbands, x_5 from Figure 5.1, it will be shifted to the nearest edge, i.e., it is replaced by x'_5 . The same method is applied to this zero to preserve the minimum. Another way to solve this problem is to split it into two zeros of smaller multiplicity and place one on each gap edge. This method complicates the process because a new zero is created. Thus the first method is preferred in the program.

When searching for the minimum x''_i in the arc which is between the stopbands, there appear two points to be considered for the minimum. This is shown in Figure 5.2.

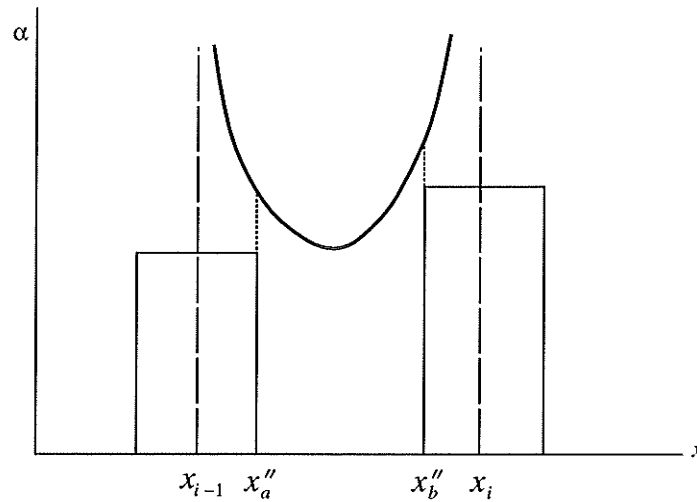


Figure 5.2: An arc crosses between two stopbands

In this situation, the two points on the edges (x''_a and x''_b in Figure 5.2) are considered for the minimum, but only the smaller one is chosen. Another possibility is that either $y \rightarrow \infty$ or $y = 0$, or both may exist. This corresponds to $x_\infty = 1$, $x_0 = t_2/t_1$, respectively. Since any prescribed frequency does not limit an arc, the minimum is searched for in the same way as before. Separate cases are shown in Figure 5.3.

In the figure, it can be seen that between the zeros, x_{i-1} and x_i , only one minimum is considered even though one more arc appears between them. Thus, either x''_a or x''_b is chosen depending on which one is smaller.

In the input specifications, if the attenuation is large, the α function (defined in (4.2)) can be simplified as follows. Combining (4.2), (4.3), and (4.4), it is written as

$$\alpha(f) = 10 \log[1 + k(s)] \qquad s = j2\pi f$$

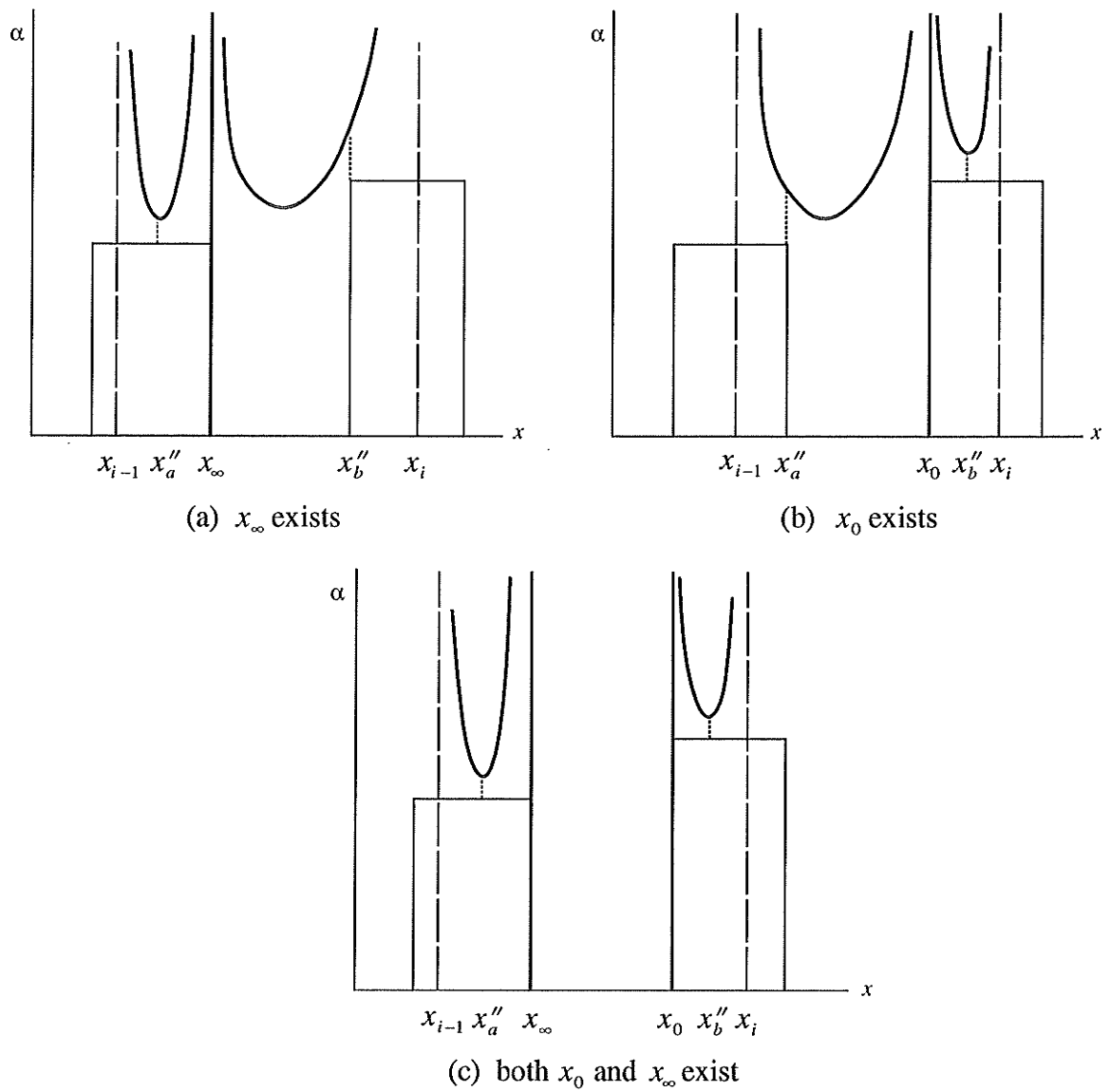


Figure 5.3: Existence of x_0 , x_∞ , or both

For large $k(s)$,

$$\alpha(f) \approx 10 \log[k(s)]$$

or from (4.8)

$$\alpha(y) \approx 10 \log[k(y)]$$

or from (4.10)

$$\alpha(x) \approx 10 \log[k(x)] = 10 \frac{\ln k(x)}{\ln(10)}$$

and finally from (4.15)

$$\alpha(x) \approx 10 \frac{\beta(x)}{\ln(10)}$$

This is applied to the input of the program when $k(s) > 20\text{dB}$.

The least-square distance method is also applied to compare the old set of $x_i^{(k-1)}$ with the new $x_i^{(k)}$. This is to ensure that the process is not repeated indefinitely, especially to prevent the pole from shifting around the stopbands.

5.3 Application of Cohen Algorithm

Example 1: The specification of this example is similar to the one in Figure 2.3. This simple specification is chosen to point out the results on different sets of initial values. The specifications can be seen as dotted lines in all of the figures that follow. The polynomials, f and h , have been frequency normalized by a factor $2\pi \cdot 1000$. The figures are drawn with attenuation (in dB) vs. normalized frequency (in kHz).

Example 1.1: The first example has the following initial values corresponding to the characteristic function in Figure 5.4. The initial zeros are selected to be equally spaced. The initial poles are randomly chosen.

initial abscissas	initial poles
12.34	6.00
12.68	10.00
13.02	16.50
13.36	
13.70	
14.04	
14.38	
14.72	
15.06	

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```
5.254314709150877448e+9
0.000000000000000000 ± 12.04593392530837557 j
0.000000000000000000 ± 12.48506205580107519 j
0.000000000000000000 ± 13.54951724998213224 j
0.000000000000000000 ± 14.78153845245242221 j
0.000000000000000000 ± 15.34690068466112214 j
```

Polynomial f of degree 6

```
1.000000000000000000
0.000000000000000000 ± 10.68306078187241802 j
0.000000000000000000 ± 11.49378424396263164 j
0.000000000000000000 ± 15.71366879340894288 j
```

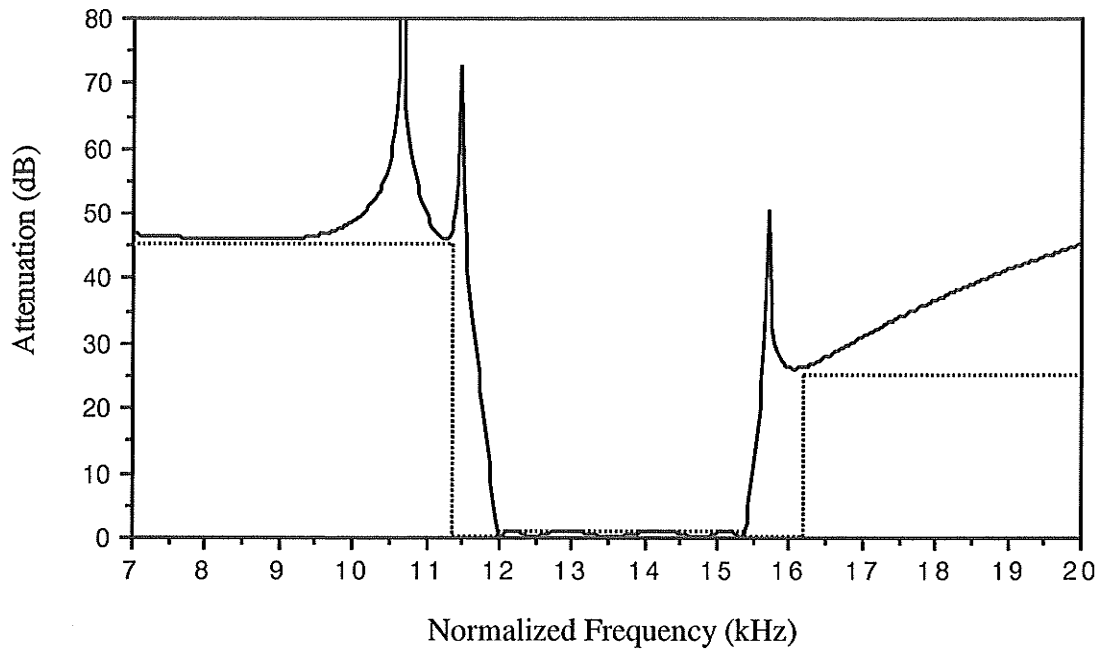


Figure 5.4a: Stopband attenuation of the characteristic function from example 1.1

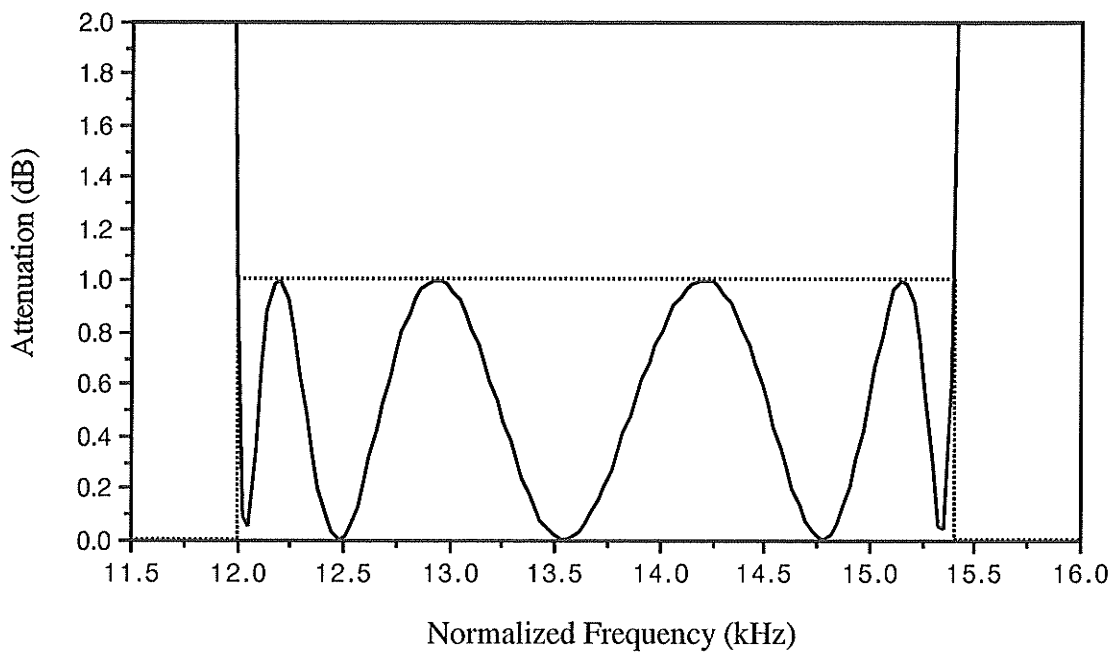


Figure 5.4b: Passband attenuation of the characteristic function from example 1.1

Example 1.2: The following initial values are similar to Example 1.1, except that the initial zeros are purposely chosen close to the lower passband edge. The result is not affected. This is because the interpolation stage generates a unique polynomial for the passband. The characteristic function is shown in Figure 5.5.

initial abscissas	initial poles
12.10	6.00
12.20	10.00
12.30	16.50
12.40	
12.50	
12.60	
12.70	
12.80	
12.90	

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```
5.254314709150877446e+9
0.000000000000000000 ± 12.04593392530837557 j
0.000000000000000000 ± 12.48506205580107519 j
0.000000000000000000 ± 13.54951724998213224 j
0.000000000000000000 ± 14.78153845245242221 j
0.000000000000000000 ± 15.34690068466112214 j
```

Polynomial f of degree 6

```
1.000000000000000000
0.000000000000000000 ± 10.68306078187241802 j
0.000000000000000000 ± 11.49378424396263164 j
0.000000000000000000 ± 15.71366879340894288 j
```

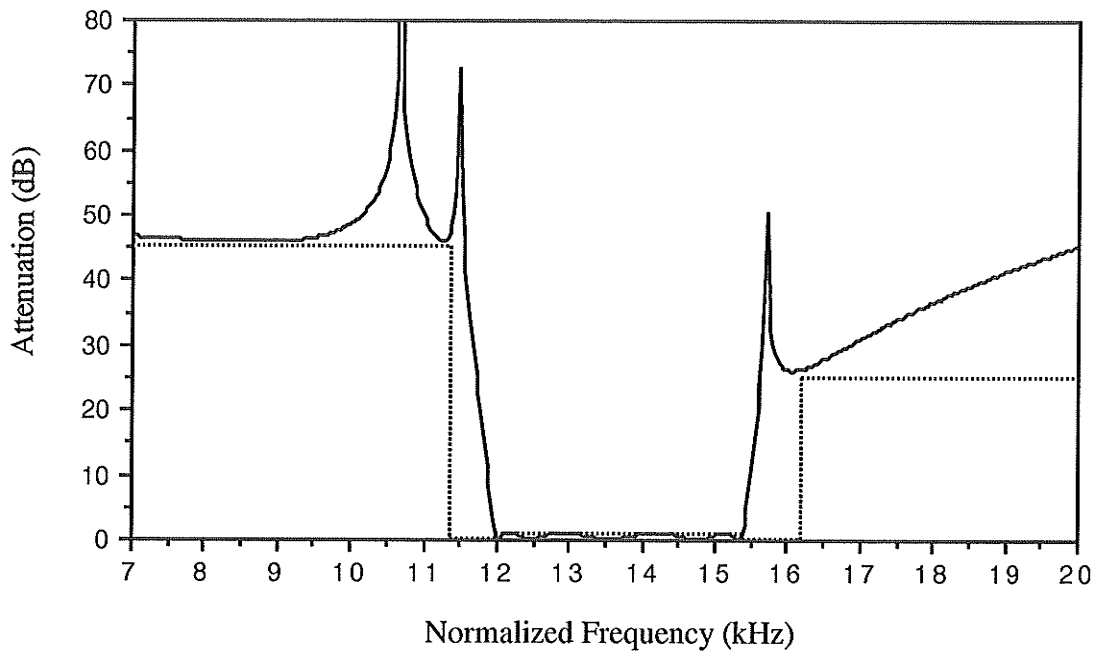



Figure 5.5a: Stopband attenuation of the characteristic function from example 1.2

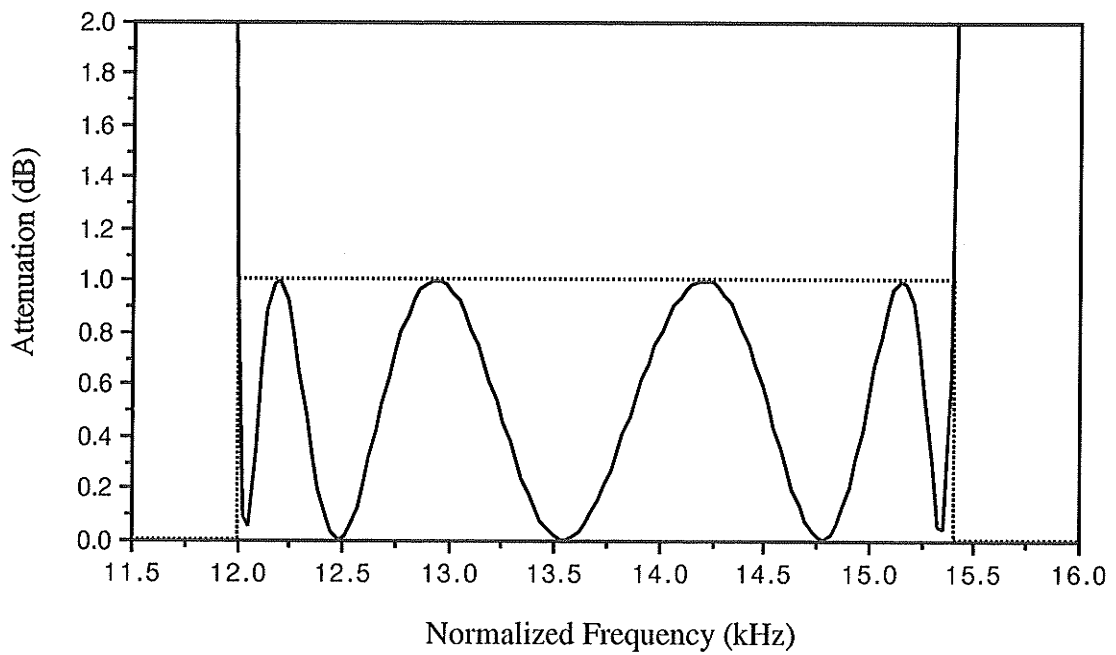


Figure 5.5b: Passband attenuation of the characteristic function from example 1.2

Example 1.3: The initial zeros remain the same as in Example 1.1. This time only the poles are changed. The changes of the positions of the poles can be seen in Figure 5.6.

initial abscissas	initial poles
12.50	8.00
12.80	9.00
13.20	16.50
13.50	
13.90	
14.20	
14.60	
14.90	
15.30	

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```
5.104227019846357370e+9
0.000000000000000000 ± 12.04776732992191037 j
0.000000000000000000 ± 12.49488417803756942 j
0.000000000000000000 ± 13.56192105551688730 j
0.000000000000000000 ± 14.79230940808511863 j
0.000000000000000000 ± 15.34879366781494345 j
```

Polynomial f of degree 6

```
1.000000000000000000
0.000000000000000000 ± 10.92645573297463075 j
0.000000000000000000 ± 11.34925460874910487 j
0.000000000000000000 ± 15.68992354350136796 j
```

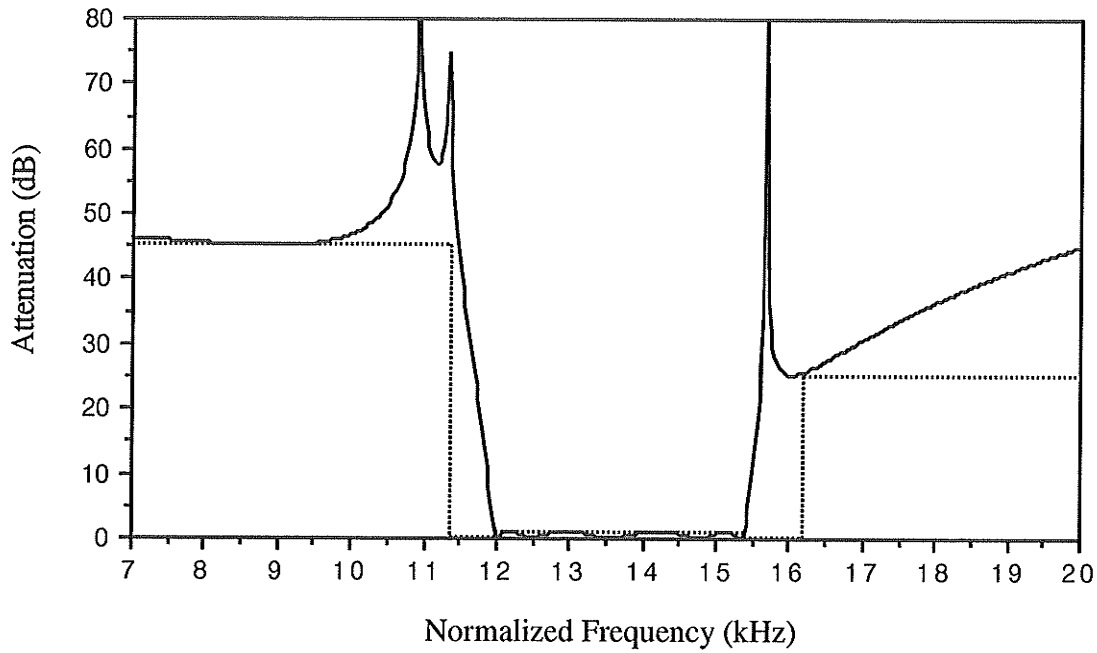


Figure 5.6a: Stopband attenuation of the characteristic function from example 1.3

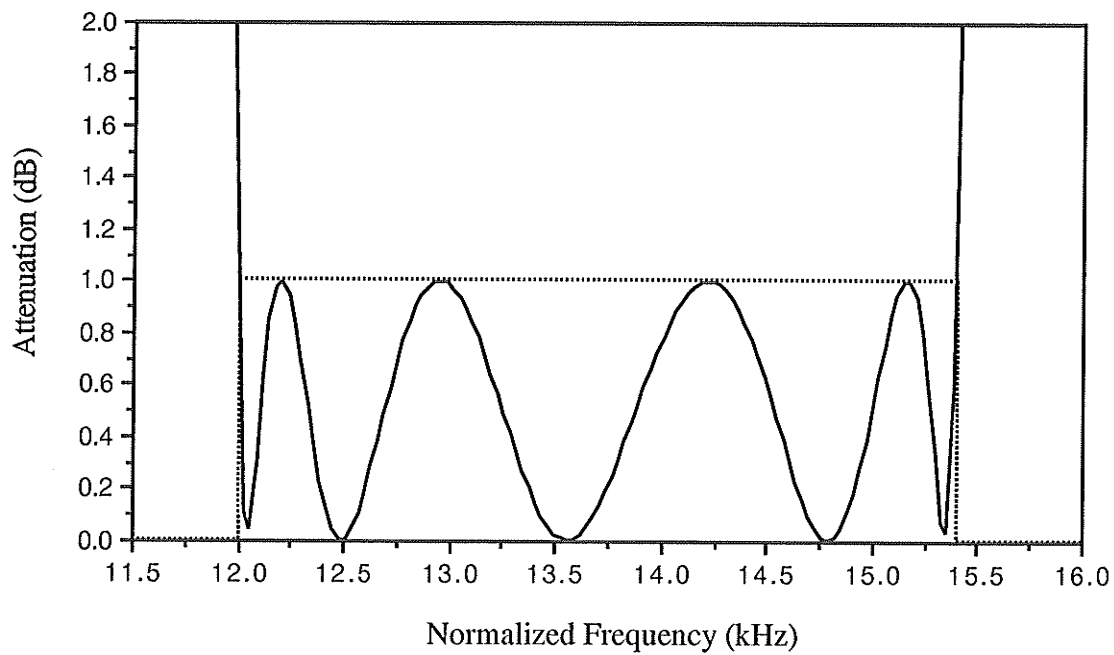


Figure 5.6b: Passband attenuation of the characteristic function from example 1.3

Example 2: The specification of this example is taken from [12]. This example presents multi-level specifications in the passband as well as in the stopbands. The polynomials, f and h , have been frequency normalized by a factor $2\pi \cdot 1000$.

Example 2.1: Since this example has multi-level specifications, a higher order polynomial is required. The input for this example has the following initial values and corresponds to the characteristic function in Figure 5.7.

initial abscissas	initial poles
5.40	2.00
5.80	3.00
6.20	4.00
6.60	15.00
7.00	20.00
7.40	
7.80	
8.20	
8.60	
9.00	
9.40	

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 12

```
4.741986041618690390e+5
0.000000000000000000 ± 5.075738544221072682 j
0.000000000000000000 ± 5.351439306085449912 j
0.000000000000000000 ± 6.330775193348349394 j
0.000000000000000000 ± 7.736746515987279962 j
0.000000000000000000 ± 9.283829954784733567 j
0.000000000000000000 ± 9.889094021931434663 j
```

Polynomial f of degree 10

```
1.000000000000000000
0.000000000000000000 ± 2.883986683076854059 j
0.000000000000000000 ± 4.402573778302598261 j
0.000000000000000000 ± 4.817078431632080558 j
0.000000000000000000 ± 10.503072349953212100 j
0.000000000000000000 ± 12.016036471391500560 j
```

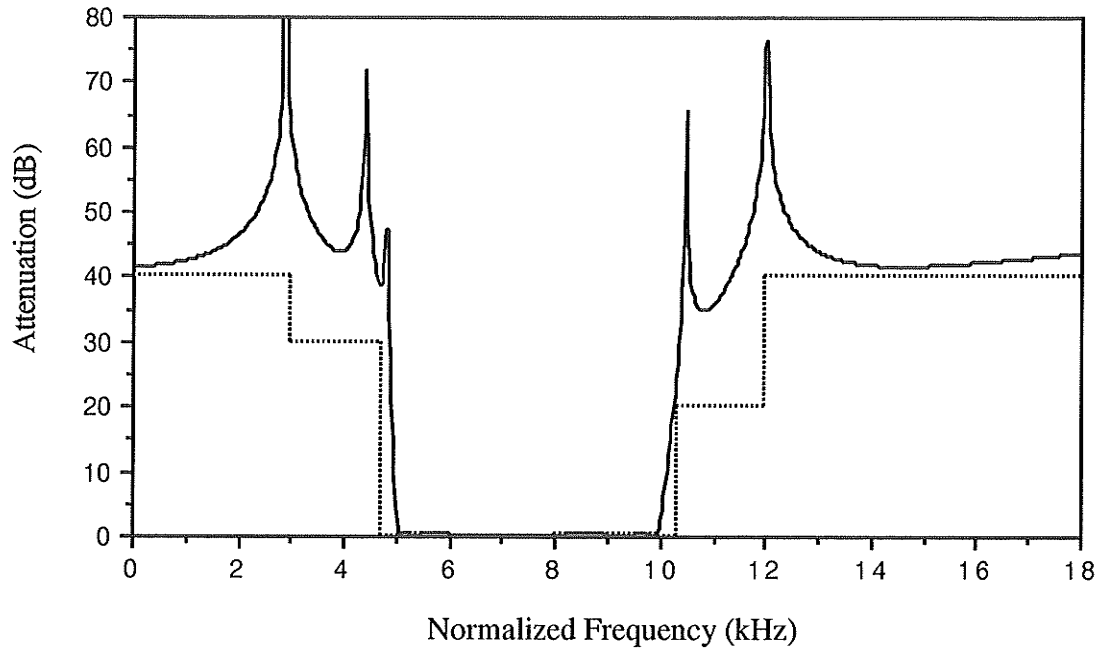


Figure 5.7a: Stopband attenuation of the characteristic function from example 2.1

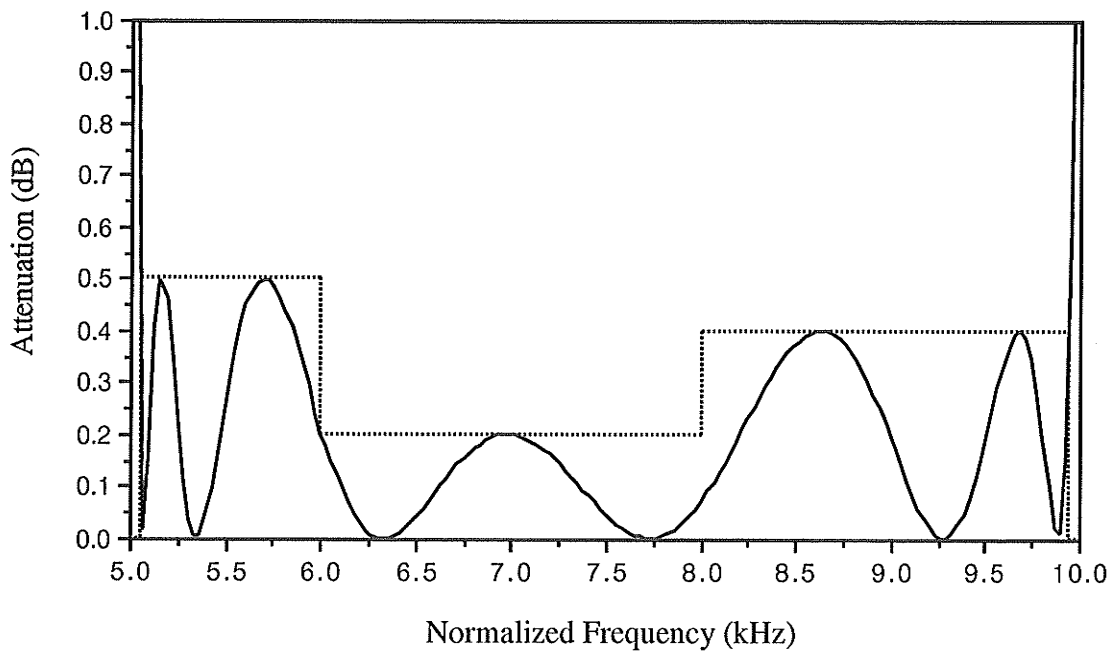


Figure 5.7b: Passband attenuation of the characteristic function from example 2.1

Example 2.2: This time different poles are chosen and the zeros are kept the same as Example 2.1, since in Example 1.2, it was shown that no matter how the zeros are changed, the result does not change much. The corresponding characteristic function is shown in Figure 5.8.

initial abscissas	initial poles
5.40	2.00
5.80	3.00
6.20	4.00
6.60	10.50
7.00	15.00
7.40	
7.80	
8.20	
8.60	
9.00	
9.40	

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 12

```
5.699596799136046096e+5
0.000000000000000000 ± 5.088392389221405558 j
0.000000000000000000 ± 5.493153603250590484 j
0.000000000000000000 ± 6.428704541966253526 j
0.000000000000000000 ± 7.856346860394778388 j
0.000000000000000000 ± 9.310297140419867495 j
0.000000000000000000 ± 9.889142048767258215 j
```

Polynomial f of degree 10

```
1.000000000000000000
0.000000000000000000 ± 2.749114031461626753 j
0.000000000000000000 ± 3.810626269537346128 j
0.000000000000000000 ± 4.630075275720158620 j
0.000000000000000000 ± 11.459238635065949390 j
0.000000000000000000 ± 10.689063090562989460 j
```

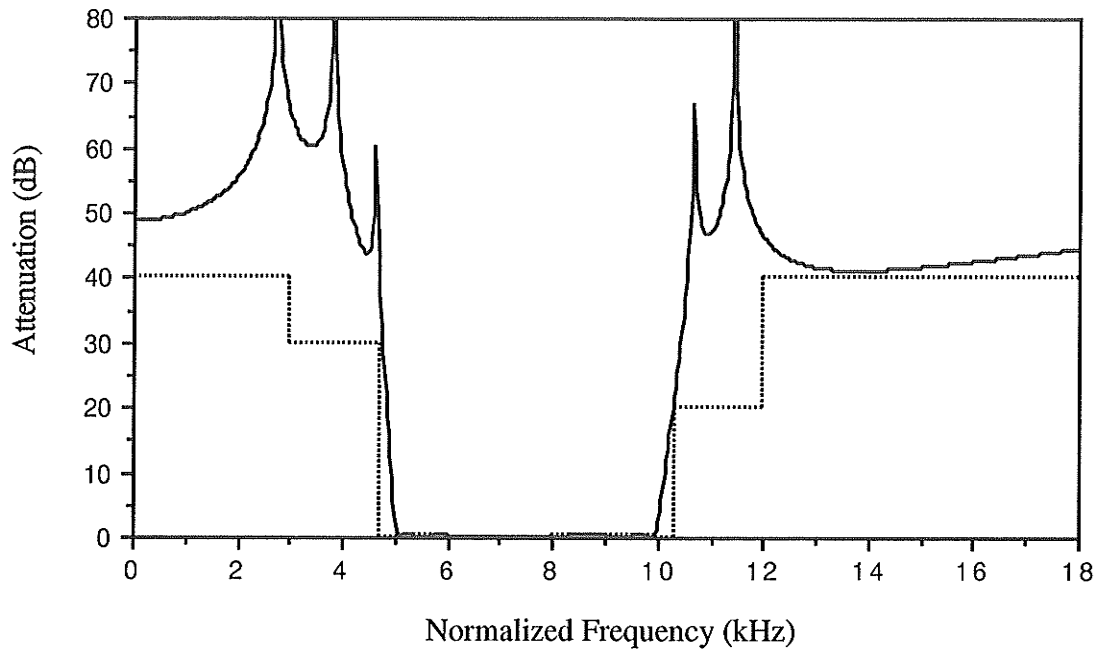


Figure 5.8a: Stopband attenuation of the characteristic function from example 2.2

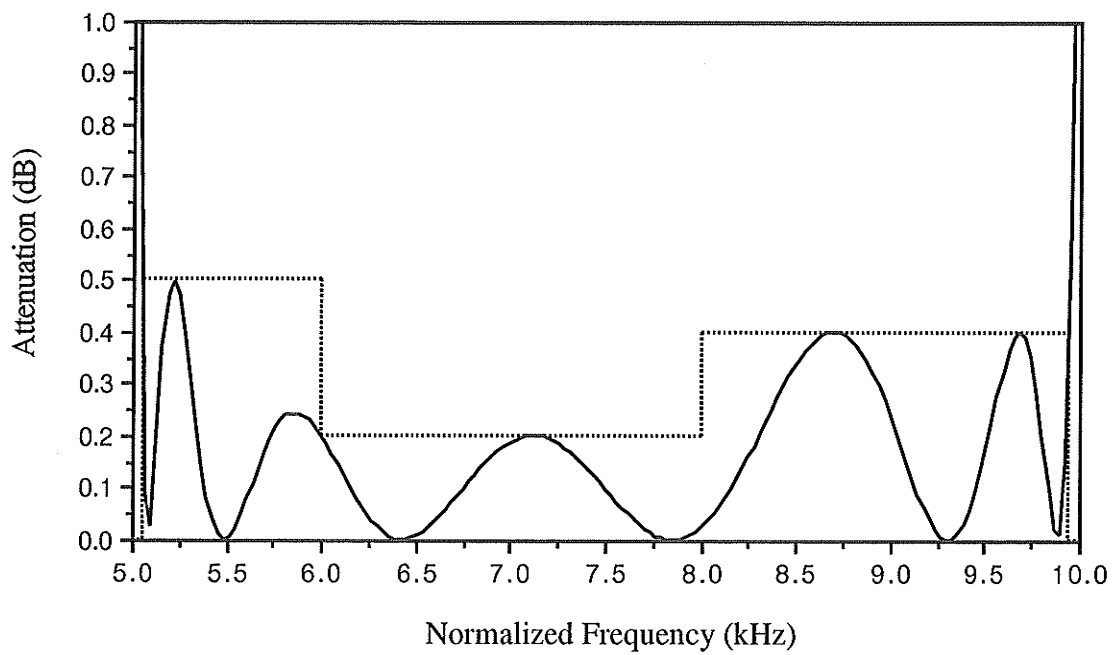


Figure 5.8b: Passband attenuation of the characteristic function from example 2.2

5.4 Application of Devleeschouwer and Grenez Algorithm

Example 3: This specification is the same as in Example 1. The polynomials, f and h , have been frequency normalized by a factor $2\pi \cdot 1000$.

Example 3.1: The initial values in Example 1.1 have been tested and do not generate a similar result. Thus the following values are chosen for the test. The corresponding characteristic function is shown in Figure 5.9.

i	Yi	mi	
1	12.10	-2	(initial zero)
2	12.30	-2	
3	13.70	-2	
4	15.20	-2	
5	15.35	-2	
6	20.00	2	(initial pole)
7	7.00	2	
8	10.00	2	

```

Mo = 0
Cx = 2.00
WBP = 0,   WBS = 1
# of Attenuation Zeros = 5
# of Transmission Zeros = 3
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```

9.211986898592325876e+9
0.000000000000000000 ± 12.04970351444368106 j
0.000000000000000000 ± 12.49947546809637230 j
0.000000000000000000 ± 13.50941024683424771 j
0.000000000000000000 ± 14.68449540168852547 j
0.000000000000000000 ± 15.32731686281930904 j

```

Polynomial f of degree 6

```

1.000000000000000000
0.000000000000000000 ± 10.40503979373769278 j
0.000000000000000000 ± 11.33228663028388318 j
0.000000000000000000 ± 16.22627503558274393 j

```

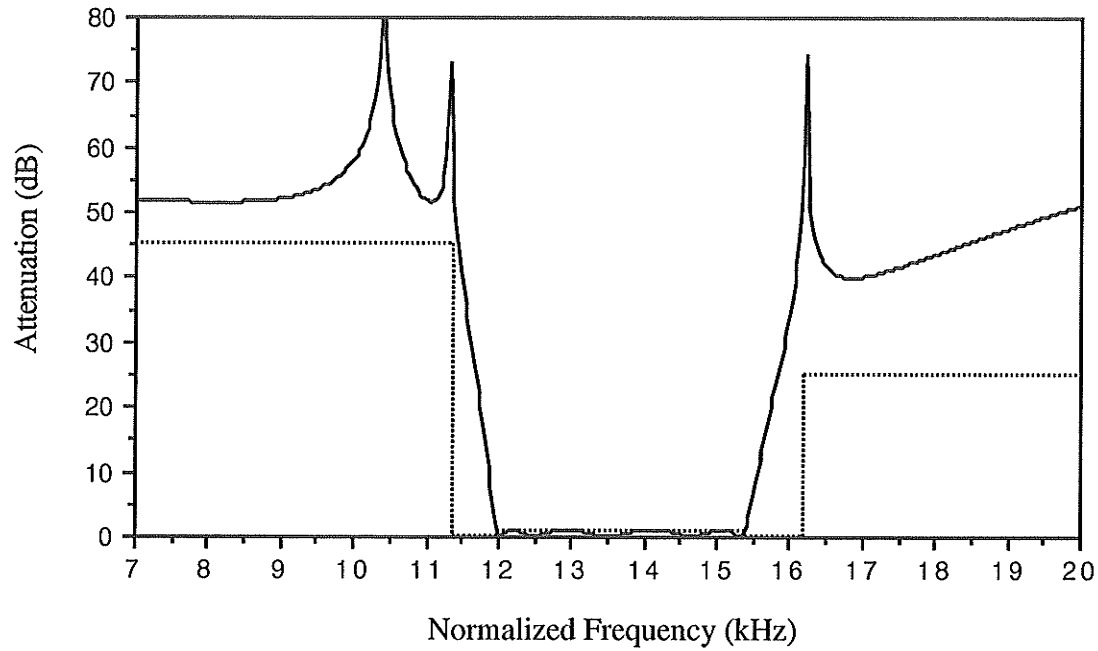



Figure 5.9a: Stopband attenuation of the characteristic function from example 3.1

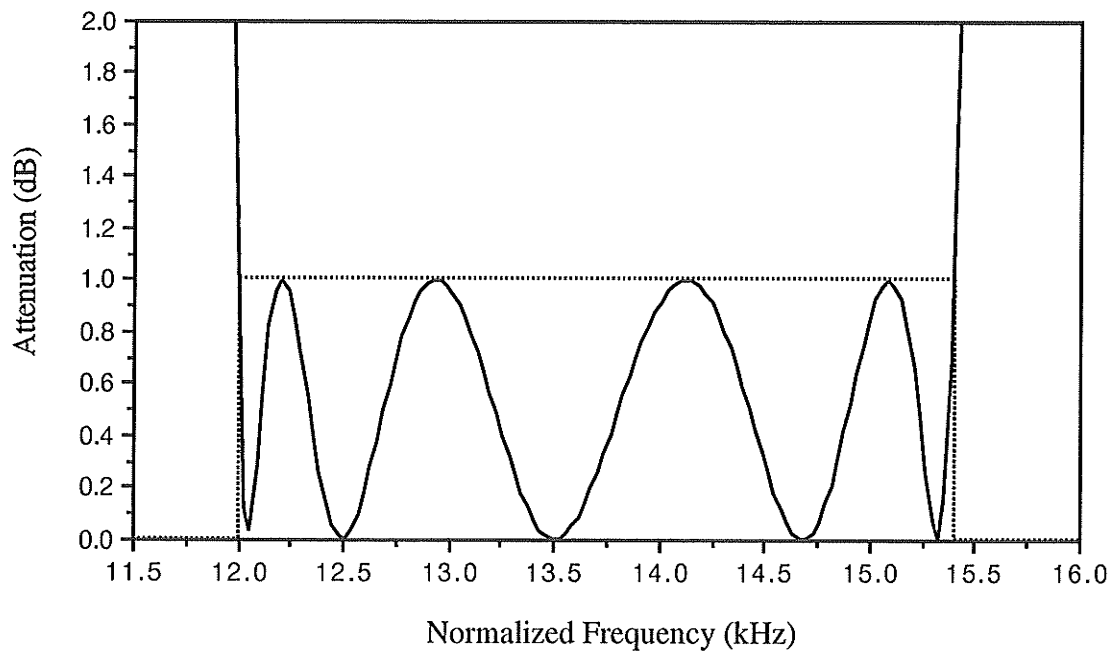


Figure 5.9b: Passband attenuation of the characteristic function from example 3.1

Example 3.2: Based on Example 3.1, the same initial poles are used in this example with unevenly spaced zeros. The corresponding characteristic function is shown in Figure 5.10. The resulting function is very sensitive to the initial values. Even the initial values in example 3.1 have been chosen after many tests.

i	Yi	mi	
1	12.20	-2	(initial zero)
2	12.40	-2	
3	13.70	-2	
4	15.00	-2	
5	15.20	-2	
6	20.00	2	(initial pole)
7	7.00	2	
8	10.00	2	

```

Mo = 0
Cx = 2.00
WBP = 0,   WBS = 1
# of Attenuation Zeros = 5
# of Transmission Zeros = 3
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```

7.378607123826667980e+9
0.000000000000000000 ± 12.06474052935842666 j
0.000000000000000000 ± 12.49334775933006165 j
0.000000000000000000 ± 13.48486461173998572 j
0.000000000000000000 ± 14.65884695931239003 j
0.000000000000000000 ± 15.33244411569819797 j

```

Polynomial f of degree 6

```

1.000000000000000000
0.000000000000000000 ± 10.35696559493572915 j
0.000000000000000000 ± 11.34415307387125688 j
0.000000000000000000 ± 16.20465277918208898 j

```

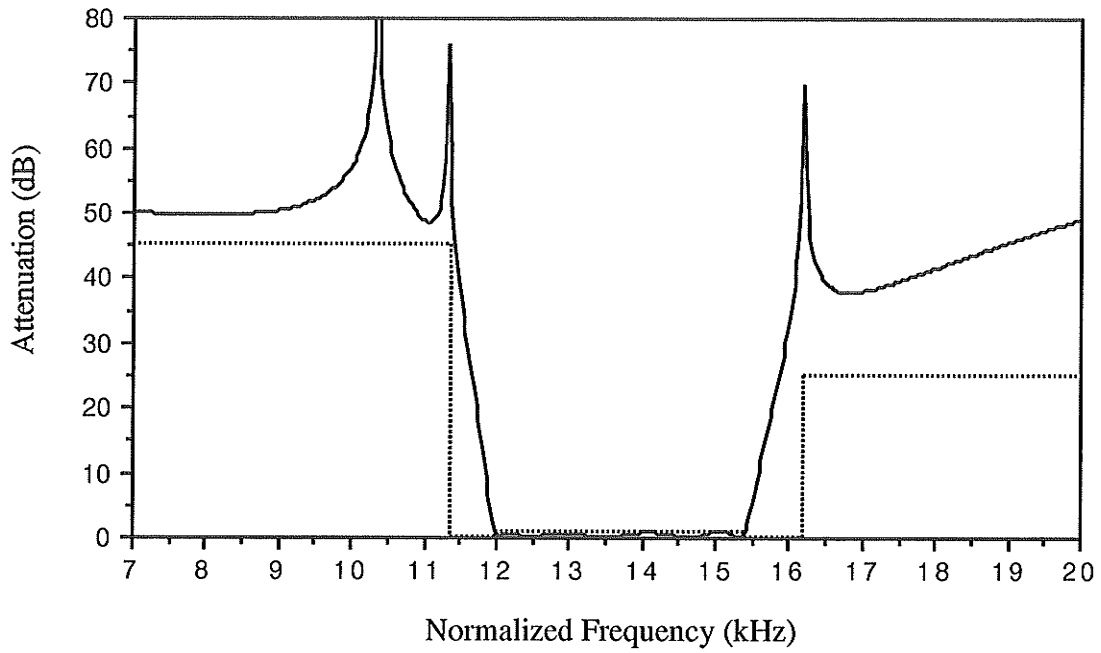


Figure 5.10a: Stopband attenuation of the characteristic function from example 3.2

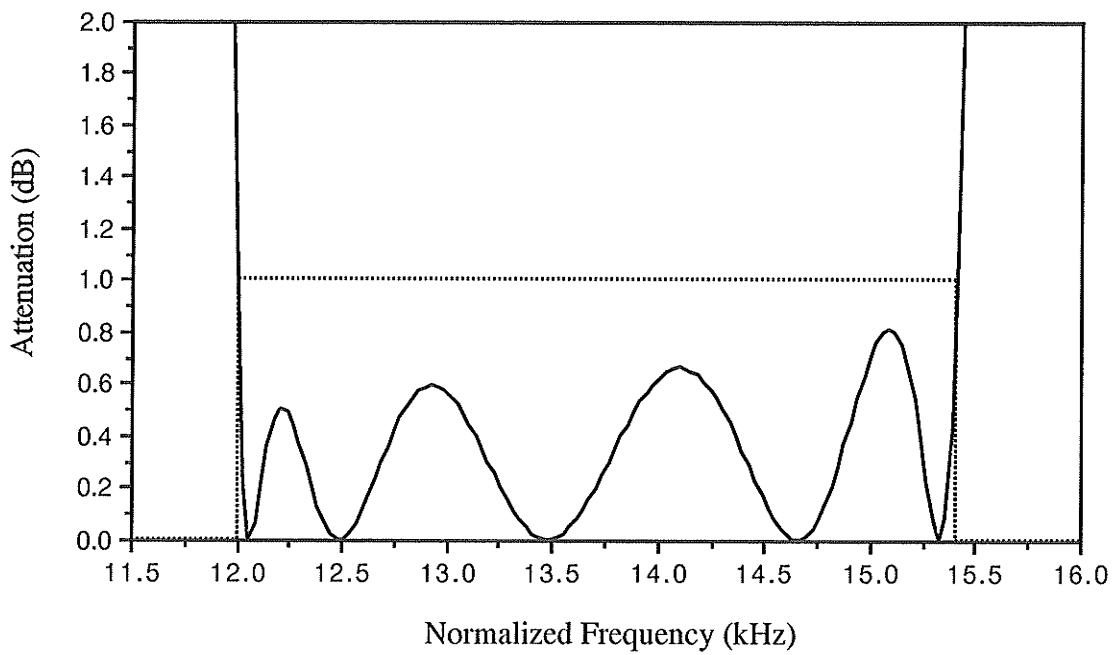


Figure 5.10b: Passband attenuation of the characteristic function from example 3.2

Example 3.3: Based on Example 3.1, the same zeros are used. The initial poles are rearranged to show the effects of the initial pole positions. The corresponding characteristic function is shown in Figure 5.11. The changes of zeros in the passband are significant.

i	Yi	mi	
1	12.10	-2	(initial zero)
2	12.30	-2	
3	13.70	-2	
4	15.20	-2	
5	15.35	-2	
6	18.00	2	(initial pole)
7	8.00	2	
8	11.00	2	

```

Mo = 0
Cx = 2.00
WBP = 0, WBS = 1
# of Attenuation Zeros = 5
# of Transmission Zeros = 3
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```

7.537480448530746570e+9
0.000000000000000000 ± 12.06896365374062065 j
0.000000000000000000 ± 12.51925534395231430 j
0.000000000000000000 ± 13.50927229441838860 j
0.000000000000000000 ± 14.92510320221487189 j
0.000000000000000000 ± 15.30436918613485065 j

```

Polynomial f of degree 6

```

1.000000000000000000
0.000000000000000000 ± 10.12829012649028614 j
0.000000000000000000 ± 11.35959053980312729 j
0.000000000000000000 ± 16.79949450044944089 j

```

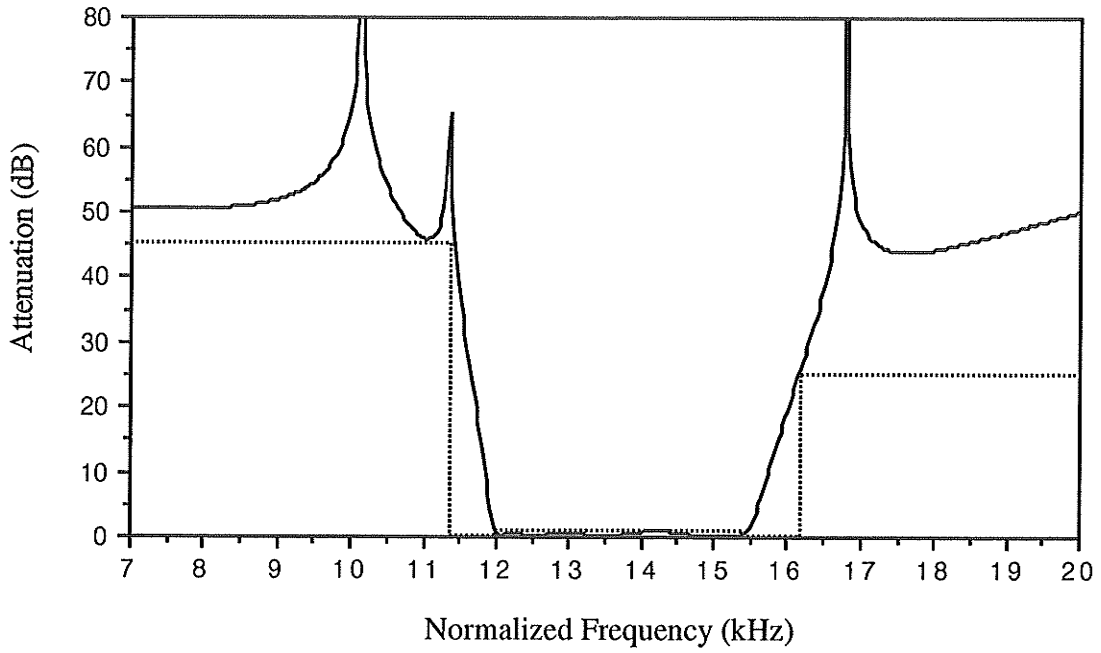


Figure 5.11a: Stopband attenuation of the characteristic function from example 3.3

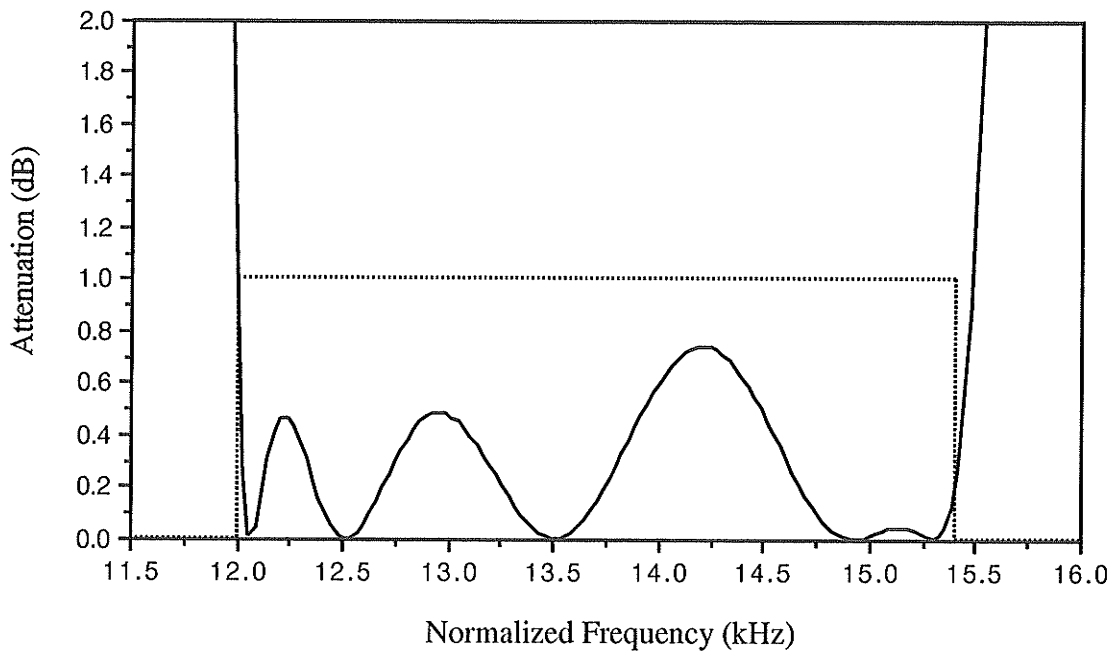


Figure 5.11b: Passband attenuation of the characteristic function from example 3.3

Example 3.4: Based on Example 3.1, m_0 is set to have a pole at $s = 0$. The same initial values as in Example 3.1 are used. The corresponding characteristic function is shown in Figure 5.12.

i	Yi	mi	
1	12.10	-2	(initial zero)
2	12.30	-2	
3	13.70	-2	
4	15.20	-2	
5	15.35	-2	
6	20.00	2	(initial pole)
7	7.00	2	
8	10.00	2	

```

Mo = 2
Cx = 2.00
WBP = 0, WBS = 1
# of Attenuation Zeros = 5
# of Transmission Zeros = 4
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 10

```

1.326345763545256473e+6
0.000000000000000000 ± 12.04675888432228117 j
0.000000000000000000 ± 12.43664097085471716 j
0.000000000000000000 ± 13.39517392476005969 j
0.000000000000000000 ± 14.67909997021228186 j
0.000000000000000000 ± 15.30892599074993180 j

```

Polynomial f of degree 8

```

1.000000000000000000
0.000000000000000000 ± 0.000000000000000000 j
0.000000000000000000 ± 10.74574174536325849 j
0.000000000000000000 ± 11.37461515397993145 j
0.000000000000000000 ± 16.22233831315463527 j

```

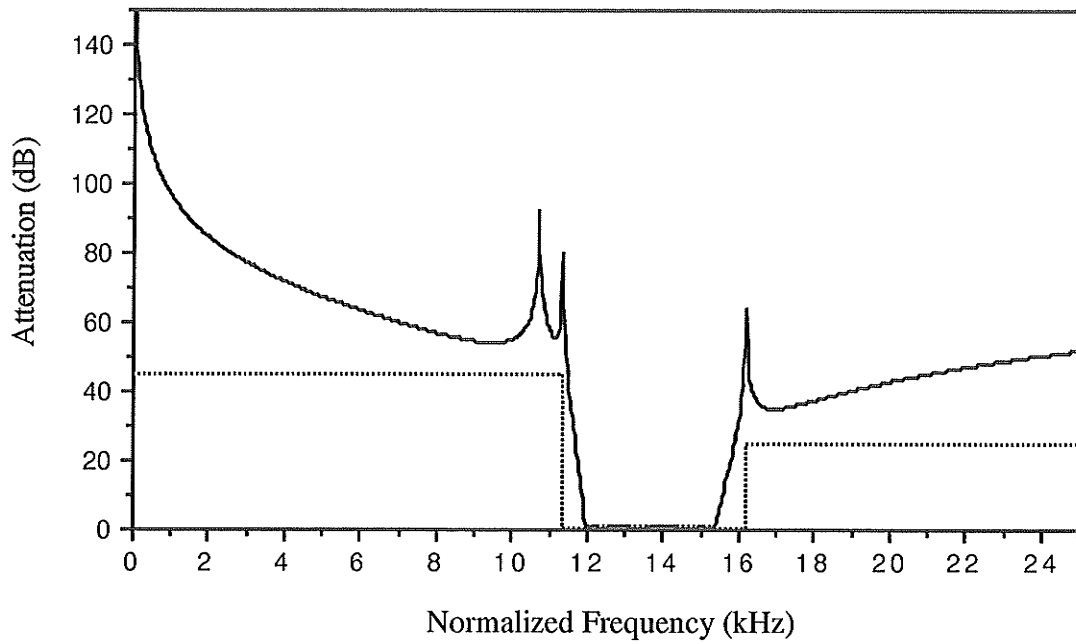


Figure 5.12a: Stopband attenuation of the characteristic function from example 3.4

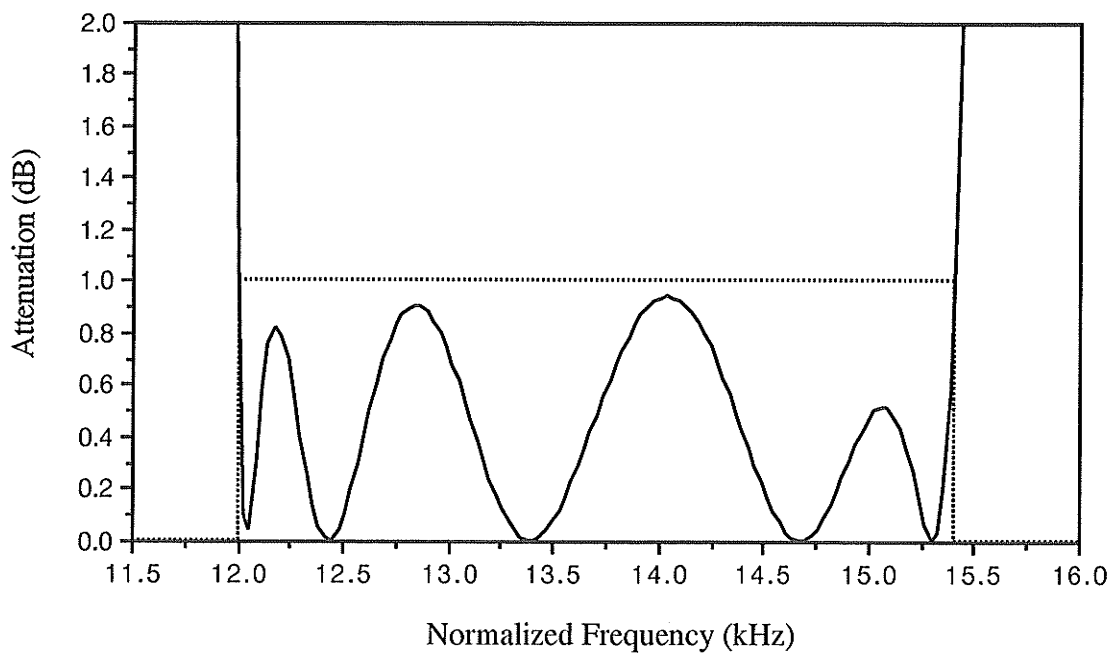


Figure 5.12b: Passband attenuation of the characteristic function from example 3.4

Example 3.5: Based on the example 3.1, the same initial poles and zeros are used in this example with W_{BP} and W_{BS} changed. This shows that the effect of the width of the margins is significant. The corresponding characteristic function is shown in Figure 5.13. Though the result is not obvious, the ripples in the passband are approximately equal in margin width.

i	Yi	mi	
1	12.10	-2	(initial zero)
2	12.30	-2	
3	13.70	-2	
4	15.20	-2	
5	15.35	-2	
6	20.00	2	(initial pole)
7	7.00	2	
8	10.00	2	

```

Mo = 0
Cx = 2.00
WBP = 1,   WBS = 0
# of Attenuation Zeros = 5
# of Transmission Zeros = 4
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

```

Polynomial h of degree 10
5.151336391893201834e+9
0.000000000000000000 ± 12.05204608443945720 j
0.000000000000000000 ± 12.49456681759141532 j
0.000000000000000000 ± 13.50596651663920431 j
0.000000000000000000 ± 14.69598131330014682 j
0.000000000000000000 ± 15.30550299662252417 j

```

```

Polynomial f of degree 6
1.000000000000000000
0.000000000000000000 ± 10.39577617227759463 j
0.000000000000000000 ± 11.35128765761250281 j
0.000000000000000000 ± 16.20465277918208898 j

```

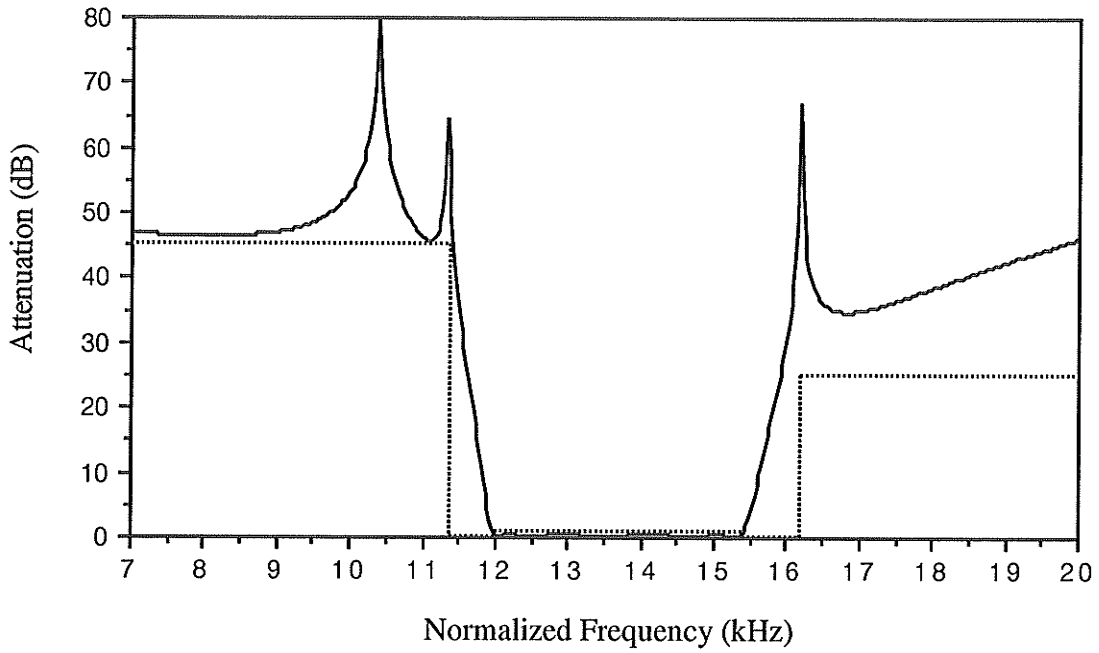



Figure 5.13a: Stopband attenuation of the characteristic function from example 3.5

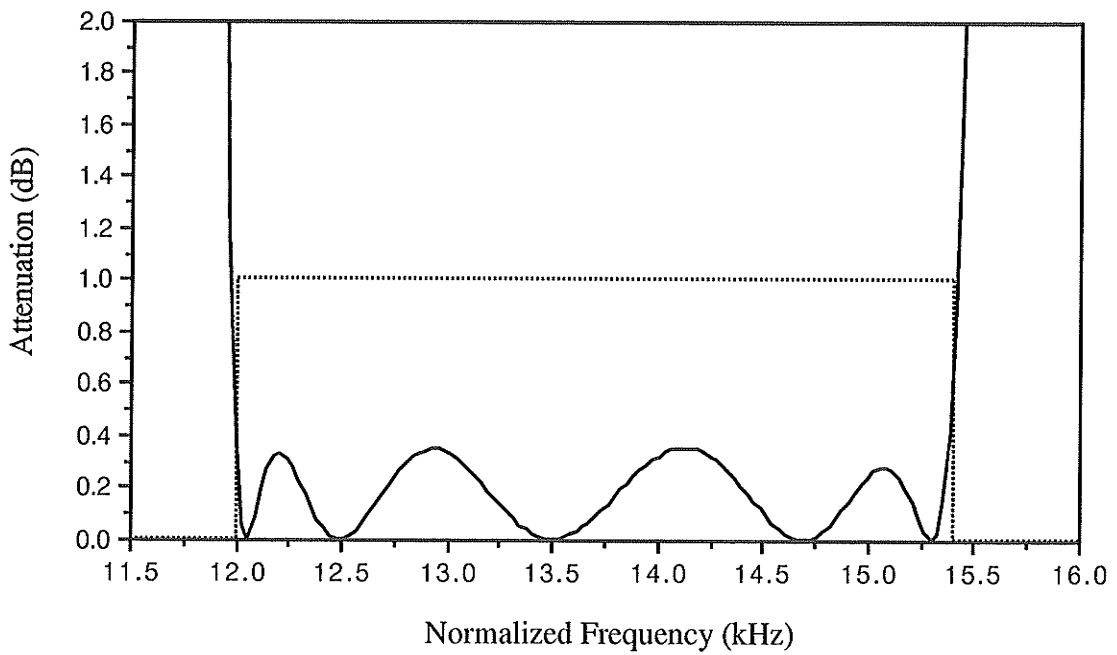


Figure 5.13b: Passband attenuation of the characteristic function from example 3.5

Example 4: This example is the same as Example 2. The polynomials, f and h , have been frequency normalized by a factor $2\pi \cdot 1000$.

Example 4.1: As mentioned in Example 2.1, this approximation requires more poles to meet the multi-leveled specification. To best approximate this specification, a pole is set at $s = 0$. The corresponding characteristic function is shown in Figure 5.14.

i	Yi	mi	
1	5.10	-2	(initial zero)
2	5.40	-2	
3	6.50	-2	
4	7.50	-2	
5	9.00	-2	
6	9.90	-2	
7	11.00	2	(initial pole)
8	13.00	2	
9	2.00	2	
10	4.00	2	

```

Mo = 2
Cx = 2.00
WBP = 0, WBS = 1
# of Attenuation Zeros = 6
# of Transmission Zeros = 4
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 12

```

9.169828570011476454e+5
0.000000000000000000 ± 5.091132919222259876 j
0.000000000000000000 ± 5.533587746194019748 j
0.000000000000000000 ± 6.472616858404367932 j
0.000000000000000000 ± 7.888686913368535970 j
0.000000000000000000 ± 9.333225334831782635 j
0.000000000000000000 ± 9.894813337143080185 j

```

Polynomial f of degree 10

```

1.000000000000000000
0.000000000000000000 ± 0.000000000000000000 j
0.000000000000000000 ± 3.584043653419447275 j
0.000000000000000000 ± 4.647016421258392894 j
0.000000000000000000 ± 10.37677177000781193 j
0.000000000000000000 ± 13.06563546909918193 j

```

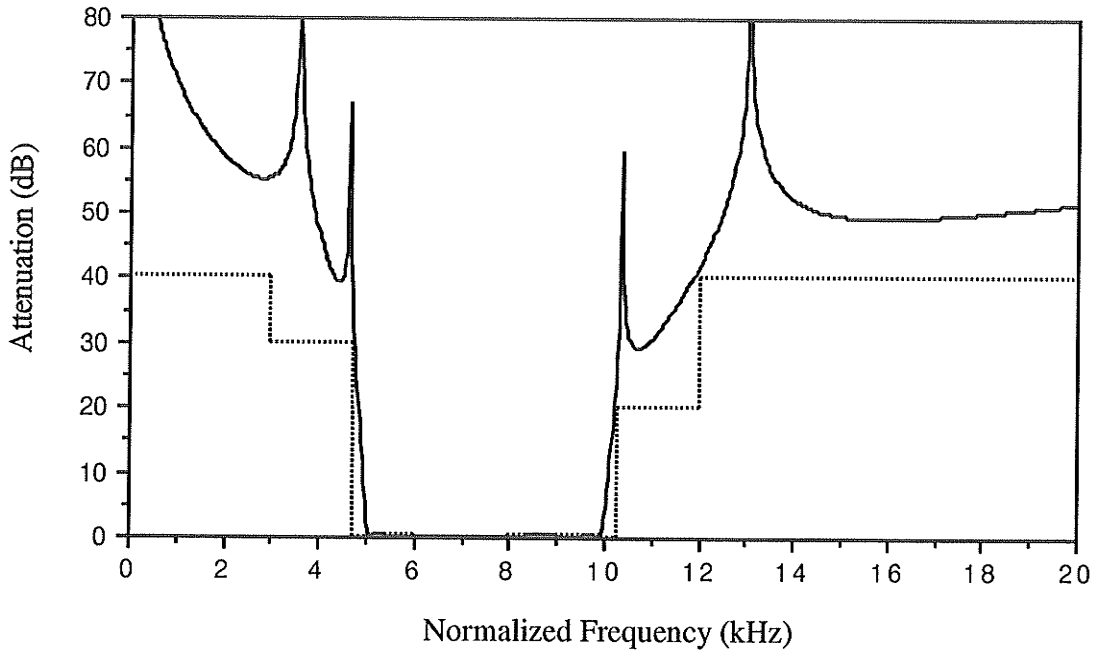


Figure 5.14a: Stopband attenuation of the characteristic function from example 4.1

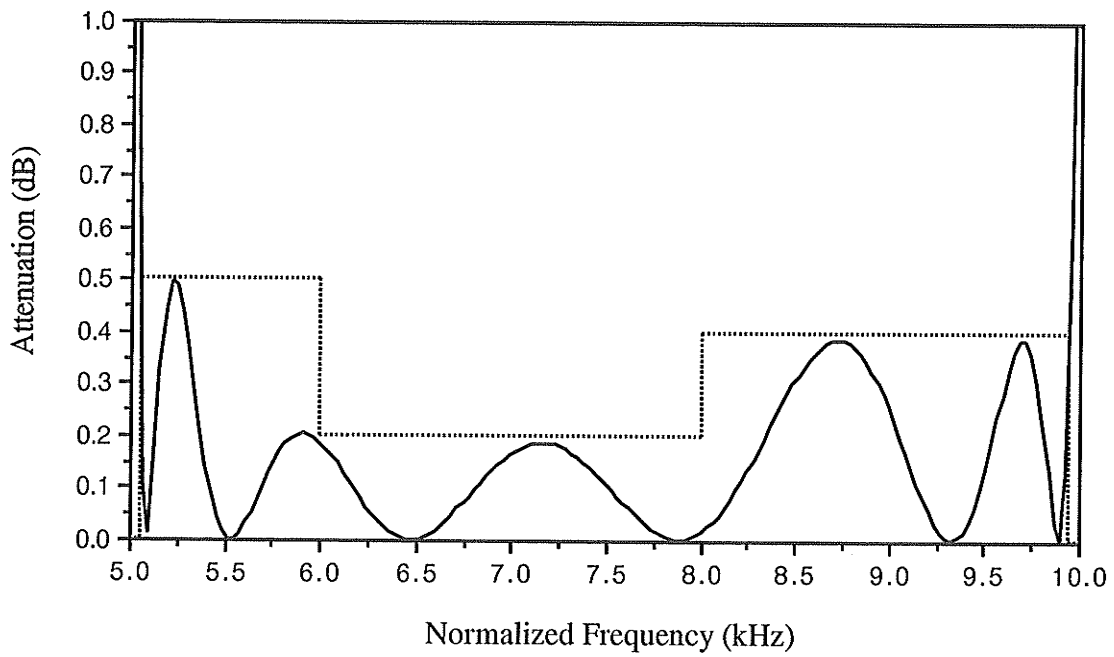


Figure 5.14b: Passband attenuation of the characteristic function from example 4.1

Example 4.2: Random initial values are used instead of fixing the pole. The corresponding characteristic function is shown in Figure 5.15. It can be seen that the ripples in the passband do not reach the upper bounds specification.

i	Yi	mi	
1	5.10	-2	(initial zero)
2	5.40	-2	
3	6.50	-2	
4	7.90	-2	
5	9.30	-2	
6	9.90	-2	
7	10.40	2	(initial pole)
8	13.00	2	
9	24.00	2	
10	1.00	2	
11	4.30	2	

```

Mo = 0
Cx = 2.00
WBP = 0, WBS = 1
# of Attenuation Zeros = 6
# of Transmission Zeros = 5
No real Transmission Zero ? TRUE

```

The characteristic function obtained by the algorithm follows (for each polynomial the constant multiplier is followed by zeros):

Polynomial h of degree 12

```

8.317148817814617386e+5
0.000000000000000000 ± 5.091225890688672102 j
0.000000000000000000 ± 5.536367360595796740 j
0.000000000000000000 ± 6.478247994831224182 j
0.000000000000000000 ± 7.897087949957372874 j
0.000000000000000000 ± 9.336393079449026313 j
0.000000000000000000 ± 9.895060666081495431 j

```

Polynomial f of degree 10

```

1.000000000000000000
0.000000000000000000 ± 0.09405107320175541973 j
0.000000000000000000 ± 3.70894302495113588100 j
0.000000000000000000 ± 4.64765707179966752000 j
0.000000000000000000 ± 10.37647886476979095000 j
0.000000000000000000 ± 13.04546018741714027000 j

```

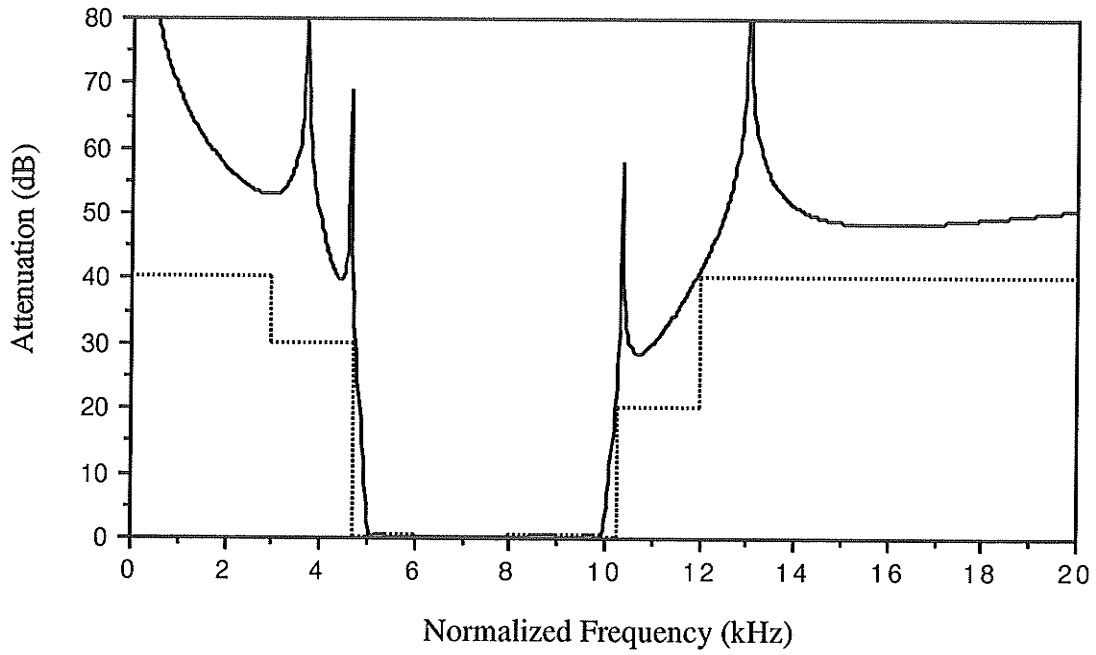


Figure 5.15a: Stopband attenuation of the characteristic function from example 4.2

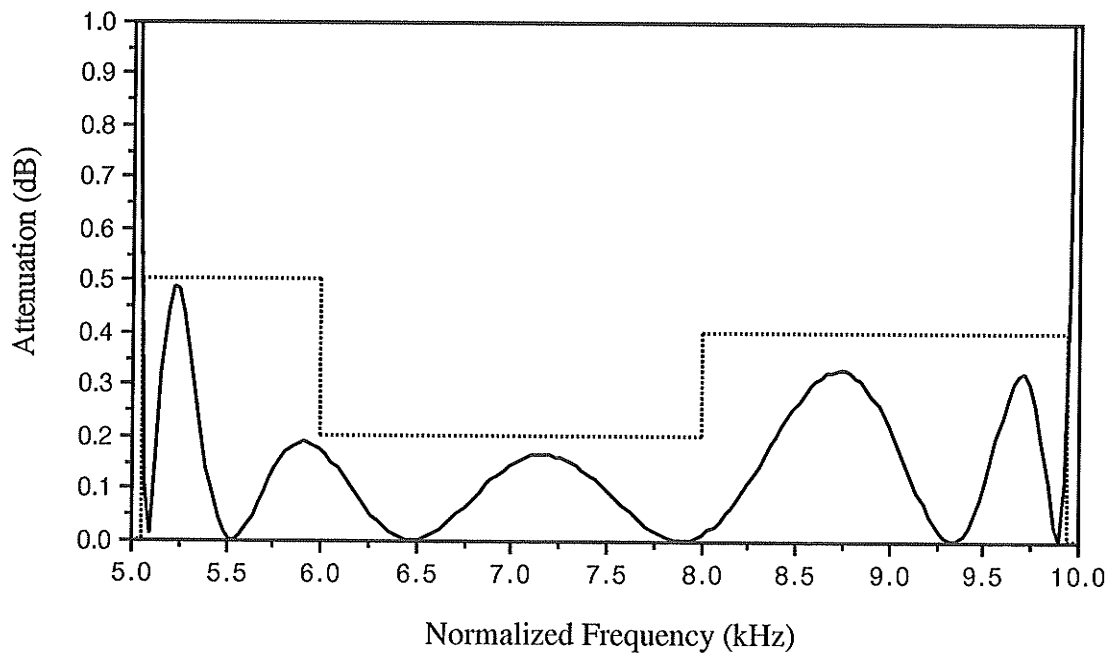


Figure 5.15b: Passband attenuation of the characteristic function from example 4.2

5.5 Discussion

From the results of the last two sections, some significant features of these two algorithms can be deduced. The Cohen algorithm always generates rational functions which meet the passband specifications with the maximum points equal to the upper bounds. But this is not true in the stopbands, since there is no indication as to how to choose the initial poles. This requires several pole shifts to lower the function. It may not guarantee that the pole positions are optimum. Thus, the initial pole positions are very important.

The choice of the initial abscissas does not cause a problem in the interpolation stage. The interpolation stage takes most of the approximating time especially if the degree of the polynomials is very high. The product form for polynomials is used to avoid high coefficient sensitivity in the addition of polynomials in the application of Newton's Interpolation Algorithm, thus requiring considerable time for zero-finding.

The Devleeschouwer and Grenez algorithm uses less time for the approximation. This is because the rational function is transformed into logarithmic form. This greatly reduces the evaluation time for the function when it is of high degree. The algorithm, however, requires the weighted margins for the stopbands and passband. The resulting function will not be equal to the upper/lower bounds of the specifications. Also the initial values cannot be randomly chosen. This algorithm is very sensitive to the initial values. The initial values in the last two section have been chosen after many tests in order to give the best results. In some cases, the zeros are shifted to the wrong side of the axis (for example, TZ move to the negative side of the x -axis).

Chapter 6

Conclusions

In filter design, a very common problem is to find the characteristic function. In general, this is easier for lowpass filters than for bandpass filters. Usually, bandpass filters are obtained through frequency transformation. This limits the specifications of such a filter. In this thesis, two approximation techniques are investigated. Though they are not limited to the approximation of bandpass filters; in general, they can be applied to all kinds of filters using the same technique.

The Cohen algorithm separates the approximation of zeros and poles into two stages. An interpolation algorithm is applied to generate the zeros of the characteristic function. The approximation of the pole positions is done by linearization. This simplifies the process by defining the locations of the zeros first, then approximating the poles. But there is no convergence guarantee for obtaining the function with randomly chosen initial values. Also the function is introduced in rational product-form, which requires a zero-finding routine for keeping the polynomials in product-form during the interpolation process. Most of the approximation time is spent on that routine. A more efficient zero-finding routine is required. A method to initially indicate the choice of poles is needed. This will guarantee the shift to be in the right direction.

The Devleeschouwer and Grenez algorithm uses two successive changes of variables to convert the design problem into a common optimization problem. This is done by the efficient Remez algorithm. This algorithm is applied simultaneously to the transmission zeros and attenuation zeros. It can be used to design a large class of filters. The limitation is that the attenuation function must be bounded by zero in the passband. The algorithm is very sensitive to the initial values. Transmission zeros or attenuation zeros can be fixed at a prescribed frequency or at $s = 0$, $s = \infty$. Also, the initial pole positions do not guarantee convergence.

For further improvement, a way should be developed for locating the initial values for the poles that would guarantee the convergence of the poles to an optimum position.

Appendix A

Program Listing of Cohen Algorithm

The computer program that describes the Cohen algorithm is included. Note that this program (and the one in Appendix B) is written in the PASCAL computer language running in double precision arithmetic on the Macintosh II computer using a Think Pascal compiler.

The input file to this program is described as follows:

```
# of level in the left stopband
normalized frequency (in kHz)      characteristic attenuation (in dB)
...
# of level in the passband
normalized frequency (in kHz)      characteristic attenuation (in dB)
...
# of level in the right stopband
normalized frequency (in kHz)      characteristic attenuation (in dB)
...
number of zeros (not degree of zeros!)
initial abscissas (in kHz) in passband
...
initial abscissas (in kHz) in passband
number of poles
initial poles
...
```

```

program Bandpass;
uses
  Types, Complex, Arithmetic, Polynomial, Plotting;
const
  MaxNoOfPole = 10;
  tolerance8 = 1.0e-8;
  tolerance10 = 1.0e-10;
  tolerance18 = 1.0e-18;
  Iteration_Limit = 10;
  Search_Limit = 5;
  Interpolation_Limit = 20;
  ScaleFactor = 20;
  PoleInSingle = TRUE;
  PoleInDouble = FALSE;
type
  SquareMatrix = array[1..MaxNoOfPole, 1..MaxNoOfPole] of extended;
  Distance = record
    error: boolean;
    least: extended;
  end;
var
  DataFile: text;
  Pass, Left_Stop, Right_Stop, Abscissa: complexPairArray;
  Xminus, Xplus, X_s, Xs: complex;
  Transfer_Function, Optimal_Function: rational;
  Least_Distance, Optimal_Least_Distance: extended;
  No_Shift, rational_Found: boolean;
  r, c, TwoPi, ln10, EndPoint: extended;
  Iteration, Optimal_index, Optimal_Iteration: integer;

  {Selects data file produced by "apple Edit" }
procedure SelectDataFile (var DataFile: text; prompt: Str255);
  var
    windowRect: Rect;
    StringName: Str255;
begin
  HideAll;
with windowRect do begin
    left := ScreenBits.bounds.right div 2 - 150;
    right := left + 300;
    bottom := ScreenBits.bounds.bottom - 10;
    top := bottom - 28;
  end;
  SetTextRect(windowRect);
  ShowText;
  write(Prompt);
  SysBeep(2);
  SysBeep(2);
  StringName := OldFileName('select data file from disk');
  writeln(' ', StringName);
  reset(DataFile, StringName);
end;      {procedure SelectDataFile}

```

```

procedure Read_Signal (var DataFile: text;
                      var Left_Stop, Pass, Right_Stop: complexPairArray);
  procedure Input_Signal (var InFile: text; var signal: complexPairArray);
    var
      i: integer;
    begin
      readln(InFile, signal.size);
      for i := 0 to signal.size do begin
        readln(InFile, signal.cPair[i].x.Re, signal.cPair[i].y.Re);
        signal.cPair[i].x.Im := 0;
        signal.cPair[i].y.Im := 0;
      end
    end;
  begin
    Input_Signal(DataFile, Left_Stop);
    Input_Signal(DataFile, Pass);
    Input_Signal(DataFile, Right_Stop);
  end;

procedure Print_Signal (var Left, Pass, Right: complexPairArray; msg1, msg2: Str255);
  procedure Output_Signal (var signal: complexPairArray; msge: Str255);
    var
      i: integer;
    begin
      writeln(msge, ':', ' ' : 5);
      for i := 0 to signal.size do
        writeln(signal.cPair[i].x.Re : 29 : 20, signal.cPair[i].y.Re : 35 : 20)
      end;
    begin
      writeln;
      writeln('Specifications:', msg1 : 8, msg2 : 34);
      Output_Signal(Left, 'Left Stopband');
      Output_Signal(Pass, 'Passband');
      Output_Signal(Right, 'Right Stopband');
    end;

procedure Input_Initial (var DataFile: text; var Abscissa: complexPairArray;
                       var Poly: polynomial);
  var
    i, last: integer;
    zero, pole: extended;
  begin
    with Abscissa do begin
      readln(DataFile, size);
      size := size * 2;
      writeln;
      writeln('Initial abscissa : (# of zero = ', size div 2 : 1, ')');
      writeln('frequency' : 27, 'omega square' : 30);
      cPair[0].x := Pass.cPair[0].x;
      writeln(cPair[0].x.Re : 64 : 20);
      last := size - 1;
    end;
  end;

```

```

for i := 1 to last do begin
  readln(DataFile, zero);
  cPair[i].x.Re := sqr(TwoPi * zero);
  cPair[i].x.Im := 0;
  writeln(zero : 28 : 10, cPair[i].x.Re : 36 : 20);
end;
cPair[size].x := Pass.cPair[Pass.size].x;
writeln(cPair[size].x.Re : 64 : 20);

readln(DataFile, Poly.deg);
Poly.rep := prodRep;
Poly.coef[0] := C_ONE;
writeln('Initial poles : (# of pole = ', Poly.deg : 1, ')');
for i := 1 to Poly.deg do begin
  readln(DataFile, pole);
  Poly.coef[i].Re := sqr(TwoPi * pole);
  Poly.coef[i].Im := 0;
  writeln(pole : 28 : 10, Poly.coef[i].Re : 36 : 20);
end
end
end;

procedure Convert (var signal: complexPairArray);
  var
    i: integer;
begin
with signal do
  for i := 0 to size do begin
    cPair[i].x.Re := sqr(TwoPi * cPair[i].x.Re);
    if (cPair[i].y.Re >= 20) then
      cPair[i].y.Re := exp(cPair[i].y.Re * ln10 / 10)
    else
      cPair[i].y.Re := exp(cPair[i].y.Re * ln10 / 10) - 1;
    end
  end;

procedure Time_Spent (var StartSec, EndSec: longint; msg: Str255);
  var
    TotalSec: longint;
    DateRec: DateTimeRec;
begin
  TotalSec := EndSec - StartSec;
  Secs2Date(TotalSec, DateRec);
with DateRec do
    writeln('Total time for ', msg, ' = ', hour : 2, ':', minute : 2, ':', second : 2);
end;

function Rational_Evaluation (var F: rational; X: complex): complex;
  var
    zero, pole: complex;
begin
  zero := polyEval(X, F.num);

```

```

pole := polyEval(X, F.den);
Rational_Evaluation := CDIV(zero, CMUL(pole, pole))
end;

procedure Plot_Signal (var F: rational; Start, Stop: extended; OutFileName: Str255);
  var
    k: integer;
    plot: PlotPtr;
    outFile: text;
    h, Xi: extended;
    X, Y: complex;
  begin
    New(plot);
    rewrite(outFile, OutFileName);
    writeln(outFile, '*');
    writeln(outFile, 'Freq', chr(9), 'Attn(dB)');
    plot^.ArraySize := MaxPts;
    h := (Stop - Start) / plot^.ArraySize;
    X := CZERO;
    Xi := Start - h;
    for k := 0 to plot^.ArraySize do begin
      Xi := Xi + h;
      plot^.xpts[k] := Xi;
      X.Re := sqrt(TwoPi * Xi);
      Y := Rational_Evaluation(F, X);
      plot^.ypts[k] := 10.0 * ln(Y.Re + 1) / ln10;
      writeln(outFile, plot^.xpts[k] : 10 : 6, chr(9), plot^.ypts[k] : 25 : 10);
    end;
    plot^.xmin := Start;
    plot^.xmax := Stop;
    plot^.ymax := +INF;
    plot^.ymin := -INF;
    plot^.discrete := FALSE;
    plot^.grid := TRUE;
    plot^.xdiv := 10;
    plot^.ydiv := 10;
    PlotVec(plot^);
    dispose(plot);
    close(outFile);
  end;

procedure Print_Rational (var F: rational; SinglePole: boolean; msg: Str255);
  var
    i, j: integer;
  begin
    writeln;
    writeln('BandPass Characteristic Function (hh/ff*) ', msg);
    write(' ' : 14, 'Zeroes (degree', F.num.deg : 3, ')', ' ' : 22, 'Poles (degree)');
    if SinglePole then
      writeln(F.den.deg : 3, ')')
    else
      writeln(F.den.deg * 2 : 3, ')');
  end;

```

```

for i := 0 to F.num.deg do
  if i = 0 then
    writeln(' ' : 7, F.num.coef[0].Re : 34, F.den.coef[0].Re : 34 : 25)
  else begin
    if SinglePole then
      j := i
    else
      j := i div 2 + i mod 2;
    if j <= F.den.deg then
      writeln(F.num.coef[j].Re : 35 : 25, F.den.coef[j].Re : 40 : 25)
    else
      writeln(F.num.coef[j].Re : 35 : 25);
    end;
  end;

procedure Print_Error (msg: Str255; X: extended; code: integer);
begin
  writeln;
  write('ERROR : ');
  if (code = 2) then
    write(X : 1 : 25);
  writeln(msg);
  if (code = 1) then
    writeln(' ' : 10, 'Try another initial values or increase the order of poles/zeros. ')
  else
    Halt;
  end;

function Cmin (x, y: complex): complex;
begin
  if CABS(x) < CABS(y) then
    Cmin := x
  else
    Cmin := y
  end;
function Cmax (x, y: complex): complex;
begin
  if CABS(x) > CABS(y) then
    Cmax := x
  else
    Cmax := y
  end;

function Ceiling (X: complex): complex;
  var
    Ceil: complex;
    m, i: integer;
begin
  m := Pass.size;
  if abs(X.Re - Xminus.Re) < tolerance8 then
    Ceil := Pass.cPair[1].y
  else if abs(X.Re - Xplus.Re) < tolerance8 then

```

```

    Ceil := Pass.cPair[m].y
else if (X.Re < Xminus.Re) or (X.Re > Xplus.Re) then
    Print_Error(' is out of the passband', X.Re, 2)
else begin
    i := 1;
    Ceil := C_ONE_;
    repeat
        if (abs(X.Re - Pass.cPair[i].x.Re) < tolerance8) and (i < m) then
            Ceil := Cmin(Pass.cPair[i].y, Pass.cPair[i + 1].y)
        else if X.Re < Pass.cPair[i].x.Re then
            Ceil := Pass.cPair[i].y;
        i := i + 1
    until (Ceil.Re >= 0) or (i > m)
    end;
    Ceiling := Ceil
end;

function Floor (X: complex): complex;
    var
        Xsr_1: complex;

    function SignalBetween (var Signal: complexPairArray; X: complex): complex;
        var
            i: integer;
            Flr: complex;
        begin
            i := 1;
            Flr := C_ONE_;
            repeat
                if (abs(X.Re - Signal.cPair[i].x.Re) < tolerance8) and (i < Signal.size) then
                    Flr := Cmax(Signal.cPair[i].y, Signal.cPair[i + 1].y)
                else if X.Re < Signal.cPair[i].x.Re then
                    Flr := Signal.cPair[i].y;
                i := i + 1
            until (Flr.Re >= 0) or (i > Signal.size);
            SignalBetween := Flr;
        end;

    begin (* Floor *)
        Xsr_1 := Right_Stop.cPair[Right_Stop.size - 1].x;
        if (X.Re < 0) or (X.Re > X_s.Re) and (X.Re < Xs.Re) then
            Print_Error(' is out of the stopbands', X.Re, 2)
        else if (X.Re <= X_s.Re) then
            if CABS(X) < tolerance8 then
                Floor := Left_Stop.cPair[1].y
            else if abs(X.Re - X_s.Re) < tolerance8 then
                Floor := Left_Stop.cPair[Left_Stop.size].y
            else
                Floor := SignalBetween(Left_Stop, X)
        else if abs(X.Re - Xs.Re) < tolerance8 then
            Floor := Right_Stop.cPair[1].y
        else if X.Re > Xsr_1.Re then

```

```

    Floor := Right_Stop.cPair[Right_Stop.size].y
  else
    Floor := SignalBetween(Right_Stop, X);
  end;

function f_minus_C (var F: rational; X: complex): complex;
  var
    Fx: complex;
  begin
    Fx := Rational_Evaluation(F, X);
    f_minus_C := CSUB(Fx, Ceiling(X));
  end;

function Max_Golden (var F: rational; AX, CX: complex;
                    function Max_Func (var U: rational; X: complex): complex);
  var
    f1, f2: complex;
    BX, X0, X1, X2, X3: complex;
  begin
    BX.Re := AX.Re + (CX.Re - AX.Re) / 3.0;
    BX.Im := 0.0;
    X0 := AX;
    X3 := CX;
    X1 := BX;
    X2 := CZERO;
    X2.Re := BX.Re + c * (CX.Re - BX.Re);
    f1 := Max_Func(F, X1);
    f2 := Max_Func(F, X2);
    while (ABS(X3.Re - X0.Re) > tolerance8 * (ABS(X1.Re) + ABS(X2.Re))) do begin
      if f2.Re > f1.Re then begin
        X0 := X1;
        X1 := X2;
        X2.Re := r * X1.Re + c * X3.Re;
        f1 := f2;
        f2 := Max_Func(F, X2)
      end
      else begin
        X3 := X2;
        X2 := X1;
        X1.Re := r * X2.Re + c * X0.Re;
        f2 := f1;
        f1 := Max_Func(F, X1)
      end
    end;
    if (f1.Re > f2.Re) then
      Max_Golden := X1
    else
      Max_Golden := X2
    end;
  end;

function g_minus_F (var F: rational; X: complex): complex;
  var

```



```

    temp: complex;
begin
temp := Rational_Evaluation(F, X);
g_minus_F := CSUB(temp, Floor(X));
end;

function Min_Golden (var F: rational; AX, CX: complex;
                    function Min_Func (var U: rational; X: complex): complex): complex;
    var
        f0, f1, f2, f3: complex;
        BX, X0, X1, X2, X3: complex;
begin
BX.Re := AX.Re + (CX.Re - AX.Re) / 3.0;
BX.Im := 0.0;
X0 := AX;
X3 := CX;
X1 := BX;
X2 := CZERO;
X2.Re := BX.Re + c * (CX.Re - BX.Re);
f1 := Min_Func(F, X1);
f2 := Min_Func(F, X2);
while (ABS(X3.Re - X0.Re) > tolerance8 * (ABS(X1.Re) + ABS(X2.Re)))
    and (abs(X3.Re - X0.Re) > tolerance10) do begin
    if f2.Re < f1.Re then begin
        X0 := X1;
        X1 := X2;
        X2.Re := r * X1.Re + c * X3.Re;
        f1 := f2;
        f2 := Min_Func(F, X2);
    end
    else begin
        X3 := X2;
        X2 := X1;
        X1.Re := r * X2.Re + c * X0.Re;
        f2 := f1;
        f1 := Min_Func(F, X1);
    end
    end;
f0 := Min_Func(F, X0);
f3 := Min_Func(F, X3);
if (f1.Re < f2.Re) then
    if (f0.Re < f1.Re) then
        Min_Golden := X0
    else
        Min_Golden := X1
    else if (f2.Re < f3.Re) then
        Min_Golden := X2
    else
        Min_Golden := X3
end;

procedure Evaluate_Abscissa (var F: rational; var Abscissa: complexPairArray);

```

```

    var
      pole: complex;
      i: integer;
    begin
    for i := 0 to Abscissa.size do
      with Abscissa.cPair[i] do
        if i mod 2 = 0 then begin
          y := Ceiling(x);
          pole := polyEval(x, F.den);
          y := CMUL(y, CMUL(pole, pole));
        end
        else
          y := CZERO;
        end
      end
    end;

    procedure Newton_Interpolation (var U: Polynomial; var Data: complexPairArray);
      var
        M, New_M: Polynomial;
        Alpha, Sigma: complex;
        k: integer;
      begin
        U.rep := prodRep;
        M.rep := prodRep;
        U.deg := 0;
        M.deg := 0;
        U.coef[0] := Data.cPair[0].y;
        M.coef[0] := C_ONE;
        for k := 1 to Data.size do begin
          M.coef[k] := Data.cPair[k - 1].x;
          M.deg := M.deg + 1;
          Alpha := polyEval(Data.cPair[k].x, M);
          Alpha := CDIV(C_ONE, Alpha);
          Sigma := polyEval(Data.cPair[k].x, U);
          Sigma := CSUB(Data.cPair[k].y, Sigma);
          Sigma := CMUL(Alpha, Sigma);
          if CABS(Sigma) > tolerance18 then begin
            New_M := M;
            New_M.coef[0] := CMUL(Sigma, M.coef[0]);
            U := FindprodRepOfSum(U, New_M)
          end
        end
      end;
    end;    { procedure Newton_Interpolation }

    procedure Sort_Abscissa (var AR: complexPairArray);
      var
        M, N, k, i: integer;
        sort: boolean;
        temp: complexPair;
      begin
        N := AR.size - 1;
        M := AR.size;
        k := 0;

```

```

sort := FALSE;
while (k <= N) and not sort do begin
  k := k + 1;
  M := M - 1;
  sort := TRUE;
  for i := 0 to M do
    if CABS(AR.cPair[i].y) > CABS(AR.cPair[i + 1].y) then begin
      temp := AR.cPair[i + 1];
      AR.cPair[i + 1] := AR.cPair[i];
      AR.cPair[i] := temp;
      sort := FALSE
    end
  end
end;

procedure Sort_Zeros (var U: polynomial);
  var
    M, N, k, i: integer;
    sort: boolean;
    temp: complex;
  begin
    N := U.deg - 1;
    M := U.deg;
    k := 0;
    sort := FALSE;
    while (k <= N) and not sort do begin
      k := k + 1;
      M := M - 1;
      sort := TRUE;
      for i := 1 to M do
        if CABS(U.coef[i]) > CABS(U.coef[i + 1]) then begin
          temp := U.coef[i + 1];
          U.coef[i + 1] := U.coef[i];
          U.coef[i] := temp;
          sort := FALSE
        end
      end
    end;

procedure Locate_abscissa_zeros (var F: rational;
                                  var NewX: complexPairArray; Asize: integer);
  var
    i, last: integer;
  begin
    NewX.size := Asize;
    last := Asize - 1;
    NewX.cPair[0].x := Xminus;
    for i := 1 to last do
      if (i mod 2 = 1) then
        NewX.cPair[i].x := Min_Golden(F, F.num.coef[i], F.num.coef[i + 1], Rational_Evaluation)
      else
        NewX.cPair[i].x := Max_Golden(F, F.num.coef[i], F.num.coef[i + 1], f_minus_C);

```

```

NewX.cPair[j].x := Xplus
end;

procedure Interpolation_Stage (var F: rational; var Abscissa: complexPairArray);
  var
    Abscissa_InOrder, NewX: complexPairArray;
    i, j: integer;
    success: boolean;
    StartSec, EndSec: longint;
begin
  j := 0;
  success := FALSE;
  GetDateTime(StartSec);
repeat
  j := j + 1;
  Evaluate_Abscissa(F, Abscissa);
  Abscissa_InOrder := Abscissa;
  Sort_Abscissa(Abscissa_InOrder);
  Newton_Interpolation(F.num, Abscissa_InOrder);
  Sort_Zeros(F.num);
  Locate_abscissa_zeros(F, NewX, Abscissa.size);
  i := 0;
  while (i <= Abscissa.size)
    and CEqualityTol(NewX.cPair[i].x, Abscissa.cPair[i].x, tolerance8) do
    i := i + 1;
  if i > Abscissa.size then
    success := TRUE;
  for i := 0 to Abscissa.size do
    Abscissa.cPair[i].x := NewX.cPair[i].x;
until success or (j > Interpolation_Limit);
  GetDateTime(EndSec);
if not success then
  Print_Error('Can't interpolate the zeros !', 0, 1)
else begin
  writeln('# of interpolation =', j : 3);
  Time_Spent(StartSec, EndSec, 'Interpolation');
  end;
end;

function Check_End (EndPoint: complex): complex;
begin
if EndPoint.Re > X_s.Re then
  Check_End := X_s
else
  Check_End := EndPoint
end;

function Check_Start (StartPoint: complex): complex;
begin
if StartPoint.Re < Xs.Re then
  Check_Start := Xs
else

```

```

    Check_Start := StartPoint
end;

procedure Locate_abscissa_poles (var G: rational; var NewX: complexPairArray);
  var
    i: integer;
    StartPoint, EndPoint: complex;
begin
  NewX.size := G.den.deg;
  EndPoint := Check_End(G.den.coef[1]);
  NewX.cPair[1].x := Min_Golden(G, CZERO, EndPoint, g_minus_F);
  i := 2;
while (G.den.coef[i].Re < Pass.cPair[0].x.Re) do begin
  EndPoint := Check_End(G.den.coef[i]);
  NewX.cPair[i].x := Min_Golden(G, G.den.coef[i - 1], EndPoint, g_minus_F);
  i := i + 1
end;
while (i < G.den.deg) do begin
  StartPoint := Check_Start(G.den.coef[i]);
  NewX.cPair[i].x := Min_Golden(G, StartPoint, G.den.coef[i + 1], g_minus_F);
  i := i + 1
end;
  StartPoint := Check_Start(G.den.coef[i]);
  EndPoint := Right_Stop.cPair[Right_Stop.size].x;
  EndPoint.Re := EndPoint.Re * 2;           { increase searching length }
  NewX.cPair[i].x := Min_Golden(G, StartPoint, EndPoint, g_minus_F);

for i := 1 to NewX.size do
  NewX.cPair[i].y := Rational_Evaluation(G, NewX.cPair[i].x);
end;

procedure Matrix_Evaluate (var A: SquareMatrix; size: integer);
  var
    k, row, col, last: integer;
    w: extended;

  procedure Exchange_Row (var A: SquareMatrix; size, k: integer);
    var
      i, j: integer;
      temp: extended;
    begin
      i := k + 1;
      while (i <= size) and (abs(A[i, k]) < tolerance18) do
        i := i + 1;
      if (i <= size) then
        for j := 1 to size + 1 do begin
          temp := A[k, j];
          A[k, j] := A[i, j];
          A[i, j] := temp;
        end
      else
        Print_Error('Matrix cannot be evaluated !', 0, 3);

```

```

end;

begin (* Matrix_Evaluate *)
last := size + 1;
for k := 1 to size - 1 do begin
  if abs(A[k, k]) < tolerance8 then
    Exchange_Row(A, size, k);
  for row := k + 1 to size do
    if abs(A[row, k]) > tolerance18 then begin
      w := A[k, k] / A[row, k];
      for col := k + 1 to last do
        A[row, col] := A[k, col] - w * A[row, col];
      A[row, k] := 0;
    end;
  end;

for k := size downto 2 do begin
  A[k, last] := A[k, last] / A[k, k];
  A[k, k] := 1.0;
  for row := k - 1 downto 1 do begin
    for col := k - 1 downto row do
      A[row, col] := A[row, col] - A[k, col] * A[row, k];
    A[row, last] := A[row, last] - A[k, last] * A[row, k];
    A[row, k] := 0;
  end;
end;
A[1, last] := A[1, last] / A[1, 1];
end;

procedure Check_Stop (var G: rational; var Xi: complexPairArray;
                      var BelowFloor: Distance);

  var
    i: integer;
    FlrX: complex;
begin
  if not No_Shift then
    Locate_abscissa_poles(G, Xi);
  BelowFloor.least := 0;
  for i := 1 to Xi.size do begin
    FlrX := Floor(Xi.cPair[i].x);
    if (Xi.cPair[i].y.Re < FlrX.Re) then
      BelowFloor.least := BelowFloor.least + sqr(Xi.cPair[i].y.Re - FlrX.Re);
    end;
  if (BelowFloor.least < tolerance8) then
    BelowFloor.error := FALSE
  else begin
    BelowFloor.least := sqrt(BelowFloor.least);
    BelowFloor.error := TRUE;
  end;
end;

procedure Check_Stop_Edge (var G: rational; var BelowEdge: Distance);

```

```

var
  Left_Edge, Right_Edge: Distance;

procedure Check_Edge (var G: rational; X: complex; var Edge: Distance);
  var
    Gx, Flrx: complex;
  begin
    Gx := Rational_Evaluation(G, X);
    Flrx := Floor(X);
    Edge.least := sqr(Gx.Re - Flrx.Re);
    if (Gx.Re > Flrx.Re) then
      Edge.error := FALSE
    else
      Edge.error := TRUE;
    end;

begin
  Check_Edge(G, X_s, Left_Edge);
  Check_Edge(G, Xs, Right_Edge);
  BelowEdge.error := Left_Edge.error or Right_Edge.error;
  if not BelowEdge.error or Left_Edge.error and Right_Edge.error then
    BelowEdge.least := sqrt(Left_Edge.least + Right_Edge.least)
  else if Left_Edge.error then
    BelowEdge.least := sqrt(Left_Edge.least)
  else
    BelowEdge.least := sqrt(Right_Edge.least)
  end;

procedure Check_Pass (var G: rational; var OverPass: Distance);
  var
    i: integer;
    Over: Distance;
    Xi: complex;

  procedure Check_Over (var G: rational; X: complex; var Over: Distance);
    var
      Gx, Cx: complex;
    begin
      Gx := Rational_Evaluation(G, X);
      Cx := Ceiling(X);
      Over.least := 0;
      if (Gx.Re > Cx.Re) then begin
        Over.error := TRUE;
        Over.least := sqr(Gx.Re - Cx.Re);
      end
    else
      Over.error := FALSE;
    end;

  begin
    Check_Over(G, Xminus, Over);
    OverPass := Over;

```

```

i := 2;
while (i < G.num.deg) do begin
  Xi := Max_Golden(G, G.num.coef[i], G.num.coef[i + 1], Rational_Evaluation);
  Check_Over(G, Xi, Over);
  OverPass.error := OverPass.error or Over.error;
  OverPass.least := OverPass.least + Over.least;
  i := i + 2;
end;
Check_Over(G, Xplus, Over);
OverPass.error := OverPass.error or Over.error;
OverPass.least := sqrt(OverPass.least + Over.least);
end;

procedure Find_DeltaP (var G: rational; var Xi: complexPairArray;
                      var M: SquareMatrix);

  var
    last, i, j: integer;
    temp: complex;
begin
  last := G.den.deg + 1;
for i := 1 to Xi.size do begin
  temp := Rational_Evaluation(G, Xi.cPair[i].x);
  for j := 1 to G.den.deg do
    M[i, j] := 2 * temp.Re / (Xi.cPair[i].x.Re - G.den.coef[j].Re);
  temp := CSUB(Floor(Xi.cPair[i].x), temp);
  M[j, last] := temp.Re;
  end;
  Matrix_Evaluate(M, G.den.deg);
end;

function Check_DeltaP (var M: SquareMatrix; size: integer): boolean;
  var
    last, i: integer;
begin
  i := 1;
  last := size + 1;
while (abs(M[i, last]) < tolerance8) and (i <= size) do
  i := i + 1;
if (i > size) then
  Check_DeltaP := TRUE
else
  Check_DeltaP := FALSE;
end;

procedure Search_BestP (var G: rational; var Xi: complexPairArray;
                      var M: SquareMatrix; var UnderFloor, UnderEdge, AbovePass: Distance);
  var
    NewG: rational;
    K, scale: extended;
    Best_Factor, i, j, last: integer;
    In_Pass: boolean;
    BelowFloor, BelowEdge, OverPass: Distance;

```



```

    Least_Distance, Total_Distance: extended;
begin
Least_Distance := INF;
scale := 1 / ScaleFactor;
j := -1;
last := G.den.deg + 1;
NewG.num := G.num;
with NewG.den do
    while (j < ScaleFactor) and not rational_Found do begin
        j := j + 1;
        K := j * scale;
        NewG.den := G.den;
        i := 1;
        In_Pass := FALSE;
        (* check new pole whether in pass band *)
        while (j > 0) and (i <= deg) and not In_Pass do begin
            coef[i].Re := coef[i].Re + M[i, last] * K;
            if (0 > coef[i].Re) or (coef[i].Re > Xminus.Re) and (coef[i].Re < Xplus.Re) then
                In_Pass := TRUE;
            i := i + 1;
        end;

        if not In_Pass then begin
            Check_Stop(NewG, Xi, BelowFloor);
            Check_Stop_Edge(NewG, BelowEdge);
            if j > 0 then
                Check_Pass(NewG, OverPass)
            else begin
                OverPass.error := FALSE;
                OverPass.least := 0;
            end;
            if BelowEdge.error then
                Total_Distance := BelowFloor.least + BelowEdge.least + OverPass.least
            else
                Total_Distance := BelowFloor.least + OverPass.least;
            if (Total_Distance < Least_Distance) then begin
                Best_Factor := j;
                Least_Distance := Total_Distance;
                UnderFloor := BelowFloor;
                UnderEdge := BelowEdge;
                AbovePass := OverPass;
            end;
            if not (BelowEdge.error or BelowFloor.error or OverPass.error) then
                rational_Found := TRUE;
            end;
        end;

    end;

if (Best_Factor = 0) then begin
    No_Shift := TRUE;
    writeln('Pole is best not to be shifted !')
end
else begin

```

```

K := Best_Factor * scale;
writeln('New delta p : with scaling factor = ', K : 1 : 3);
for i := 1 to G.den.deg do begin
  M[i, last] := M[i, last] * K;
  writeln(M[i, last] : 30 : 20);
  G.den.coef[i].Re := G.den.coef[i].Re + M[i, last];
end;
end;
end;

procedure Pole_Shifting_Stage (var G: rational; var Least_Edge_Distance: extended);
var
  Xi: complexPairArray;
  M: SquareMatrix;
  Under_Floor, Under_Edge, Above_Pass: Distance;
  StartSec, EndSec: longint;
begin
  writeln;
  GetDateTime(StartSec);
  Locate_abscissa_poles(G, Xi);
  Find_DeltaP(G, Xi, M);
  No_Shift := Check_DeltaP(M, G.den.deg);

  rational_Found := FALSE;
  if No_Shift then begin
    Check_Stop(G, Xi, Under_Floor);
    Check_Stop_Edge(G, Under_Edge);
    if not (Under_Floor.error or Under_Edge.error) then
      rational_Found := TRUE;
    Above_Pass.error := FALSE;
    writeln('No pole is shifted !')
    end
  else
    Search_BestP(G, Xi, M, Under_Floor, Under_Edge, Above_Pass);

  writeln;
  Least_Edge_Distance := Under_Edge.least;
  if Under_Edge.error then
    writeln('least distance Under Stop edges = ', Under_Edge.least : 27);
  if Under_Floor.error then
    writeln('least distance Under Stop Band = ', Under_Floor.least : 27);
  if Above_Pass.error then
    writeln('least distance Above Pass Band = ', Above_Pass.least : 27);
  GetDateTime(EndSec);
  Time_Spent(StartSec, EndSec, 'Pole-Shifting');
end;

procedure Output_Optimal (var G: rational);
var
  i, j, decimal: integer;
  number, remainder, digit: extended;
  Fw, Ff: rational;

```

```

begin
i := 1;
with G.num do
  while (i <= deg) do begin
    decimal := abs(trunc(ln(coef[i + 1].Re - coef[i].Re) / ln10 + 0.1));
    number := trunc(coef[i + 1].Re);
    remainder := coef[i + 1].Re - number;
    for j := 1 to decimal do begin
      remainder := remainder * 10;
      digit := trunc(remainder);
      remainder := remainder - digit;
      digit := digit * exp(-j * ln10);
      number := number + digit;
    end;
    coef[i].Re := number;
    coef[i + 1].Re := coef[i].Re;
    i := i + 2;
  end;

  for i := 0 to G.num.deg do begin
    Fw.num.coef[i] := G.num.coef[i * 2];
    Ff.num.coef[i] := Fw.num.coef[i];
    if (i > 0) then
      Ff.num.coef[i].Re := sqrt(Fw.num.coef[i].Re) / TwoPi;
    end;
    Fw.den := G.den;
    Ff.den := Fw.den;
    for i := 1 to Ff.den.deg do
      Ff.den.coef[i].Re := sqrt(Ff.den.coef[i].Re) / TwoPi;
    Fw.num.deg := G.num.deg div 2;
    Ff.num.deg := Fw.num.deg;
    Print_Rational(Fw, PoleInSingle, 'in omega square');
    Print_Rational(Ff, PoleInSingle, 'in frequency');
  end;

  (***** M A I N P R O G R A M *****)
begin
DateTime;
c := (3.0 - sqrt(5.0)) / 2.0;
r := 1.0 - c;
TwoPi := 2 * PI;
ln10 := ln(10);
SelectDataFile(DataFile, 'Select data file :');
OpenTextWindow(50, 50, ScreenBits.bounds.right - 50, ScreenBits.bounds.bottom);

Read_Signal(DataFile, Left_Stop, Pass, Right_Stop);
Print_Signal(Left_Stop, Pass, Right_Stop, 'fi', 'lff*/gg*I in dB');
EndPoint := Right_Stop.cPair[Right_Stop.size].x.Re;

Convert(Pass);
Convert(Right_Stop);
Convert(Left_Stop);

```

```

Print_Signal(Left_Stop, Pass, Right_Stop, 'Xi', 'f(Xi)');
Input_Initial(DataFile, Abscissa, Transfer_Function.den);
close(DataFile);

Xminus := Pass.cPair[0].x;
Xplus := Pass.cPair[Pass.size].x;
X_s := Left_Stop.cPair[Left_Stop.size].x;
Xs := Right_Stop.cPair[0].x;

Optimal_Least_Distance := -INF;
Optimal_Iteration := -1;
Iteration := 0;
Optimal_index := 0;

repeat
  Iteration := Iteration + 1;
  writeln;
  writeln('----- I t e r a t i o n -----', Iteration : 1);
  Interpolation_Stage(Transfer_Function, Abscissa);
  Pole_Shifting_Stage(Transfer_Function, Least_Distance);
  Print_Rational(Transfer_Function, PoleInDouble, 'in omega square');

  if rational_Found then
    if (Least_Distance > Optimal_Least_Distance) then begin
      Optimal_Least_Distance := Least_Distance;
      Optimal_Function := Transfer_Function;
      Optimal_Iteration := Iteration;
      Optimal_index := 0;
    end
    else if (Optimal_index >= Search_Limit) then
      No_Shift := TRUE
    else begin
      writeln('**** Transfer function is not Optimal');
      Optimal_index := Optimal_index + 1;
    end
  until No_Shift or (Optimal_Iteration = -1) and (Iteration >= Iteration_Limit);

if (Optimal_Iteration = -1) then
  Print_Error('No Characteristic Function is found.', 0, 1)
else begin
  writeln;
  writeln('----- O p t i m a l   T r a n s f e r   F u n c t i o n -----');
  writeln;
  writeln('Optimal function is found in ', Optimal_Iteration : 1, 'th iteration. ');
  Output_Optimal(Optimal_Function);
  Plot_Signal(Optimal_Function, 0, EndPoint, 'ff*/gg*');
  writeln;
  writeln('----- E n d   o f   P r o c e s s i n g -----');
  end;
end.

```

The following program listings are the 'units' which are required by the previous program; they were developed by Prof. G. O. Martens.

```

unit Types;
interface
  const
    PI = 3.141592653589793238462643383279;
    MAXDEG = 30;
  type
    {*****}
    {complex Types follow}

    complex = record
      Re, Im: Extended
    end;
    complexPair = record
      x: complex;
      y: complex
    end;
    complexPairArrayType = array[0..101] of complexPair;
    complexPairArray = record
      size: integer;
      cPair: complexPairArrayType;
    end;
    ComplexArrayType = array[0..MAXDEG] of Complex;

    {*****}
    {polynomial Types follow}

    polyRep = (sumRep, prodRep);
    valueType = (fctpole, fctzero, fctkonst);
    polynomial = record
      rep: polyRep;
      deg: integer;
      coef: ComplexArrayType;
    end;

    {*****}
    {rational Types follow}

    rational = record
      num: Polynomial;
      den: Polynomial;
    end;

    {*****}

  procedure DateTime;
  procedure OpenTextWindow (left1, top1, right1, bottom1: integer);    {Opens Text Window}

```

Implementation

```
procedure DateTime;  
  var  
    secs: longint;  
    daterec: DateTimerec;  
begin  
  GetDateTime(secs);  
  Secs2Date(secs, daterec);  
with daterec do begin  
  writeln(year : 4, '/', month : 2, '/', day : 2);  
  writeln(hour : 2, ':', minute : 2, ':', second : 2);  
  end;  
end;  
  
procedure OpenTextWindow (left1, top1, right1, bottom1: integer);    {Opens Text Window}  
  var  
    windowRect: Rect;  
begin  
with windowRect do begin  
  left := left1;           {10}  
  top := top1;             {40}  
  right := right1;        {630}  
  bottom := bottom1;     {470}  
  end;  
  SetTextRect(windowRect);  
  ShowText;  
  end;  
  
end.
```

```

unit Complex;
interface
  uses
    Types;
  const
    EQUALITY_EPSILON = 1e-16;

  function CZERO: complex;
  function C_ONE: complex;
  function C_ONE_: complex;
  function Re_Im_to_C (Re, Im: extended): complex;
  function CONJ (A: complex): complex;
  function SCALE (K: extended; A: complex): complex;
  function CABS (A: complex): extended;
  function QUAD1 (A: complex): complex; { Reflects A into the 1st Quadrant}
  function QUAD (A: complex): integer; { Determines the Quadrant}
  function R_TO_P (A: complex): complex;
  function P_TO_R (A: complex): complex;
  function CADD (A, B: complex): complex;
  function CSUB (A, B: complex): complex;
  function CMUL (D, E: complex): complex;
  function CDIV (A, B: complex): complex;
  function CSQRT (A: complex): complex;
  function kthPowerOfC (k: integer; z: complex): complex;
  function complexNo (x, y: extended): complex;
  function CEqual (A, B: complex): Boolean; { A =B exactly}
  function CEquality (A, B: complex): boolean;
  function CNEGATE (A: complex): complex;
  function ZeroCResidue (A: complex): complex;
  function CEqualityTol (A, B: complex; tol: real): boolean; {tolerance can be specified}
  function RelativeZero (testVal, reference: complex; tol: real): boolean;
  function RelativeCEquality (testVal, reference: complex; tol: real): boolean;

```

Implementation

```

function CZERO: complex;
  var
    z: complex;
  begin
    z.Re := 0;
    z.Im := 0;
    CZERO := z;
  end;

function C_ONE: complex;
  var
    z: complex;
  begin
    z.Re := 1;
    z.Im := 0;
    C_ONE := z;
  end;

```

```
function C_ONE_: complex;
  var
    z: complex;
begin
  z.Re := -1;
  z.Im := 0;
  C_ONE_ := z;
end;

function Re_Im_to_C (Re, Im: extended): complex;
  var
    C: complex;
begin
  C.Re := Re;
  C.Im := Im;
  Re_Im_to_C := C;
end;

function CONJ (A: complex): complex;
  var
    B: complex;
begin
  B.Re := A.Re;
  B.Im := -A.Im;
  Conj := B;
end;

function SCALE (K: extended; A: complex): complex;
begin
  A.Re := K * A.Re;
  A.Im := K * A.Im;
  SCALE := A;
end;

function CABS (A: complex): extended;
begin
  if (A.Re = 0) and (A.Im = 0) then
    CABS := 0
  else begin
    A.Re := abs(A.Re);
    A.Im := abs(A.Im);
    if (A.Re = INF) or (A.Im = INF) then
      CABS := INF
    else begin
      if (A.Re >= A.Im) then
        CABS := A.Re * SQRT(1 + SQR(A.Im / A.Re))
      else
        CABS := A.Im * SQRT(1 + SQR(A.Re / A.Im));
      end;
    end;
  end;
end;
```



```
function QUAD1 (A: complex): complex; { Reflects A into the 1st Quadrant}
begin
  A.Re := ABS(A.Re);
  A.Im := ABS(A.Im);
  Quad1 := A;
end;
```

```
function QUAD (A: complex): integer; { Determines the Quadrant}
begin
  if A.Re >= 0 then
    if A.Im >= 0 then
      QUAD := 1
    else
      QUAD := 4
    else if A.Im >= 0 then
      QUAD := 2
    else
      QUAD := 3
  end;
```

```
function R_TO_P (A: complex): complex; {  $-\pi \leq \text{angle} \leq \pi$ }
  var
    B: complex;
    R: extended;
begin
  if (A.Re = 0) and (A.Im = 0) then
    R_TO_P := CZERO
  else begin
    B := QUAD1(A);
    if B.Re >= B.Im then
      R := arctan(B.Im / B.Re)
    else
      R := pi / 2 - arctan(B.Re / B.Im);
    case QUAD(A) of
      1:
        B.Im := R;
      2:
        B.Im := pi - R;
      3:
        B.Im := R - pi;
      4:
        B.Im := -R;
    end;
    B.Re := CABS(A);
    R_TO_P := B;
  end;
end;
```

```
function P_TO_R (A: complex): complex;
  var
    B: complex;
```

```
begin
  B.Re := A.Re * cos(A.Im);
  B.Im := A.Re * sin(A.Im);
  P_TO_R := B;
end;

function CADD (A, B: complex): complex;
  var
    C: complex;
begin
  C.Re := A.Re + B.Re;
  C.Im := A.Im + B.Im;
  CADD := C;
end;

function CSUB (A, B: complex): complex;
  var
    C: complex;
begin
  C.Re := A.Re - B.Re;
  C.Im := A.Im - B.Im;
  CSUB := C;
end;

function CMUL (D, E: complex): complex;
  var
    C, CZero: complex;
begin
  if CEqual(D, CZero) or CEqual(E, CZero) then
    C := CZero
  else begin
    C.Re := (D.Re * E.Re) - (D.Im * E.Im);
    C.Im := (D.Re * E.Im) + (D.Im * E.Re);
  end;
  CMUL := C;
end;

function CDIV (A, B: complex): complex;
  var
    C: complex;
    R: extended;
begin
  C := CONJ(B);
  R := CABS(B);
  A.Re := A.Re / R;
  A.Im := A.Im / R;
  C.Re := C.Re / R;
  C.Im := C.Im / R;
  CDIV := CMUL(A, C);
end;
```

```

function CSQRT (A: complex): complex;
  var
    C: complex;
begin
  C := R_TO_P(A);
  C.Re := sqrt(C.Re);
  C.Im := C.Im / 2;
  CSQRT := P_TO_R(C);
end;

function kthPowerOfC (k: integer; z: complex): complex;
  var
    t, v: complex;
    u: integer;
begin
  t := z;
  u := k;
  v.Re := 1;
  v.Im := 0;
while u <> 0 do begin
  if u mod 2 <> 0 then
    v := CMUL(t, v);
    t := CMUL(t, t);
    u := u div 2;
  end;
  kthPowerOfC := v;
end;

function complexNo (x, y: extended): complex;
begin
  complexNo.Re := x;
  complexNo.Im := y;
end;

function CEqual (A, B: complex): Boolean;
begin
  CEqual := (A.Re = B.Re) and (A.Im = B.Im)
end;

function CEquality (A, B: complex): boolean;
  var
    x, y1, y2: extended;
begin
  x := CABS(CSUB(A, B));
  if x = 0 then
    CEquality := TRUE
  else begin
    y1 := CABS(A);
    y2 := CABS(B);
    if (y1 > y2) then
      y2 := x / y1
    else

```

```
    y2 := x / y2;
  if y2 < EQUALITY_EPSILON then
    CEquality := TRUE
  else
    CEquality := FALSE;
  end;
end;

function CNEGATE (A: complex): complex;
  var
    C: complex;
  begin
    C.Re := -A.Re;
    C.Im := -A.Im;
    CNEGATE := C;
  end;

function ZeroCResidue (A: complex): complex;
  var
    C: complex;
  begin
    C := A;
    if CEqual(A, CZERO) then {changed to CEqual from CEquality 90/11/16}
      C := CZERO
    else begin
      if (abs(C.Re) / CABS(C)) < EQUALITY_EPSILON then
        C.Re := 0;
      if (abs(C.Im) / CABS(C)) < EQUALITY_EPSILON then
        C.Im := 0;
      end;
    ZeroCResidue := C;
  end;

function CEqualityTol (A, B: complex; tol: real): boolean;
  var
    x, y1, y2: extended;
  begin
    x := CABS(CSUB(A, B));
    if x = 0 then
      CEqualityTol := TRUE
    else begin
      y1 := CABS(A);
      y2 := CABS(B);
      if (y1 > y2) then
        y2 := x / y1
      else
        y2 := x / y2;
      if y2 < tol then
        CEqualityTol := TRUE
      else
        CEqualityTol := FALSE;
      end;
    end;
  end;
```

end;

function RelativeZero (testVal, reference: complex; tol: real): boolean;

begin

RelativeZero := TRUE;

if not CEqual(testVal, CZERO) **then**

if (CABS(CDIV(testVal, reference)) > tol) **then**

 RelativeZero := FALSE;

end;

function RelativeCEquality (testVal, reference: complex; tol: real): boolean;

begin

testVal := CSUB(testVal, reference);

if (CABS(CDIV(testVal, reference)) < tol) **then**

 RelativeCEquality := TRUE

else

 RelativeCEquality := FALSE;

end;

end.

```

unit Arithmetic;
interface
  uses
    Types, Complex;

  procedure SortCPairArray (var X: complexPairArrayType; N: integer);
    {sorts in ascending order of y-mag--from Pascal Programs}
  function MaxOfTwoIntegers (m, n: integer): integer;
  function MinOfTwoExtendeds (m, n: extended): extended;
  function MaxOfTwoExtendeds (m, n: extended): extended;

```

Implementation

```

procedure SortCPairArray (var X: complexPairArrayType; N: integer);
  {sorts in ascending order of y-mag--from Pascal Programs}

procedure QSortCPairArray (var X: complexPairArrayType; M, N: integer);
  var
    i, j: integer;

  procedure Partit (var A: complexPairArrayType; var i, j: integer;
    left, right: integer); {sorts in ascending order of y-mag--from Pascal Programs}
  var
    pivot: complexPair;

  procedure Swap (var p, q: complexPair);
  var
    hold: complexPair;
  begin
    hold := p;
    p := q;
    q := hold;
  end; {swap}

  begin
    pivot := A[(left + right) div 2];
    i := left;
    j := right;
  while i <= j do begin
    while CABS(A[i].y) < CABS(pivot.y) do
      i := i + 1;
    while CABS(pivot.y) < CABS(A[j].y) do
      j := j - 1;
    if i <= j then begin
      swap(A[i], A[j]);
      i := i + 1;
      j := j - 1;
    end
  end {while}
end; {partit}

```

```
begin {QSortCPairArray}
if M < N then begin
  partit(X, i, j, M, N); {divide in two}
  QSortCPairArray(X, M, j); {sort left part}
  QSortCPairArray(X, i, N); {sort right part}
end
end; {QSortCPairArray}
```

```
begin {SortCPairArray}
QSortCPairArray(X, 1, N);
end; {SortCPairArray}
```

```
function MaxOfTwoIntegers (m, n: integer): integer;
begin
if m >= n then
  MaxOfTwoIntegers := m
else
  MaxOfTwoIntegers := n;
end;
```

```
function MinOfTwoExtendeds (m, n: extended): extended;
begin
if m <= n then
  MinOfTwoExtendeds := m
else
  MinOfTwoExtendeds := n;
end;
```

```
function MaxOfTwoExtendeds (m, n: extended): extended;
begin
if m >= n then
  MaxOfTwoExtendeds := m
else
  MaxOfTwoExtendeds := n;
end;
```

end.

unit Polynomial;

interface

uses

Types, Complex, Arithmetic;

function ZeroPoly (representation: polyRep): polynomial;

function UnitPoly (representation: polyRep): polynomial;

function AddPoly (var p1, p2: polynomial): polynomial;

function MulPoly (var p1, p2: polynomial): polynomial;

function PolyEval (z: complex; var p: polynomial): complex;

function MaxPolyRadius (var p: polynomial): extended;
 {max radius= mag of largest zero mag ≤ 0 }

function MinPolyRadius (var p: polynomial): extended;
 {min radius= mag of smallest zero mag ≤ 0 }

function SortPolyZeros (var p: polynomial): polynomial;

function DFT_Coefs (k, N: integer; r: extended; var p: polynomial): complex;

procedure StoreZero (testZero: complex; var found_p: polynomial);

procedure FindCommonZeros (var p1, p2, common_p: polynomial);

function RemoveFactor_i (i: integer; var p: polynomial): polynomial;

function PicksHighestDegPoly (var p1, p2: polynomial): polynomial;

function PolyDelay (z: complex; var p: polynomial): complex; {f'/f evaluated at z}

function PolyDeriv (var p: polynomial): polynomial;

function PolyDelayPair (z: complex; var p1, p2: polynomial): complexPair;

function PolyDelayII (z: complex; var p: polynomial): complex; {f''/f evaluated at z}

function LaguerreStep (z: complex; p1, p2, q: polynomial): complex;
 {see Orchard CAS Nov 1989 p.1377}

function FindprodRepOfSum (var q1, q2: polynomial): polynomial; {q1, q2 are in prodRep}

implementation

function ZeroPoly (representation: polyRep): polynomial;

var

q: polynomial;

begin

q.deg := 0;

q.rep := representation;

q.coef[0] := CZERO;

ZeroPoly := q;

end;

function UnitPoly (representation: polyRep): polynomial;

var

q: polynomial;

begin

q.deg := 0;

q.rep := representation;

q.coef[0] := C_ONE;

UnitPoly := q;

end;


```

function PolyEqual (var p1, p2: polynomial): boolean;
  var
    i: integer;
begin
if (p1.rep = p2.rep) and (p1.deg = p2.deg) then begin
  for i := 0 to p1.deg do
    if not CEqual(p1.coef[i], p2.coef[i]) then
      PolyEqual := FALSE
    else
      PolyEqual := TRUE
    end
  end
else
  PolyEqual := FALSE;
end;

function AddPoly (var p1, p2: polynomial): polynomial;
  var
    i, minDeg: integer;
    q: polynomial;
begin
if (p1.deg = 0) and CEqual(p1.coef[0], CZERO) then
  q := p2
else if (p2.deg = 0) and CEqual(p2.coef[0], CZERO) then
  q := p1
else begin {nonspecial case}
  if (p1.rep <> p2.rep) then begin
    writeln('addition of prodRep with sumRep not available');
    ExitToShell;
  end;
  if (p1.rep = prodRep) and (p2.rep = prodRep) then begin
    q := FindprodRepOfSum(p1, p2);
  end
  else begin
    q.rep := sumRep;
    if p1.deg < p2.deg then begin
      minDeg := p1.deg;
      q := p2;
    end;
    if p2.deg < p1.deg then begin
      minDeg := p2.deg;
      q := p1;
    end;
    if p1.deg = p2.deg then begin {check for degree reduction}
      minDeg := p1.deg;
      while CEquality(p1.coef[minDeg], CNEGATE(p2.coef[minDeg])) and (minDeg > 0) do
        minDeg := minDeg - 1;
        q.deg := minDeg
      end;
    if (p1.deg = p2.deg) and CEquality(p1.coef[minDeg], CNEGATE(p2.coef[minDeg]))
      and (minDeg = 0) then
      q := ZeroPoly(p1.rep)
    else

```

```

        for i := 0 to minDeg do
            q.coef[i] := CADD(p1.coef[i], p2.coef[i]);
        end;
    end; {nonspecial case}
AddPoly := q;
end;

function MulPoly (var p1, p2: polynomial): polynomial;
    var
        i, j: integer;
        q: polynomial;
    begin
    if (p1.rep <> p2.rep) then begin
        writeln('multiplication of prodRep with sumRep not available');
        ExitToShell;
        end;
    if ((p1.deg = 0) and (CABS(p1.coef[0]) = 0)) or ((p2.deg = 0) and (CABS(p2.coef[0]) = 0))
        then
        q := ZeroPoly(p1.rep)
    else begin
        q.deg := p1.deg + p2.deg;
        q.rep := p1.rep;
        if (p1.rep = prodRep) and (p2.rep = prodRep) then begin
            q.coef[0] := CMUL(p1.coef[0], p2.coef[0]);
            for i := 1 to p1.deg do
                q.coef[i] := p1.coef[i];
            for i := p1.deg + 1 to q.deg do
                q.coef[i] := p2.coef[i - p1.deg];
            end; {end if prodRep}
            if (p1.rep = sumRep) and (p2.rep = sumRep) then begin
                for i := 0 to q.deg do begin
                    q.coef[i] := CZERO;
                    for j := 0 to p1.deg do
                        if ((i - j) <= p2.deg) and (0 <= (i - j)) then
                            q.coef[i] := CADD(q.coef[i], CMUL(p1.coef[j], p2.coef[i - j]));
                        end; {end i for-loop}
                    end; {end if sumRep}
                end;
            end;
        end;
        MulPoly := q;
    end;

function ProdPolyEvaluation (z: complex; var p: polynomial): complex;
    var
        i: integer;
        value: complex;
    begin
    value := p.coef[0];
    for i := 1 to p.deg do
        value := CMUL(value, CSUB(z, p.coef[i]));
    ProdPolyEvaluation := value;
    end;

```

```

function SumPolyEvaluation (z: complex; var p: polynomial): complex;
  var
    i: integer;
    value: complex;
  begin
    value := p.coef[p.deg];
    for i := p.deg - 1 downto 0 do
      value := CADD(CMUL(value, z), p.coef[i]);
    SumPolyEvaluation := value;
  end;

```

```

function PolyEval (z: complex; var p: polynomial): complex;
  var
    value: complex;
  begin
    if p.rep = sumRep then
      value := SumPolyEvaluation(z, p)
    else
      value := ProdPolyEvaluation(z, p);
    PolyEval := value;
  end;

```

```

function maxPolyRadius (var p: polynomial): extended;
  var
    q: polynomial;
    x: extended;
  begin
    q := SortPolyZeros(p);
    x := CABS(q.coef[q.deg]);
    if x <> 0 then
      maxPolyRadius := x
    else
      maxPolyRadius := CABS(q.coef[0]);
  end;

```

```

function minPolyRadius (var p: polynomial): extended;
  var
    q: polynomial;
    x: extended;
    i: integer;
  begin
    q := SortPolyZeros(p);
    x := CABS(q.coef[1]);
    i := 1;
    while (x = 0) and (i < p.deg) do begin
      i := i + 1;
      x := CABS(q.coef[i]);
    end;
    if x <> 0 then
      minPolyRadius := x
    else
      minPolyRadius := CABS(q.coef[0]);
  end;

```

```

end;

function SortPolyZeros (var p: polynomial): polynomial;
  var
    B: complexPairArray;
    q: polynomial;
    i: integer;
begin
  q := p;
  for i := 1 to p.deg do
    B.cPair[i].y := p.coef[i];
  SortCPairArray(B.cPair, p.deg);
  for i := 1 to p.deg do
    q.coef[i] := B.cPair[i].y;
  SortPolyZeros := q;
end;

function EvalRationalOnCircle (z: complex;
  {evaluates a rational fct with num=num1+num2 at N points on a circle of radius r about z }
  r: extended; N: integer; num1, num2, den: polynomial): polynomial;
  var
    q: polynomial;
    z1: complex;
    value: valueType;
    i, j: integer;
begin
  q.deg := N - 1;
  q.rep := sumRep;
  for i := 0 to q.deg do begin
    z1 := CADD(z, P_TO_R(Re_Im_to_C(r, 2 * PI / N * i)));
    q.coef[i] := CDIV(CADD(PolyEval(z1, num1), PolyEval(z1, num2)), PolyEval(z1, den));
  end;
  EvalRationalOnCircle := q;
end;

function DFT_Coefs (k, N: integer; r: extended; var p: polynomial): complex;
  var
    i: integer;
    value, z: complex;
begin
  value := p.coef[0];
  for i := 1 to N - 1 do begin
    if k = 0 then
      z := p.coef[i]
    else
      z := CMUL(p.coef[i], P_TO_R(Re_Im_to_C(1, -2 * PI * k * i / N)));
    value := CADD(value, z);
  end;
  value := SCALE(1 / N, value);
  for i := 1 to k do
    value := SCALE(1 / r, value);
  DFT_Coefs := value;
end;

```

end;

```
procedure StoreZero (testZero: complex; var found_p: polynomial);
begin
  found_p.deg := found_p.deg + 1;
  found_p.coef[found_p.deg] := ZeroCResidue(testZero);
end;
```

```
procedure InterChangePolys (var p1, p2: polynomial);
  var
    q: polynomial;
begin
  q := p2;
  p2 := p1;
  p1 := q;
end;
```

```
procedure FindCommonZeros (var p1, p2, common_p: polynomial);
  var
    i, j, k, m, n, d: integer;
    InterChanged: boolean;
begin
  InterChanged := FALSE;
  common_p := UnitPoly(prodRep);
  if p2.rep = sumRep then
    writeln('FindCommonZeros is not available in sumRep')
  else if (p1.deg > 0) and (p2.deg > 0) then begin {begin outer if}
    if p2.deg < p1.deg then begin
      InterChanged := TRUE;
      InterChangePolys(p1, p2);
    end;
    d := p1.deg;
    k := 0;
    repeat
      k := k + 1;
      m := -1;
      n := -1;
      for i := 1 to p1.deg do begin
        if CEqual(p1.coef[i], CZERO) then begin
          for j := 1 to p2.deg do
            if CEqual(p2.coef[j], CZERO) then begin
              m := j;
              n := i;
            end;
          end
        else begin
          for j := 1 to p2.deg do
            if RelativeCEquality(p2.coef[j], p1.coef[i], 1e-8) then begin
              m := j;
              n := i;
            end;
          end;
        end;
      end;
    end;
```

```

    end;{i-loop}
  if (1 <= m) and (m <= p2.deg) then begin
    StoreZero(p2.coef[m], common_p);
    p1 := RemoveFactor_i(n, p1);
    p2 := RemoveFactor_i(m, p2);
    end;
  until (k = d);
  if InterChanged then
    InterChangePolys(p1, p2);
  end; {outer if}
end;

function RemoveFactor_i (i: integer; var p: polynomial): polynomial;
  var
    j: integer;
    q: polynomial;
  begin
  if p.rep = sumRep then begin
    writeln('RemoveFactor_i not available in sumRep');
    ExitToShell;
  end
  else begin
    if p.deg = 0 then
      q := p
    else begin
      q.deg := p.deg - 1;
      for j := 0 to i - 1 do
        q.coef[j] := p.coef[j];
      for j := i to q.deg do
        q.coef[j] := p.coef[j + 1];
      end;
    end;
  q.rep := p.rep;
  RemoveFactor_i := q;
  end;

function PicksHighestDegPoly (var p1, p2: polynomial): polynomial;
begin
if p1.deg >= p2.deg then
  PicksHighestDegPoly := p1
else
  PicksHighestDegPoly := p2;
end;

function PolyDelay (z: complex; var p: polynomial): complex;
  var
    i: integer;
    z1, z2: complex;
    q: polynomial;
  begin
  if p.rep = prodRep then begin
    z1 := CZERO;

```

```

for i := 1 to p.deg do begin
  z2 := CSUB(z, p.coef[i]);
  if CABS(z2) = 0 then begin
    z1.Re := INF;
    z1.Im := INF;
  end
  else
    z1 := CADD(z1, CDIV(C_ONE, z2));
  end; {end for loop}
end {end if prodRep}
else begin
  q := PolyDeriv(p);
  z2 := SumPolyEvaluation(z, p);
  if CABS(z2) = 0 then begin
    z1.Re := INF;
    z1.Im := INF;
  end
  else begin
    z1 := SumPolyEvaluation(z, q);
    z1 := CDIV(z1, z2);
  end;
  end;
  PolyDelay := z1;
end;

function PolyDeriv (var p: polynomial): polynomial;
  var
    i: integer;
    q, q1: polynomial;
  begin
  if p.rep = prodRep then begin
    q := RemoveFactor_i(1, p);
    for i := 2 to p.deg do begin
      q1 := RemoveFactor_i(i, p);
      q := AddPoly(q, q1);
    end;
  end
  else begin
    if p.deg = 0 then begin
      q.deg := 0;
      q.coef[0] := CZERO;
    end
    else begin
      q.deg := p.deg - 1;
      for i := 0 to q.deg do
        q.coef[i] := SCALE((i + 1), p.coef[i + 1]);
      end;
    end;
  end;
  q.rep := p.rep;
  PolyDeriv := q;
end;

```

```

function PolyDelayPair (z: complex; var p1, p2: polynomial): complexPair;
  var
    z1, z2, z3: complex;
    c_pair: complexPair;
begin
  z1 := polyEval(z, p1);
  z2 := polyEval(z, p2);
  z3 := CADD(z1, z2);
if CABS(z3) = 0 then begin
  z1.Re := INF;
  z1.Im := INF;
  z2.Re := INF;
  z2.Im := INF;
  end
else begin
  z1 := CDIV(z1, z3);
  z2 := CDIV(z2, z3);
  end;
  c_pair.x := CADD(CMUL(z1, PolyDelay(z, p1)), CMUL(z2, PolyDelay(z, p2)));
  c_pair.y := CADD(CMUL(z1, PolyDelayII(z, p1)), CMUL(z2, PolyDelayII(z, p2)));
  PolyDelayPair := c_pair;
end;

```

```

function PolyDelayII (z: complex; var p: polynomial): complex;
  var
    i, j: integer;
    z1, z2: complex;
    q, q1: polynomial;
begin
if p.rep = prodRep then begin
  z1 := CZERO;
  for i := 1 to p.deg do
    for j := i + 1 to p.deg do begin
      z2 := CMUL(CSUB(z, p.coef[i]), CSUB(z, p.coef[j]));
      if CABS(z2) = 0 then begin
        z1.Re := INF;
        z1.Im := INF;
        end
      else
        z1 := CADD(z1, CDIV(C_ONE, z2));
      end; {end for loop}
    z1 := SCALE(2, z1);
  end {end if prodRep}
else begin
  q := PolyDeriv(p);
  q1 := PolyDeriv(q);
  z2 := SumPolyEvaluation(z, p);
  if CABS(z2) = 0 then begin
    z1.Re := INF;
    z1.Im := INF;
    end
  else begin

```



```

    z1 := SumPolyEvaluation(z, q1);
    z1 := CDIV(z1, z2);
  end;
end;
PolyDelayII := z1;
end;

function LaguerreStep (z: complex;    {see Orchard CAS Nov 1989 p.1377}
                      p1, p2, q: polynomial): complex;
  var
    a0, a1, a2, H, new_z, sqrtH, den1, den2, den, q_delay: complex;
    m: integer;
    delayPair: complexPair;
  begin
    if p1.deg > p2.deg then
      m := p1.deg
    else
      m := p2.deg;
    m := m - q.deg;
    q_delay := PolyDelay(z, q);
    a1 := CSUB(PolyDelayPair(z, p1, p2).x, q_delay);
    a2 := CSUB(PolyDelayPair(z, p1, p2).y, PolyDelayII(z, q));
    a0 := SCALE(2, CMUL(a1, q_delay));
    H := CMUL(SCALE(m - 1, a1), SCALE(m - 1, a1));
    H := CSUB(H, SCALE(m * (m - 1), CSUB(a2, a0)));
    sqrtH := CSQRT(H);
    den1 := CSUB(a1, sqrtH);
    den2 := CADD(a1, sqrtH);
    if CABS(den1) > CABS(den2) then
      den := den1
    else
      den := den2;
    new_z := CSUB(z, SCALE(m, CDIV(C_ONE, den)));
    LaguerreStep := new_z;
  end; {function}

function FindprodRepOfSum (var q1, q2: polynomial): polynomial; {q1, q2 are in prodRep}
  var
    startZero, testZero, constFactor, threshold, NZValAtZero: complex;
    z, z1, z2, z3, z4, m1, m2, m3: complex;
    k, counter, startingZeroCount, potDeg, sumDegs: integer;
    degRed, ConjPairPresent, ZeroFound: boolean;
    p1, p2, found_p, common_p, startPoly: polynomial;
    r: extended;

  {*****}

  function FctVal (startZero: complex; p1, p2: polynomial): complex;
  begin
    FctVal := CADD(PolyEval(startZero, p1), PolyEval(startZero, p2));
  end;

```

```

function FctValZeroOrConst (m1, m2, m3, z1, z2, z3: complex): boolean;
  var
    dev: extended;
    Value: boolean;
  begin
    Value := FALSE;
    if CEqual(z3, CZERO) or CEqual(m3, CZERO) then
      Value := TRUE
    else begin
      dev := CABS(CDIV(CSUB(m3, z3), m3));
      if (dev < 1e-5) or CEqual(m1, m3) then {m1=m3 detects oscillations with period=2}
        Value := TRUE;
      end;
    if FALSE then begin
      writeln('mean=', m3.Re : 30, ' ', m3.Im : 30);
      writeln('dev=', dev : 30);
      end;
    FctValZeroOrConst := Value;
  end;

function FctValEqualsZero (startZero: complex; {changed to 1}
                           p1, p2: polynomial; tolerance: extended): boolean;

  var
    z1: complex;
  begin
    FctValEqualsZero := FALSE;
    z1 := CDIV(CADD(PolyEval(startZero, p1), PolyEval(startZero, p2)), NZValAtZero);
    if (CABS(z1) < tolerance) then begin
      FctValEqualsZero := TRUE;
      end;
  end;

procedure CheckForDegreeReduction (var p1, p2: polynomial; var potDeg: integer;
                                     var degRed: boolean; var constFactor: complex);

  var
    tol, r: extended;
    z: complex;
    q: polynomial;
  begin {assumes p1.deg = p2.deg}
    degRed := FALSE;
    tol := 1e-15;
    r := 1.1 * MaxOfTwoExtendeds(maxPolyRadius(p1), maxPolyRadius(p2));
    if r = 0 then
      r := 1;
    if RelativeCEquality(p1.coef[0], CNEGATE(p2.coef[0]), tol) then begin
      repeat
        potDeg := potDeg - 1;
        q := EvalRationalOnCircle(CZERO, r, potDeg + 1, p1, p2, UnitPoly(prodRep));
        z := DFT_Coefs(potDeg, potDeg + 1, r, q);
      until not RelativeZero(z, p1.coef[0], tol) or (potDeg = 0);
      degRed := TRUE;
      constFactor := ZeroCResidue(z);
    end;

```

```

    if (potDeg = 0) then
      if RelativeZero(z, p1.coef[0], tol) then
        constFactor := CZERO;
      end {end if}
    else
      constFactor := CADD(p1.coef[0], p2.coef[0]);
    end;

procedure FindZerosAtOrigin (var p1, p2, found_p: polynomial;
                             var NZValAtZero: complex);

  var
    maxDeg: integer;
    tol, r: extended;
    z: complex;
    q: polynomial;
  begin
    tol := 1e-10;
    r := minOfTwoExtendeds(minPolyRadius(p1), minPolyRadius(p2)) / 2;
    if r = 0 then
      r := 1;
    maxDeg := MaxOfTwoIntegers(p1.deg, p2.deg);
    z := CADD(PolyEval(CZERO, p1), PolyEval(CZERO, p2));
    while RelativeZero(z, p1.coef[0], tol) and (found_p.deg < maxDeg) do begin
      StoreZero(CZERO, found_p);
      q := EvalRationalOnCircle(CZERO, r, maxdeg, p1, p2, found_p);
                                                {rational=(p1+p2)/found_p}
      z := DFT_Coefs(0, maxdeg + 1, r, q);
    end; {end while}
    NZValAtZero := z;
  end;

procedure SumTwoNonZeroPolysDegLessThanEqual1 (p1, p2: polynomial;
                                                  var found_p: polynomial);

  begin
  if (p1.deg = 0) and (p2.deg = 0) then begin
    found_p.coef[0] := CADD(p1.coef[0], p2.coef[0]);
    if RelativeZero(found_p.coef[0], p1.coef[0], 1e-10) then {set to 1e-10 on 90/9/23}
      found_p.coef[0] := CZERO;
    end;
  if (p1.deg = 0) and (p2.deg = 1) then begin
    found_p := p2;
    found_p.coef[1] := CSUB(p2.coef[1], CDIV(p1.coef[0], p2.coef[0]));
    end;
  if (p1.deg = 1) and (p2.deg = 0) then begin
    found_p := p1;
    found_p.coef[1] := CSUB(p1.coef[1], CDIV(p2.coef[0], p1.coef[0]));
    end;
  if (p1.deg = 1) and (p2.deg = 1) then begin
    found_p.deg := 1;
    found_p.coef[0] := CADD(p1.coef[0], p2.coef[0]);
    if RelativeZero(found_p.coef[0], p1.coef[0], 1e-10) then begin
      found_p.deg := 0;

```

```

        found_p.coef[0] := CMUL(p1.coef[0], CSUB(p2.coef[1], p1.coef[1]));
    end
else
    found_p.coef[1] := CDIV(CADD(CMUL(p1.coef[0], p1.coef[1]), CMUL(p2.coef[0],
        p2.coef[1])), found_p.coef[0]);
end;
end;
end;

{*****}

begin    {begin FindprodRepOfSum }
p1 := q1;
p2 := q2;
common_p := UnitPoly(prodRep);
FindCommonZeros(p1, p2, common_p);
found_p := UnitPoly(prodRep);
{Special cases}
if (p1.deg = 0) and CEqual(p1.coef[0], CZERO) then
    found_p := p2;
if (p2.deg = 0) and CEqual(p2.coef[0], CZERO) then
    found_p := p1;
if (p1.deg <= 1) and (p2.deg <= 1) and not CEqual(p1.coef[0], CZERO) and not
    CEqual(p2.coef[0], CZERO) then
    SumTwoNonZeroPolysDegLessThanEqual1(p1, p2, found_p);
{end special cases}
{ begin nonspecial case}
if ((p1.deg > 1) or (p2.deg > 1)) and not CEqual(p1.coef[0], CZERO) and not
    CEqual(p2.coef[0], CZERO) then begin
    startPoly := PicksHighestDegPoly(p1, p2);
    startPoly := SortPolyZeros(startPoly);
    if p1.deg > p2.deg then begin
        potDeg := p1.deg;
        constFactor := p1.coef[0];
    end;
    if p2.deg > p1.deg then begin
        potDeg := p2.deg;
        constFactor := p2.coef[0];
    end;
    if p2.deg = p1.deg then begin
        potDeg := p1.deg;
        CheckForDegreeReduction(p1, p2, potDeg, degRed, constFactor);
    end;
    FindZerosAtOrigin(p1, p2, found_p, NZValAtZero);
    if potDeg = found_p.deg then
        found_p.coef[0] := NZValAtZero;
    if potDeg > found_p.deg then begin
        startingZeroCount := 0;
    end;
    repeat
        startingZeroCount := startingZeroCount + 1;
        startZero := SCALE(1 + 1e-5, startPoly.coef[startingZeroCount]);
        if startZero.Im = 0 then
            if startZero.Re = 0 then

```

```

        startZero := ComplexNo(1, 1)
    else
        startZero.Im := startZero.Re;
    counter := 0;
    z2 := CZERO;
    z3 := CZERO;
    m2 := CZERO;
    m3 := CZERO;
    repeat {begin iterations}
        counter := counter + 1;
        testZero := LaguerreStep(startZero, p1, p2, found_p);
        startZero := ZeroCResidue(testZero);
        z := CDIV(CADD(PolyEval(startZero, p1), PolyEval(startZero, p2)), NZValAtZero);
        z1 := z2;
        z2 := z3;
        z3 := QUAD1(z);
        m1 := m2;
        m2 := m3;
        m3 := SCALE(1 / 3, CADD(CADD(z1, z2), z3));
        ZeroFound := FctValZeroOrConst(m1, m2, m3, z1, z2, z3) or (counter = 30);
    until ZeroFound;
    if CABS(CDIV(kthPowerOfC(potDeg, startZero), NZValAtZero)) = 0 then
        startZero := CZERO; {found zero can be set equal zero}
    StoreZero(startZero, found_p);
    ConjPairPresent := FALSE;
    if (found_p.deg >= 2) then
        if CEQUAL(CONJ(found_p.coef[found_p.deg - 1]), found_p.coef[found_p.deg]) then
            ConjPairPresent := TRUE
        else
            ConjPairPresent := FALSE;
    z := CSUB(FctVal(startZero, p1, p2), CONJ(FctVal(CONJ(startZero), p1, p2)));
    if (startZero.Im <> 0) and not ConjPairPresent and (potDeg > found_p.deg) then
        if RelativeZero(z, found_p.coef[0], 1e-17) then {changed 92/3/11}
            begin
                StoreZero(CONJ(startZero), found_p);
                startingZeroCount := startingZeroCount + 1;
            end;
    until (found_p.deg = potDeg);
    {end finding zeros}
end; {end potDeg > 0}
found_p.coef[0] := ZeroCResidue(constFactor);
end; {end nonspecial case }
FindprodRepOfSum := MulPoly(common_p, found_p);
end; {end function}

end.

```

The following program listing is a modification (by the author) of a unit written by Smith [27].

```

unit Plotting;
interface
  const
    MaxPts = 1000;
    MenuID = 1;
    DialogID = 100;
    Y_inf = 2.0e9;
  type
    PointsRange = 0..MaxPts;
    PlotArray = array[PointsRange] of real;
    PlotPtr = ^PlotRec;
    PlotRec = record
      ArraySize: PointsRange;
      ymin, ymax, xmin, xmax: real;
      ydiv, xdiv: integer;
      window: WindowPtr;
      xpts, ypts: PlotArray;
      discrete, grid: boolean;
    end;
  var
    PlotMenu: MenuHandle;
    PlotWindow: WindowPtr;
    Done: boolean;
    DragRect: Rect;

  procedure PlotVec (plot: PlotRec);
implementation
  function EndPoints (var x: PlotArray; xmin: real; size: PointsRange;
    Start: boolean): integer;

    var
      k: integer;
      found: boolean;
    begin
    if Start then
      EndPoints := 0
    else
      EndPoints := size;
    k := 0;
    found := false;
    while (k <= size) and not found do
      if (x[k] >= xmin) then begin
        EndPoints := k;
        found := true;
      end
    else
      k := k + 1;
    end;

```

```

function Scale (var ymin, ymax: real): real;
  var
    plotMax, plotScale, ymax0, ymin0: real;
    t, k: integer;
begin
if abs(ymin) > abs(ymax) then
  plotMax := abs(ymin)
else
  plotMax := abs(ymax);
plotScale := 1;
if plotMax < 1 then begin
  t := Trunc(ln(plotMax) / ln(10)) - 1;
  for k := 1 to (-t) do
    plotScale := plotScale / 10;
  end
else begin
  t := Trunc(ln(plotMax) / ln(10));
  for k := 1 to t do
    plotScale := plotScale * 10;
  end;
ymax0 := Round(ymax / plotScale + 0.49);
ymax := ymax0 * plotScale;
ymin0 := Round(ymin / plotScale - 0.49);
ymin := ymin0 * plotScale;
Scale := plotScale;
end;

function WriteScale (ymax: real; left, top: integer): integer;
  var
    s1, s2: Str255;
    power: integer;
begin
if (abs(ymax) >= 1) then
  power := trunc(ln(abs(ymax)) / ln(10) + 1.0e-8)
else
  power := round(ln(abs(ymax)) / ln(10) - 0.49);
if (power < 0) or (power > 2) then begin
  TextFace([bold]);
  s1 := '* 10e';
  s2 := StringOf(abs(power) : 1);
  if (power > 0) then
    s1 := concat(s1, '+', s2)
  else
    s1 := concat(s1, '-', s2);
  MoveTo(left, top);
  DrawString(s1);
  TextFace([]);
  end
else
  power := 0;
WriteScale := power;

```

end;

procedure XLabel (rec: Rect; xmin, xmax: real; grid: boolean; xdiv: integer);

var

k, hpos, power: integer;

xDelta, num: real;

s: Str255;

begin

if xdiv <= 0 **then**

xdiv := 5;

xDelta := (xmax - xmin) / xdiv;

power := WriteScale(xmax, rec.right - 20, rec.bottom + 30);

for k := 0 **to** xdiv **do begin**

hpos := Round(rec.left + k * (rec.right - rec.left) / xdiv);

MoveTo(hpos, rec.bottom + 3);

LineTo(hpos, rec.bottom);

if grid **then begin**

PenPat(gray);

LineTo(hpos, rec.top);

PenPat(black);

end;

num := k * xDelta + xmin;

num := num / exp(power * ln(10)); { change scale for output }

s := StringOf(num : 4 : 2);

MoveTo(hpos - 10, rec.bottom + 18);

DrawString(s);

end;

end;

procedure YLabel (rec: Rect; ymin, ymax, yscale: real; grid: boolean);

var

k, vpos, ndiv, power: integer;

num: real;

s, s1, s2: Str255;

begin

yscale := yscale / 1; {added to change scale divisions: divide by 1, 2, 5 or 10 etc.}

ndiv := Round((ymax - ymin) / yscale);

power := WriteScale(ymax, rec.left - 47, rec.top - 8);

for k := 0 **to** ndiv **do begin**

vpos := Round(rec.bottom - k * (rec.bottom - rec.top) / ndiv);

MoveTo(rec.left - 3, vpos);

LineTo(rec.left, vpos);

if grid **then begin**

PenPat(gray);

LineTo(rec.right, vpos);

PenPat(black);

end;

num := ymin + k * yscale;

num := num / exp(power * ln(10)); { change scale for output }

s := StringOf(num : 7 : 2);

MoveTo(rec.left - 40, vpos + 4);

DrawString(s);


```

    end;
end;

procedure PlotCurve (var plot: PlotRec);
  var
    hpos, vpos, xaxis, i, j: integer;
    n1, n2: PointsRange;
    hscale, vscale, ymin, ymax, yscale: real;
    rec, rec2: Rect;
begin
  TextFont(times);
  TextSize(10);
  rec.top := thePort^.portRect.top + 20;           { set graph rectangle }
  rec.bottom := thePort^.portRect.bottom - 50;
  rec.left := thePort^.portRect.left + 50;
  rec.right := thePort^.portRect.right - 35;
  ymin := +INF;
  ymax := -INF;
  n1 := EndPoints(plot.xpts, plot.xmin, plot.ArraySize, TRUE);
  n2 := EndPoints(plot.xpts, plot.xmax, plot.ArraySize, FALSE);
  hscale := (rec.right - rec.left) / (n2 - n1);
  vscale := rec.bottom - rec.top;
  for i := n1 to n2 do begin                     { find max & min points of plot }
    if plot.ypts[i] > ymax then
      ymax := plot.ypts[i];
    if plot.ypts[i] < ymin then begin
      ymin := plot.ypts[i];
      j := i;
    end;
  end;
  if ymax > plot.ymax then                         { set max point if less than Y_inf }
    ymax := plot.ymax;
  if ymax > Y_inf then
    ymax := Y_inf;
  if ymin < plot.ymin then                         { set min point if less than -Y_inf }
    ymin := plot.ymin;
  if ymin < -Y_inf then
    ymin := -Y_inf;
  yscale := Scale(ymin, ymax);
  if plot.ydiv > 0 then
    yscale := (ymax - ymin) / plot.ydiv;
  xaxis := rec.top + round(vscale * ymax / (ymax - ymin));
  XLabel(rec, plot.xpts[n1], plot.xpts[n2], plot.grid, plot.xdiv);
  YLabel(rec, ymin, ymax, yscale, plot.grid);
  MoveTo(rec.right, xaxis);
  LineTo(rec.left, xaxis);
  if not plot.discrete then
    for i := n1 to n2 do begin
      hpos := rec.left + round((i - n1) * hscale);
      if plot.ypts[i] > ymax then
        vpos := rec.top
      else if plot.ypts[i] < ymin then

```

```

        vpos := rec.bottom
    else
        vpos := xaxis - round(vscalescale * plot.ypts[i] / (ymax - ymin));
    LineTo(hpos, vpos);
    end
else
    for i := n1 to n2 do begin
        hpos := rec.left + round((i - n1) * hscalescale);
        if plot.ypts[i] <= ymax then begin
            vpos := xaxis - round(vscalescale * plot.ypts[i] / (ymax - ymin));
            MoveTo(hpos, xaxis);
            LineTo(hpos, vpos);
            PaintCircle(hpos, vpos, 2);
        end
    end;
rec.right := rec.right + 1;
rec.bottom := rec.bottom + 1;
FrameRect(rec);
end;

function SetPlotParam (old_num: real; msg: string): real;
var
    item, kind: integer;
    str: Str255;
    dlog: DialogPtr;
    box: Rect;
    hand: Handle;
    num: real;
begin
    str := "";
    ParamText(msg, ", ", "");
    dlog := GetNewDialog(DialogID, nil, pointer(-1));
    ShowWindow(dlog);
    repeat
        ModalDialog(nil, item);
        GetDItem(dlog, item, kind, hand, box);
        if item = 3 then
            GetIText(hand, str);
    until (item = 1) or (item = 2);
    if (item = 1) and (str <> "") then begin
        ReadString(str, num);
        SetPlotParam := num;
    end
    else if (str = "") then
        SetPlotParam := old_num;
    DisposDialog(dlog);
end;

procedure SpoolOutPICT (var picHand: picHandle);
var
    where: Point;
    reply: SFReply;

```

```

    longZero, byteCount: longint;
    err, fRefNum, k: integer;
begin
where.h := 104;
where.v := 45;
SFPutFile(where, 'Save the PICT as:', 'pict ', nil, reply);
if reply.good then begin
    err := Create(reply.fName, reply.vRefNum, '????', 'PICT');
    err := FOpen(reply.fName, reply.vRefNum, fRefNum);
    longZero := 0;
    byteCount := SizeOf(longZero);
    for k := 1 to (512 div byteCount) do
        err := FWrite(fRefNum, byteCount, @longZero);
    byteCount := picHand^^.picSize;
    err := FWrite(fRefNum, byteCount, Ptr(picHand^));
    err := FSClose(fRefNum);
    end;
KillPicture(picHand);
end;

procedure MenuCmnd (result: longint; var plot: PlotRec);
    var
        item, menu: integer;
        rec: Rect;
        pic: PicHandle;
begin
menu := HiWord(result);
if menu = MenuID then begin
    item := LoWord(result);
    case item of
        1:
            plot.ymax := SetPlotParam(plot.ymax, 'Y-max');
        2:
            plot.ymin := SetPlotParam(plot.ymin, 'Y-min');
        3:
            plot.xmax := SetPlotParam(plot.xmax, 'X-max');
        4:
            plot.xmin := SetPlotParam(plot.xmin, 'X-min');
        6:
            plot.ydiv := Round(SetPlotParam(plot.ydiv, 'Y-div'));
        7:
            plot.xdiv := Round(SetPlotParam(plot.xdiv, 'X-div'));
        9:
            plot.grid := not plot.grid;
        10:
            plot.discrete := not plot.discrete;
        11: begin
            rec := thePort^.portRect;
            pic := OpenPicture(rec);
            PlotCurve(plot);
            ClosePicture;
            SpoolOutPICT(pic);
    end;
end;

```

```

    end;
12: begin
    DisposeWindow(PlotWindow);
    Done := true;
    end;
otherwise
    end;
if item <= 10 then
    InvalRect(thePort^.portRect);
    end;
HiliteMenu(0);
end;

procedure ResizeWindow (w1: WindowPtr; pt: Point);
    var
        rec: Rect;
        l: longint;
    begin
    SetPort(w1);
    with screenbits.bounds do
        SetRect(rec, 200, 100, right, bottom - 20);
    l := GrowWindow(w1, pt, rec);
    SetRect(rec, 0, 0, LoWord(l), HiWord(l));
    InvalRect(rec);
    SizeWindow(w1, LoWord(l), HiWord(l), true);
    end;

procedure MouseEvent (event: EventRecord; var plot: PlotRec);
    var
        w1: WindowPtr;
    begin
    case FindWindow(event.where, w1) of
    inSysWindow:
        SystemClick(event, w1);
    inMenuBar:
        MenuCmnd(MenuSelect(event.where), plot);
    inDrag:
        if w1 = PlotWindow then
            DragWindow(w1, event.where, DragRect);
    inGoAway:
        if w1 = PlotWindow then begin
            DisposeWindow(w1);
            Done := true;
            end;
    inGrow:
        if w1 = PlotWindow then
            ResizeWindow(w1, event.where);
    otherwise
    end;
end;

procedure UpEvent (event: EventRecord; var plot: PlotRec);

```

```

    var
      w: WindowPtr;
      ref: longint;
  begin
    w := WindowPtr(event.message);
    if w = PlotWindow then begin
      SetPort(w);
      BeginUpdate(w);
      EraseRect(thePort^.portRect);
      DrawGrowIcon(w);
      PlotCurve(plot);
    end;
    EndUpdate(w);
  end;

  procedure AddPlotMenu;
  begin
    PlotMenu := NewMenu(MenuID, 'Plot');
    AppendMenu(PlotMenu, 'Set Y-max;Set Y-min;Set X-max;Set X-min');
    AppendMenu(PlotMenu, '(-;Set Y-div;Set X-div');
    AppendMenu(PlotMenu, '(-;Toggle Grid;Toggle Mode;Save As PICT;Quit');
    InsertMenu(PlotMenu, 0);
    DrawMenuBar;
  end;

  procedure PlotVec;
  var
    rec: Rect;
    event: EventRecord;
  begin
    with screenbits.bounds do
      SetRect(DragRect, 4, 24, right - 4, bottom - 4);
    Done := false;
    AddPlotMenu;
    SetRect(rec, 10, 50, 470, 250);
    PlotWindow := NewWindow(nil, rec, 'Plot Window', true, 0, pointer(-1), true, 0);
    plot.window := PlotWindow;
  repeat
    SystemTask;
    if GetNextEvent(everyEvent, event) then
      case event.what of
        mouseDown:
          MouseEvent(event, plot);
        updateEvt:
          UpEvent(event, plot);
        otherwise
          end
      until Done;
    DeleteMenu(MenuID);
    DisposeMenu(PlotMenu);
  end;
end.

```

Appendix B

Program Listing of Devleeschouwer and Grenez Algorithm

The Devleeschouwer & Grenez Algorithm is implemented as a computer program which is shown later. As mentioned before, this program is written in the PASCAL computer language.

The datas in the input file for this program have the following format:

# of level in the left stopband	
normalized frequency (in kHz)	characteristic attenuation (in dB)
...	...
# of level in the passband	
normalized frequency (in kHz)	characteristic attenuation (in dB)
...	...
# of level in the right stopband	
normalized frequency (in kHz)	characteristic attenuation (in dB)
...	...
m'_0	
n_{BP}	
n_{BS}	
C_β	
initial zero (in kHz)	m'_i
...	...
initial pole (in kHz)	m'_i (for the right stopband)
...	...
initial pole (in kHz)	m'_i (for the left stopband)
...	...
W_{BP}	
W_{BS}	
True or False for allowing TZ is real.	

```

program linear;
  uses
    plotting;
  const
    Pi = 3.141592653589793238462643383279;
    MaxDegree = 20;
    tolerance8 = 1.0e-8;
    tolerance10 = 1.0e-10;
    Search_Limit = 5;
    Initial_Limit = 10;
    inBeta = TRUE;
    indB = FALSE;
  type
    arrayType = array[0..MaxDegree] of extended;
    polynomial = record
      deg: integer;
      coef: arrayType;
      m: array[1..MaxDegree] of integer;
    end;
    Signal = record
      total: integer;
      Xi: arrayType;
      mag: arrayType;
    end;
    Matrix = array[0..MaxDegree] of arrayType;
  var
    Xpoly, Ypoly, Optimal_Xpoly, Old_poly: polynomial;
    Pass_Freq, Stop_Freq: Signal;
    Pass_Band, Left_Stop, Right_Stop: Signal;
    NBP, NBS: integer;
    WBP, WBS: integer;
    DataFile: text;
    t2, t1: extended;
    i, k, Optimal_Index, Best_Iteration: integer;
    r, c, ln10, TwoPi: extended;
    A: Matrix;
    Matrix_Size, Nplus1, Nminus1, N: integer;
    StartPt, EndPt, lastPt: extended;
    Optimal_Least_Distance, Least_Distance: extended;
    Optimal_Sought, No_real_TZ, negative_Zero: boolean;
    Negative_Pole, Positive_Zero, Not_in_Order: boolean;
    Under_Stop, Above_Pass: boolean;
    StartSec, EndSec: longint;
    DateRec: DateTimeRec;

  procedure DateTime;
    var
      secs: longint;
      DateRec: DateTimeRec;
  begin
    GetDateTime(secs);
    Secs2Date(secs, DateRec);

```

```

with DateRec do begin
  writeln(year : 4, '/', month : 2, '/', day : 2);
  writeln(hour : 2, ':', minute : 2, ':', second : 2);
  end;
end;

{Selects data file produced by "apple Edit" }
procedure SelectDataFile (var DataFile: text; prompt: Str255);
  var
    windowRect: Rect;
    StringName: Str255;
begin
  HideAll;
with windowRect do begin
  left := ScreenBits.bounds.right div 2 - 150;
  right := left + 300;
  bottom := ScreenBits.bounds.bottom - 10;
  top := bottom - 28;
  end;
  SetTextRect(windowRect);
  ShowText;
  write(prompt);
  SysBeep(2);
  SysBeep(2);
  StringName := OldFileName('select data file from disk');
  writeln(' ', StringName);
  reset(DataFile, StringName);
end;      {procedure SelectDataFile}

procedure OpenTextWindow;
  var
    windowRect: Rect;
begin
with windowRect do begin
  left := 20;
  right := ScreenBits.bounds.right - 20;
  bottom := ScreenBits.bounds.bottom;
  top := 40;
  end;
  SetTextRect(windowRect);
  ShowText;
end;

procedure Read_Signal (var DataFile: text; var Left, Pass, Right: Signal);
  procedure Input_Signal (var DataFile: text; var band: Signal);
    var
      i: integer;
    begin
    readln(DataFile, band.total);
    for i := 0 to band.total do
      readln(DataFile, band.Xi[i], band.mag[i]);
    end;

```



```

begin
Input_Signal(DataFile, Left);
Input_Signal(DataFile, Pass);
Input_Signal(DataFile, Right);
end;

procedure Print_Signal (var Left, Pass, Right: Signal; msg1, msg2: Str255);
procedure Output_Signal (var band: Signal; msge: Str255);
    var
        i: integer;
    begin
        writeln(msge, ':', ' ' : 5);
        for i := 0 to band.total do
            writeln(band.Xi[i] : 29 : 20, band.mag[i] : 30 : 20)
        end;
    begin
        writeln;
        writeln('Specifications:', msg1 : 9, msg2 : 29);
        Output_Signal(Left, 'Left Stopband');
        Output_Signal(Pass, 'Passband');
        Output_Signal(Right, 'Right Stopband');
    end;

function Transform (Y: extended): extended;
begin
    Transform := (Y - t2) / (Y - t1);
end;

procedure Input_Initial (var DataFile: text; var poly: polynomial;
                        var NBP, NBS, WBP, WBS: integer; var No_real_TZ: boolean);
    var
        i, N, Mi, Mo: integer;
    begin
        readln(DataFile, Mo);
        readln(DataFile, NBP);
        readln(DataFile, NBS);
        N := NBP + NBS;
        Mi := 0;
        readln(DataFile, poly.coef[0]);
        writeln;
        writeln('Initial Rational function :');
        writeln('i' : 2, 'Yi' : 13, 'mi' : 12, 'Xi' : 25);
        writeln('Cx = ' : 35, poly.coef[0] : 24 : 20);
        for i := 1 to N do begin
            readln(DataFile, poly.coef[i], poly.m[i]);
            write(i : 2, poly.coef[i] : 15 : 5, poly.m[i] : 10);
            Mi := Mi + poly.m[i];
            poly.coef[i] := sqr(TwoPi * poly.coef[i]);
            poly.coef[i] := Transform(poly.coef[i]);
            writeln(poly.coef[i] : 32 : 20);
        end;
        poly.deg := NBP + NBS + 2;

```

```

N := poly.deg - 1;
poly.m[N] := -Mo - Mi;
poly.coef[N] := 1;
poly.m[poly.deg] := Mo;
poly.coef[poly.deg] := t2 / t1;
writeln(N : 2, 'M $\infty$  = ' : 23, poly.m[N] : 2, poly.coef[N] : 32 : 20);
writeln(poly.deg : 2, 'Mo = ' : 23, poly.m[poly.deg] : 2, poly.coef[poly.deg] : 32 : 20);
writeln;
readln(DataFile, WBP);
readln(DataFile, WBS);
readln(DataFile, No_real_TZ);
writeln('WBP = ', WBP : 1, ',', 'WBS = ' : 10, WBS : 1);
writeln('# of Attenuation Zeros = ', NBP : 1);
writeln('# of Transmission Zeros = ', NBS : 1);
writeln('No real Transmission Zero ? ', No_real_TZ);
writeln;
end;

```

```

procedure Freq_to_X (var Pass, Left, Right: Signal);

```

```

  var

```

```

    Temp: Signal;

```

```

  procedure convert (var Band: Signal);

```

```

    var

```

```

      i: integer;

```

```

  begin

```

```

  with Band do begin

```

```

    for i := 0 to total do

```

```

      Xi[i] := Transform(sqr(TwoPi * Xi[i]));

```

```

    for i := 1 to total do

```

```

      if (mag[i] > 20) then

```

```

        mag[i - 1] := mag[i] * ln10 / 10

```

```

      else

```

```

        mag[i - 1] := ln(abs(exp(mag[i] * ln10 / 10) - 1));

```

```

      mag[total] := mag[total - 1];

```

```

    end;

```

```

  end;

```

```

begin

```

```

Pass.Xi[0] := Pass.Xi[1];

```

```

Pass.Xi[Pass.total] := Pass.Xi[1];

```

```

convert(Pass);

```

```

Pass.Xi[Pass.total] := 0;

```

```

Pass.Xi[0] := -INF;

```

```

convert(Left);

```

```

convert(Right);

```

```

Temp := Left;

```

```

Left := Right;

```

```

Right := Temp;

```

```

Left.Xi[Left.total] := 1;

```

```

end;

```

```

procedure Print_Error (msg: Str255; X: extended; code: integer);
begin
  write('ERROR : ');
  if (code = 1) then
    write(X : 1 : 20);
  writeln(msg);
  if (code = 3) then
    writeln(' ' : 10, 'Try another initial values or increase the order of poles/zeros. ')
  else
    HALT;
end;

function exponential (X: extended; power: integer): extended;
begin
  if (power = 0) then
    exponential := 1
  else if (power mod 2 = 0) then
    exponential := exp(power * ln(abs(X)))
  else if (X > 0) then
    exponential := exp(power * ln(X))
  else
    exponential := -exp(power * ln(-X));
end;

function poly_Eval (var poly: polynomial; Y: extended): extended;
  var
    eval: extended;
    i: integer;
begin
  eval := poly.coef[0];
  for i := 1 to poly.deg do
    eval := eval * exponential((Y - poly.coef[i]), -poly.m[i]);
  poly_Eval := 1 / (1 + eval);
end;

function Beta (var poly: polynomial; X: extended): extended;
  var
    i: integer;
    Alpha: extended;
begin
  Alpha := ln(abs(poly.coef[0]));
  for i := 1 to poly.deg do
    if (poly.m[i] < 0) then
      Alpha := Alpha - poly.m[i] * ln(abs(X - poly.coef[i]));
  Beta := Alpha;
end;

procedure Plot_Signal (var poly: polynomial; Start, Stop: extended; inBeta: boolean;
  OutFileName: Str255);
  var
    k: integer;
    plot: PlotPtr;

```

```

    outFile: text;
    X, Y, h: extended;
begin
  New(plot);
  rewrite(outFile, OutFileName);
  writeln(outFile, '*');
  writeln(outFile, 'Freq', chr(9), 'Attn(dB)');
  plot^.ArraySize := MaxPts;
  h := (Stop - Start) / plot^.ArraySize;
  X := Start - h;
for k := 0 to plot^.ArraySize do begin
  X := X + h;
  plot^.xpts[k] := X;
  if inBeta then
    plot^.ypts[k] := Beta(poly, X)
  else
    plot^.ypts[k] := -10.0 * ln(poly_Eval(poly, sqrt(TwoPi * X))) / ln10;
  writeln(outFile, plot^.xpts[k] : 10 : 6, chr(9), plot^.ypts[k] : 25 : 10);
  end;
  plot^.xmin := Start;
  plot^.xmax := Stop;
  plot^.ymax := INF;
  plot^.ymin := -INF;
  plot^.discrete := FALSE;
  plot^.grid := TRUE;
  plot^.xdiv := 10;
  plot^.ydiv := 10;
  PlotVec(plot^);
  dispose(plot);
  close(outFile);
end;

function Beta_Between (var Band: Signal; X: extended): extended;
  var
    i: integer;
begin
  i := 1;
  while (X > Band.Xi[i]) and (i < Band.total) do
    i := i + 1;
  Beta_Between := Band.mag[i - 1];
end;

function Floor (X: extended): extended;
  var
    Flr: extended;
begin
  if (X < 0) then
    Flr := Beta_Between(Pass_Band, X)
  else if (ABS(X) < tolerance10) then
    Flr := Pass_Band.mag[Pass_Band.total]
  else if (abs(X - Left_Stop.Xi[0]) < tolerance10) then
    Flr := Left_Stop.mag[0]

```

```

else if (X > Left_Stop.Xi[0]) and (X < Left_Stop.Xi[Left_Stop.total]) then
  Flr := Beta_Between(Left_Stop, X)
else if (abs(X - Left_Stop.Xi[Left_Stop.total]) < tolerance10) then
  Flr := Left_Stop.mag[Left_Stop.total]
else if (abs(X - Right_Stop.Xi[0]) < tolerance10) then
  Flr := Right_Stop.mag[0]
else if (X > Right_Stop.Xi[0]) and (X < Right_Stop.Xi[Right_Stop.total]) then
  Flr := Beta_Between(Right_Stop, X)
else if (abs(X - Right_Stop.Xi[Right_Stop.total]) < tolerance10) then
  Flr := Right_Stop.mag[Right_Stop.total]
else
  Print_Error('is out of the Stopbands', X, 1);
Floor := Flr;
end;

function Over_Floor (var U: polynomial; X: extended): extended;
begin
Over_Floor := Beta(U, X) - Floor(X);
end;

function Under_Floor (var U: polynomial; X: extended): extended;
begin
Under_Floor := Floor(X) - Beta(U, X);
end;

function Min_Golden (var U: polynomial; AX, CX: extended;
                    function Min_Func (var P: polynomial; e: extended): extended): extended;
  var
    f0, f1, f2, f3: extended;
    BX, X0, X1, X2, X3: extended;
begin
BX := AX + (CX - AX) / 3.0;
X0 := AX;
X3 := CX;
X1 := BX;
X2 := BX + c * (CX - BX);
f1 := Min_Func(U, X1);
f2 := Min_Func(U, X2);
while (ABS(X3 - X0) > tolerance8 * (ABS(X1) + ABS(X2)))
and (ABS(X3 - X0) > tolerance10) do begin
  if f2 < f1 then begin
    X0 := X1;
    X1 := X2;
    X2 := r * X1 + c * X3;
    f0 := f1;
    f1 := f2;
    f2 := Min_Func(U, X2);
  end
else begin
    X3 := X2;
    X2 := X1;
    X1 := r * X2 + c * X0;

```

```

    f3 := f2;
    f2 := f1;
    f1 := Min_Func(U, X1);
  end
end;
f0 := Min_Func(U, X0);
f3 := Min_Func(U, X3);
if (f1 < f2) then
  if (f0 < f1) then
    Min_Golden := X0
  else
    Min_Golden := X1
  else if (f2 < f3) then
    Min_Golden := X2
  else
    Min_Golden := X3
end;

procedure Check_Stop (Xa, Xb: extended; var StartPt, EndPt: extended;
                      var Split: boolean);

begin
if (Xa < Left_Stop.Xi[0]) then
  StartPt := Left_Stop.Xi[0]
else if (Xa > Left_Stop.Xi[Left_Stop.total]) and (Xa < Right_Stop.Xi[0]) then
  StartPt := Right_Stop.Xi[0]
else
  StartPt := Xa;

if (Xa <= Left_Stop.Xi[Left_Stop.total]) and (Xb >= Right_Stop.Xi[0]) then begin
  Split := TRUE;
  EndPt := Left_Stop.Xi[Left_Stop.total];
end
else begin
  Split := FALSE;
  if (Xb > Right_Stop.Xi[Right_Stop.total]) then
    EndPt := Right_Stop.Xi[Right_Stop.total]
  else if (Xb > Left_Stop.Xi[Left_Stop.total]) and (Xb < Right_Stop.Xi[0]) then
    EndPt := Left_Stop.Xi[Left_Stop.total]
  else
    EndPt := Xb;
  end;
end;

procedure Locate_Abscissa (var poly: polynomial; var Pass_Freq, Stop_Freq: Signal);
  var
    i, k: integer;
    StartPt, EndPt: extended;
    Split, In_Left_Transit, In_Right_Transit: boolean;
    Minimum, New_Index, New_Minimum: extended;
begin
Pass_Freq.Xi[1] := -INF;
for i := 2 to NBP do

```

```

    Pass_Freq.Xi[i] := Min_Golden(poly, poly.coef[i - 1], poly.coef[i], Under_Floor);
    Pass_Freq.Xi[Pass_Freq.total] := 0;

```

```

for i := 1 to NBS do begin

```

```

    k := NBP + i;

```

```

    if (i = 1) then

```

```

        Check_Stop(0, poly.coef[k], StartPt, EndPt, Split)

```

```

    else

```

```

        Check_Stop(poly.coef[k - 1], poly.coef[k], StartPt, EndPt, Split);

```

```

    Stop_Freq.Xi[i] := Min_Golden(poly, StartPt, EndPt, Over_Floor);

```

```

    if Split then begin

```

```

        Minimum := Over_Floor(poly, Stop_Freq.Xi[i]);

```

```

        New_Index := Min_Golden(poly, Right_Stop.Xi[0], poly.coef[k], Over_Floor);

```

```

        New_Minimum := Over_Floor(poly, New_Index);

```

```

        if (New_Minimum < Minimum) then

```

```

            Stop_Freq.Xi[i] := New_Index;

```

```

        end;

```

```

    end;

```

```

    Check_Stop(poly.coef[poly.deg - 2], INF, StartPt, EndPt, Split);

```

```

    Stop_Freq.Xi[Stop_Freq.total] := Min_Golden(poly, StartPt, EndPt, Over_Floor);

```

```

end;

```

```

function Distance (var poly: polynomial; var Pass_Freq, Stop_Freq: Signal;
                    var AbovePass, UnderStop: boolean): extended;

```

```

    var

```

```

        dist, dist1: extended;

```

```

        i: integer;

```

```

begin

```

```

    dist := 0;

```

```

    AbovePass := FALSE;

```

```

for i := 2 to Pass_Freq.total do begin

```

```

    dist1 := Under_Floor(poly, Pass_Freq.Xi[i]);

```

```

    if (dist1 < 0) then begin

```

```

        dist := dist + sqr(dist1);

```

```

        AbovePass := TRUE;

```

```

    end;

```

```

end;

```

```

    UnderStop := FALSE;

```

```

for i := 1 to Stop_Freq.total do begin

```

```

    dist1 := Over_Floor(poly, Stop_Freq.Xi[i]);

```

```

    if (dist1 < 0) then begin

```

```

        dist := dist + sqr(dist1);

```

```

        UnderStop := TRUE;

```

```

    end;

```

```

end;

```

```

    Distance := sqrt(dist);

```

```

end;

```

```

procedure Create_Matrix (var A: Matrix; Msize: integer; var poly: polynomial);

```

```

    var

```

```

        i, j, k: integer;

```

```

        Nminus1, Nplus1, N: integer;

```

```

sum: extended;

function Constant_Vector (var poly: polynomial; X: extended; N: integer): extended;
  var
    sum, temp: extended;
    i: integer;
begin
  sum := 0;
  if (poly.m[N + 1] <> 0) then
    sum := sum + poly.m[N + 1] * ln(abs(X - poly.coef[N + 1]));
  if (poly.m[N + 2] <> 0) then
    sum := sum + poly.m[N + 2] * ln(abs(X - poly.coef[N + 2]));
  for i := 1 to N do begin
    temp := X - poly.coef[i];
    sum := sum + poly.m[i] * (ln(abs(temp)) + poly.coef[i] / temp);
  end;
  Constant_Vector := Floor(X) + sum;
end;

begin
  Nminus1 := Msize - 1;
  Nplus1 := Msize + 1;
  for i := 1 to Msize do
    for j := 1 to Msize do
      if (j = Nminus1) then
        A[i, Nminus1] := 1
      else if (j = Msize) then
        if (i <= Pass_Freq.total) then
          A[i, Msize] := WBP
        else
          A[i, Msize] := -WBS
      else if (i = 1) then
        A[1, j] := 0
      else if (i = Pass_Freq.total) then
        A[i, j] := -poly.m[j] / poly.coef[j]
      else if (i < Pass_Freq.total) then
        A[i, j] := poly.m[j] / (Pass_Freq.Xi[i] - poly.coef[j])
      else
        A[i, j] := poly.m[j] / (Stop_Freq.Xi[i - Pass_Freq.total] - poly.coef[j]);

  N := poly.deg - 2;
  for j := 1 to Msize do
    if (j = 1) then
      A[1, Nplus1] := Pass_Band.mag[0]
    else if (j = Pass_Freq.total) then begin
      sum := poly.m[N + 1] * ln(poly.coef[N + 1]);
      sum := sum + poly.m[N + 2] * ln(poly.coef[N + 2]);
      for i := 1 to N do
        sum := sum + poly.m[i] * (ln(abs(poly.coef[i])) - 1);
      A[j, Nplus1] := Pass_Band.mag[Pass_Band.total] + sum
    end
    else if (j < Pass_Freq.total) then

```



```

    A[j, Nplus1] := Constant_Vector(poly, Pass_Freq.Xi[j], N)
  else begin
    k := j - Pass_Freq.total;
    A[j, Nplus1] := Constant_Vector(poly, Stop_Freq.Xi[k], N)
  end
end;

procedure Matrix_Evaluate (var A: Matrix; size: integer);
  var
    k, row, col, last: integer;
    w: extended;

  procedure Exchange_Row (var A: Matrix; size, k: integer);
    var
      i, j: integer;
      temp: extended;
  begin
    i := k + 1;
    while (i <= size) and (abs(A[i, k]) < tolerance10) do
      i := i + 1;
    if (i <= size) then
      for j := 1 to size + 1 do begin
        temp := A[k, j];
        A[k, j] := A[i, j];
        A[i, j] := temp;
      end
    else
      Print_Error('Matrix cannot be evaluated !', 0, 2);
    end;

  begin (* MatrixEvaluate *)
    last := size + 1;
    for k := 1 to size - 1 do begin
      if abs(A[k, k]) < tolerance10 then
        Exchange_Row(A, size, k);
      for row := k + 1 to size do
        if abs(A[row, k]) > tolerance10 then begin
          w := A[k, k] / A[row, k];
          for col := k + 1 to last do
            A[row, col] := A[k, col] - w * A[row, col];
          A[row, k] := 0;
        end;
      end;

    for k := size downto 2 do begin
      A[k, last] := A[k, last] / A[k, k];
      A[k, k] := 1.0;
      for row := k - 1 downto 1 do begin
        for col := k - 1 downto row do
          A[row, col] := A[row, col] - A[k, col] * A[row, k];
        A[row, last] := A[row, last] - A[k, last] * A[row, k];
        A[row, k] := 0;
      end;
    end;

```

```

    end;
  end;
  A[1, last] := A[1, last] / A[1, 1];
end;

procedure Output_Xi (var Pass_Freq, Stop_Freq: Signal);
  var
    i, last: integer;
begin
  writeln('X"j' : 22, 'X"j' : 36);
  if (Pass_Freq.total < Stop_Freq.total) then
    last := Stop_Freq.total
  else
    last := Pass_Freq.total;
  for i := 1 to last do begin
    write(i : 2);
    if (i <= Pass_Freq.total) then
      write(Pass_Freq.Xi[i] : 30 : 20)
    else
      write(' ' : 30);
    if (i <= Stop_Freq.total) then
      writeln(Stop_Freq.Xi[i] : 35 : 20)
    else
      writeln;
    end;
  writeln;
end;

procedure Output_Poly_X (var poly: polynomial);
  var
    i: integer;
begin
  writeln('Rational function k(x) :');
  writeln('i' : 2, 'xi' : 20, 'mi' : 25);
  writeln('Cx', poly.coef[0] : 30 : 20);
  for i := 1 to poly.deg do
    writeln(i : 2, poly.coef[i] : 30 : 20, poly.m[i] : 15);
  writeln;
end;

procedure Pole_In_Transition (var poly: polynomial; No_real_TZ: boolean);
  const
    difference = 0.001;
  var
    i, first, last: integer;
begin
  first := NBP + 1;
  last := poly.deg - 2;
  if No_real_TZ then
    for i := first to last do
      if (poly.coef[i] > Left_Stop.Xi[Left_Stop.total])
        and (poly.coef[i] < Right_Stop.Xi[0]) then begin

```

```

    if ((poly.coef[i] - Left_Stop.Xi[Left_Stop.total]) < (Right_Stop.Xi[0] - poly.coef[i]))
    then
        poly.coef[i] := Left_Stop.Xi[Left_Stop.total] - difference
    else
        poly.coef[i] := Right_Stop.Xi[0] + difference;
        writeln('One pole is found in the gap between the Stopbands.');
```

```

    end;

if (poly.coef[first] < Left_Stop.Xi[0]) then begin
    poly.coef[first] := Left_Stop.Xi[0] + difference;
    writeln('One pole is found in the Right Transition Band.');
```

```

    end;
if (poly.coef[last] > Right_Stop.Xi[Right_Stop.total]) then begin
    poly.coef[last] := Right_Stop.Xi[Right_Stop.total] - difference;
    writeln('One pole is found in the Left Transition Band.');
```

```

    end;
end;

procedure Back_Transform (var Xpoly, Ypoly: polynomial; t1, t2: extended;
                          var negative_Zero: boolean);

    var
        i, Mo: integer;
        diff, Cy: extended;
begin
    Ypoly := Xpoly;
    Ypoly.deg := Xpoly.deg - 2;
    Mo := Xpoly.m[Xpoly.deg];
    diff := t2 - t1;
    Cy := 1;
    negative_Zero := FALSE;
for i := 1 to Ypoly.deg do begin
    Cy := Cy * exponential(diff / (Xpoly.coef[i] - 1), Xpoly.m[i]);
    Ypoly.coef[i] := (t1 * Xpoly.coef[i] - t2) / (Xpoly.coef[i] - 1);
    if not negative_Zero and (Ypoly.coef[i] < 0) then
        negative_Zero := TRUE;
    end;
    Ypoly.coef[0] := Ypoly.coef[0] * exp(Mo * ln(t1)) * Cy;
if (Mo < 0) then begin
    Ypoly.deg := Ypoly.deg + 1;
    Ypoly.m[Ypoly.deg] := Mo;
    Ypoly.coef[Ypoly.deg] := 0;
    end;
end;

procedure Output_Optimal (var Xpoly, Ypoly: polynomial; negative_Zero: boolean;
                          Iteration: integer);

    var
        N, i: integer;
begin
    writeln;
    writeln('Optimal Rational function found in ', Iteration : 1, 'th iteration:');
    writeln;

```

```

write('i' : 2, 'Xi' : 18, 'mi' : 15, 'Yi' : 18);
if negative_Zero then
  writeln
else
  writeln('fi' : 25);
with Ypoly do begin
  write('C' : 2, Xpoly.coef[0] : 27 : 20, coef[0] : 32 : 16);
  if negative_Zero then
    writeln
  else
    writeln(sqrt(coef[0]) : 27 : 20);
  write(0 : 2, Xpoly.coef[deg] : 27 : 20, m[deg] : 6, coef[deg] : 26 : 16);
  if negative_Zero then
    writeln
  else
    writeln(sqrt(coef[deg]) / TwoPi : 27 : 20);
  N := deg - 1;
  for i := 1 to N do begin
    write(i : 2, Xpoly.coef[i] : 27 : 20, m[i] : 6, coef[i] : 26 : 16);
    if negative_Zero then
      writeln
    else
      writeln(sqrt(coef[i]) / TwoPi : 27 : 20);
    end;
  end;
end;

(*****          M A I N      P R O G R A M      *****)
begin
DateTime;
SelectDataFile(DataFile, 'Select data file :');
OpenTextWindow;
Read_Signal(DataFile, Left_Stop, Pass_Band, Right_Stop);
Print_Signal(Left_Stop, Pass_Band, Right_Stop, 'fi', 'Alpha(fi) in dB');

lastPt := Right_Stop.Xi[Right_Stop.total];
TwoPi := 2 * Pi;
ln10 := ln(10);
t1 := sqr(TwoPi * Pass_Band.Xi[0]);
t2 := sqr(TwoPi * Pass_Band.Xi[Pass_Band.total]);

Freq_to_X(Pass_Band, Left_Stop, Right_Stop);
Print_Signal(Left_Stop, Pass_Band, Right_Stop, 'Xi', 'β(Xi)');
Input_Initial(DataFile, Xpoly, NBP, NBS, WBP, WBS, No_real_TZ);
close(DataFile);

Pass_Freq.total := NBP + 1;
Stop_Freq.total := NBS + 1;
Matrix_Size := NBP + NBS + 2;
Nplus1 := Matrix_Size + 1;
Nminus1 := Matrix_Size - 1;
N := Xpoly.deg - 2;

```

```

c := (3.0 - sqrt(5.0)) / 2.0;
r := 1.0 - c;

Positive_Zero := FALSE;
Negative_Pole := FALSE;
Not_in_Order := FALSE;
Optimal_Sought := FALSE;
Optimal_Least_Distance := INF;
Optimal_Index := 0;
Best_Iteration := -1;
k := 0;
GetDateTime(StartSec);
repeat
  k := k + 1;
  Old_poly := Xpoly;
  Locate_Abscissa(Xpoly, Pass_Freq, Stop_Freq);
  Least_Distance := Distance(Xpoly, Pass_Freq, Stop_Freq, Above_Pass, Under_Stop);
  write('Least distance = ', Least_Distance : 1 : 20);

  if not (Above_Pass or Under_Stop) and (Least_Distance < Optimal_Least_Distance)
    or (Least_Distance < tolerance10) then begin
    Optimal_Least_Distance := Least_Distance;
    Optimal_Xpoly := Xpoly;
    Optimal_Index := 0;
    Best_Iteration := k - 1;
    writeln('*');
    if (Optimal_Least_Distance < tolerance8) then
      Optimal_Sought := TRUE;
    end
  else begin
    writeln;
    Optimal_Index := Optimal_Index + 1;
  end;

  if ((Best_Iteration = -1) and (k > Initial_Limit)) or ((Best_Iteration > -1)
    and (Optimal_Index > Search_Limit)) then
    Optimal_Sought := TRUE;

  if not Optimal_Sought then begin
    Create_Matrix(A, Matrix_Size, Xpoly);
    Matrix_Evaluate(A, Matrix_Size);
    i := 1;
    while (ABS(A[i, Nplus1] - Old_poly.coef[i]) < tolerance8) and (i <= N) do
      i := i + 1;
    if (i > N) then
      Optimal_Sought := TRUE;
    end;

  if not Optimal_Sought then begin
    writeln;
    writeln('----- Iteration # ', k : 1, ' -----');
    Output_Xi(Pass_Freq, Stop_Freq);

```

```

for i := 1 to N do begin
  Xpoly.coef[i] := A[i, Nplus1];
  if (i <= NBP) and (A[i, Nplus1] > 0) then
    Positive_Zero := TRUE
  else if (i > NBP) and (A[i, Nplus1] < 0) and (i <= N) then
    Negative_Pole := TRUE
  else if (i < 1) and (Xpoly.coef[i - 1] > Xpoly.coef[i]) then
    Not_in_Order := TRUE;
  end;
  Pole_In_Transition(Xpoly, No_real_TZ);
  Xpoly.coef[0] := exp(A[Nminus1, Nplus1]);
  Output_Poly_X(Xpoly);
end;
until Optimal_Sought or Positive_Zero or Negative_Pole or Not_in_Order;
GetDateTime(EndSec);

if Positive_Zero or Negative_Pole or Not_in_Order then
  if Positive_Zero then
    Print_Error('An AZ appears on the positive side of x-axis.', 0, 3)
  else if Negative_Pole then
    Print_Error('A TZ appears on the negative side of x-axis.', 0, 3)
  else
    Print_Error('Poles & Zeros are not in ascending order.', 0, 3)
else if (Best_Iteration = -1) then
  Print_Error('No transfer function is found.', 0, 3)
else begin
  writeln;
  writeln('----- Optimal Transfer Function -----');
  writeln;
  Secs2Date(EndSec - StartSec, DateRec);
  with DateRec do
    writeln('Total time used = ', hour : 2, ':', minute : 2, ':', second : 2);
  Back_Transform(Optimal_Xpoly, Ypoly, t1, t2, negative_Zero);
  StartPt := round(Optimal_Xpoly.coef[1] * 1.2);
  EndPt := round(Optimal_Xpoly.coef[N] * 1.2);
  Plot_Signal(Optimal_Xpoly, StartPt, EndPt, inBeta, 'Optimal  $\beta(x)$ ');

  Output_Optimal(Optimal_Xpoly, Ypoly, negative_Zero, Best_Iteration);
  if negative_Zero then
    writeln('Rational function contains at least ONE real Transmission Zero !')
  else
    Plot_Signal(Ypoly, 0, lastPt, indB, 'ff*/gg*');
  writeln;
  writeln('----- End of Processing -----');
  end;
end.

```

References

- [1] J. A. C. Bingham, "The approximation problem for both conventional and parametric band-pass filters," *IEEE Trans. Circuit Theory (Correspondence)*, vol. CT-11, pp. 408–410, September 1964.
- [2] B. R. Smith and G. C. Temes, "An iterative approximation procedure for automatic filter synthesis," *IEEE Trans. Circuit Theory*, vol. CT-12, no. 2, pp. 107–112, March 1965.
- [3] R. W. Daniels, *Approximation Methods for Electronic Filter Design*. New York: McGraw-Hill, 1974.
- [4] J. Vlach, *Computerized Approximation and Synthesis of Linear Networks*. New York: John Wiley, 1969.
- [5] E. Cohen, "A general class of optimal polynomials related to electric filters," *J. Inst. Maths. Applics.*, vol. 24, pp. 197–208, 1979.
- [6] J. R. Rice, "The approximation of functions," *Linear Theory*. Reading, Mass.: Addison-Wesley, 1964.
- [7] E. Cohen, "General rational bandpass filter functions," *27th Midwest Symp. on Circuits and Systems*, vol II, pp. 462–465, June 1984.
- [8] H. Dubois and H. Leich, "On the approximation problem for recursive digital filters with arbitrary attenuation curve in the pass-band and the stop-band," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-23, pp. 202–207, April 1975.
- [9] L. R. Rabiner, N. Y. Graham, and H. D. Helms, "Linear programming design of IIR digital filters with arbitrary magnitude function," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 117–123, April 1974.

- [10] M. T. Dolan, "Comments on 'On the approximation problem for recursive digital filters with arbitrary attenuation curve in the pass-band and the stop-band,'" *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 575–577, December 1976.
- [11] K. Steiglitz, "Computer-aided design of recursive digital filters," *IEEE Trans. AU*, vol. AU-18, pp. 123–129, June 1970.
- [12] E. Devleeschouwer and F. Grenez, "An effective procedure for the design of a large class of analog and digital filters," *IEEE Trans. Circuits and Systems — II: Analog and digital Signal Processing*, vol. 39, no. 1, pp. 65–69, January 1992.
- [13] E. Remez, "Sur le calcul effectif des polynomes de Tchebichef," *Compt. rend. Acad. Sci.*, Paris, France, vol. 199, pp. 337–340, July 1934.
- [14] J. M. H. Williamson, "Transfer matrix factorization using product-form polynomial scattering parameters," *M.Sc. Thesis*, University of Manitoba, Winnipeg, Canada, 1987.
- [15] A. Fettweis, "Cascade synthesis of lossless two-ports by transfer matrix factorization," Boite, R. (Ed): *Network Theory*. New York: Gordon and Breach Science Publishers, pp.43–103, 1972.
- [16] A. Fettweis, "Factorization of transfer matrices of lossless two-ports," *IEEE Trans. Circuit Theory*, vol. CT-17, pp. 86–94, February 1970.
- [17] V. Belevitch, *Classical Network Theory*. San Francisco: Holden-Day, 1968.
- [18] M. Hasler and J. Neiryneck, *Electric Filters*. Artech House, 1986.
- [19] J. D. Lipson, *Elements of Algebra and Algebraic Computing*. Reading, Mass.: Addison-Wesley, 1981.
- [20] K. Chan, "Cascade synthesis of real and complex lossless two-port networks using transfer scattering matrix factorization," *M. Sc. Thesis*, University of Manitoba, Winnipeg, Canada, 1989.

-
- [21] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*. New York: Cambridge University Press, 1986.
- [22] W. Hohneker, H. Kicherer, R. Unbehauen, and A. Wüpper, "Analog circuit design in the time and frequency domains — Approximation problems," *ntz Archiv* 6, pp. 71–76, 1984.
- [23] T. Fujisawa, "Optimization of lowpass attenuation characteristics by a digital computer," *Proc. 6th Midwest Symp. on Circuit Theory*, pp. P1–P13, May 1963.
- [24] H. J. Orchard and G. C. Temes, "Filter design using transformed variables," *IEEE Trans. Circuit Theory*, vol. CT-15, pp. 385–408, December 1968.
- [25] J. A. C. Bingham, "A new method of solving the accuracy problem in filter design," *IEEE Trans. Circuit Theory*, vol. CT-11, pp. 327–341, September 1964.
- [26] G. C. Temes, "Filter design in transformed variables," in *Computer Oriented Circuit Design*, F. F. Kuo and W. Magnuson, Eds. Englewood Cliffs, N. J.: Prentice-Hall, 1968.
- [27] Randall K. Smith, "Time domain simulation and modelling of wave digital filters," *M. Sc. Thesis*, University of Manitoba, Winnipeg, Canada, 1990.