1-1-1987

A Z-domain design technique for inherently stable, causal, recursive, 2-dimensional digital filters.

Timothy John Kent
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
Electronic Theses and Dissertations. 6816.
https://scholar.uwindsor.ca/etd/6816
A Z-DOMAIN DESIGN TECHNIQUE FOR
INHERENTLY STABLE, CAUSAL, RECURSIVE,
2-DIMENSIONAL DIGITAL FILTERS

by

Timothy John Kent

A Thesis
submitted to the
Faculty of Graduate Studies and Research
through the Department of
Electrical Engineering in Partial Fulfillment
of the requirements for the Degree
of Master of Applied Science at
the University of Windsor

Windsor, Ontario, Canada
1987

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction. In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform EC54805
Copyright 2010 by ProQuest LLC
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
This thesis presents the development of various direct design techniques for inherently stable, causal, recursive, 2-dimensional digital filters whose responses imitate those of 1-dimensional analog filters. These techniques mainly center around the Chebyshev function and provide for the design of low-pass, band-pass and high-pass filters. Alternatively, an approach using the Butterworth approximation is also presented, for the design of low-pass filters. In the case of each design method presented, specific filter examples are given and design parameters are analyzed.

Application of the presented filter design techniques in image processing is also studied. A computationally efficient direct form implementation is used for numerous example images.

In addition, program source code for computer generation of a desired filter's response or transfer function is also included.
To my parents,
Muriel and John Kent
ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Dr. J. J. Solis for his guidance and patience over the course of this research. Also, thanks are extended to my family and friends for their never ending moral support throughout my academic studies. In addition, the financial support of the Natural Sciences and Engineering Research Council of Canada is greatly appreciated. Thanks are also extended to Mr. Russ Lency and Mr. Arnold DeRoy of the Media Department at the Windsor Board of Education, for the use of computer and printing equipment which made this manuscript possible.
TABLE OF CONTENTS

ABSTRACT .................................................................................................................................................................iv
DEDICATION ............................................................................................................................................................v
ACKNOWLEDGEMENTS ........................................................................................................................................vi
LIST OF TABLES ...................................................................................................................................................ix
LIST OF FIGURES ..................................................................................................................................................x

I. INTRODUCTION ...............................................................................................................................................1
   A. 2-Dimensional Digital Filters ........................................................  1
   B. Study of 2-Dimensional Recursive Digital Filters ........................................... 3
   C. 2-Dimensional Recursive Digital Filter Design Techniques .................................................4
   D. Thesis Organization .................................................................................................................................5

II. FIRST CHEBYSHEV LOW-PASS 2-DIMENSIONAL DIGITAL FILTER .............................6
   A. Filter Development ..................................................................................................................................6
   B. Design Examples ....................................................................................................................................12

III. SECOND AND THIRD CHEBYSHEV LOW-PASS 2-DIMENSIONAL DIGITAL FILTERS ......................................................................................................................................17
   A. Second Chebyshev Filter Development .........................................................................................17
   B. Third Chebyshev Filter Development ............................................................................................19
   C. Design Examples ....................................................................................................................................20

IV. BUTTERWORTH LOW-PASS 2-DIMENSIONAL DIGITAL FILTER ....................................27
   A. Filter Development ................................................................................................................................27
   B. Design Examples ....................................................................................................................................29

V. CHEBYSHEV HIGH-PASS 2-DIMENSIONAL DIGITAL FILTER ...................................34
   A. Filter Development ................................................................................................................................34
   B. Design Examples ....................................................................................................................................37

VI. GENERAL CHEBYSHEV 2-DIMENSIONAL DIGITAL FILTER ...........................................42

-reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Examples of First Chebyshev Filters</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Examples of Second and Third Chebyshev Filters</td>
<td>26</td>
</tr>
<tr>
<td>4.1</td>
<td>Examples of Butterworth Filters</td>
<td>33</td>
</tr>
<tr>
<td>5.1</td>
<td>Examples of Chebyshev High-Pass Filters</td>
<td>41</td>
</tr>
<tr>
<td>6.1</td>
<td>Examples of General Chebyshev Low-Pass Filters</td>
<td>49</td>
</tr>
<tr>
<td>6.2</td>
<td>Examples of General Chebyshev High-Pass Filters</td>
<td>52</td>
</tr>
<tr>
<td>7.1</td>
<td>Filter Input Parameters for Processed Images</td>
<td>56</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 Magnitude Response for Exact Solution ................................................................. 11
2.2 First Chebyshev Low-Pass Design Method .......................................................... 13
2.3 Example Filter Magnitude Response ...................................................................... 14
2.4 Example Filter Root Trajectories .......................................................................... 14
2.5 Example Filter Group Delay 1 ................................................................................ 15
2.6 Example Filter Group Delay 2 ................................................................................ 15
3.1 Second Chebyshev Low-Pass Design Method ....................................................... 21
3.2 Third Chebyshev Low-Pass Design Method ........................................................ 22
3.3 Example Filter Magnitude Response ...................................................................... 23
3.4 Example Filter Root Trajectories .......................................................................... 23
3.5 Example Filter Group Delay 1 ................................................................................ 24
3.6 Example Filter Group Delay 2 ................................................................................ 24
4.1 Butterworth Low-Pass Design Method .................................................................. 30
4.2 Example Filter Magnitude Response ...................................................................... 31
4.3 Example Filter Root Trajectories .......................................................................... 31
4.4 Example Filter Group Delay 1 ................................................................................ 32
4.5 Example Filter Group Delay 2 ................................................................................ 32
5.1 Chebyshev High-Pass Design Method .................................................................. 38
5.2 Example Filter Magnitude Response ...................................................................... 39
5.3 Example Filter Root Trajectories .......................................................................... 39
5.4 Example Filter Group Delay 1 ................................................................................ 40
5.5 Example Filter Group Delay 2 ................................................................................ 40
6.1 General Chebyshev Design Method ...................................................................... 45
6.2 Example Low-Pass Filter Magnitude Response .................................................... 46
6.3 Example Low-Pass Filter Root Trajectories ............................................................ 46
I. INTRODUCTION

This thesis encompasses the research and development of various design techniques for inherently stable, causal, recursive, 2-dimensional digital filters. The developments stem from techniques proposed by Soltis [1]. First, in this introduction, a brief overview of 2-dimensional filters will be offered, followed by the methods to study these filters, a brief discussion of design techniques, and concluding with a brief overview of the organization of the following chapters.

A. 2-Dimensional Digital Filters

Two dimensional digital filters can be generally classified into two major groups; non-recursive or finite-extent impulse response (FIR) and recursive or infinite-extent impulse response (IIR) filters. The general equations for these filters are given below in their transfer function and difference equation forms [2],

**FIR Filter**

Difference Equation

\[
y(n_1, n_2) = \sum_{k_1} \sum_{k_2} h(k_1, k_2) x(n_1 - k_1, n_2 - k_2)
\]  

(1.1)

Transfer Function

\[
H(z_1, z_2) = \sum_{k_1} \sum_{k_2} h(k_1, k_2) z_1^{-k_1} z_2^{-k_2}
\]  

(1.2)

where,

- \(x(n_1, n_2)\) = 2-dimensional input sequence
- \(y(n_1, n_2)\) = 2-dimensional output sequence
- \(h(k_1, k_2)\) = filter impulse response

**IIR Filter**

Difference Equation

\[
y(n_1, n_2) = \sum_{l_1} \sum_{l_2} a(l_1, l_2) x(n_1 - l_1, n_2 - l_2)
- \sum_{k_1} \sum_{k_2} b(k_1, k_2) y(n_1 - k_1, n_2 - k_2)
\]  

(1.3)

\((k_1, k_2) = (0,0)\)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Transfer Function

\[ H(z_1, z_2) = \sum_{l_1} \sum_{l_2} a(l_1, l_2) z_1^{-l_1} z_2^{-l_2} \]

\[ \sum_{k_1} \sum_{k_2} b(k_1, k_2) z_1^{-k_1} z_2^{-k_2} \]

where,

\[ x(n_1, n_2) = \text{2-dimensional input sequence} \]

\[ y(n_1, n_2) = \text{2-dimensional output sequence} \]

\[ a(l_1, l_2) = \text{filter coefficients} \]

\[ b(k_1, k_2) = \text{filter coefficients} \]

Since the exclusive design of this thesis is the recursive group of filters, it will be discussed further. The computable group of the recursive filters are classed into two major sub-groups; the first quadrant and the non-symmetric half-plane filters [2]. The non-symmetric half-plane filters are causal with respect to one of the digital variables and non-causal with respect to the other variable. The first quadrant filter is a fully causal filter. The general equations for a first quadrant filter are given below [2].

First Quadrant IIR Filter

Difference Equation

\[ y(n_1, n_2) = \sum_{l_1} \sum_{l_2} a(l_1, l_2) x(n_1 - l_1, n_2 - l_2) \]

\[ \sum_{k_1} \sum_{k_2} b(k_1, k_2) y(n_1 - k_1, n_2 - k_2) \]

\[ (k_1, k_2) = (0,0) \]
Transfer Function

\[ H(z_1,z_2) = \sum_{l_1=0}^{L_1} \sum_{l_2=0}^{L_2} a(l_1,l_2) z_1^{l_1} z_2^{l_2} \]

\[ H(z_1,z_2) = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} b(k_1,k_2) z_1^{k_1} z_2^{k_2} \]

\[ \begin{align*}
H(z_1,z_2) & = \sum_{l_1=0}^{L_1} \sum_{l_2=0}^{L_2} a(l_1,l_2) z_1^{l_1} z_2^{l_2} \\
& = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} b(k_1,k_2) z_1^{k_1} z_2^{k_2}
\end{align*} \]

B. Study of 2-Dimensional Recursive Digital Filters

The study of this type of filter is separated into two main areas. The first area is the calculation of the filter's magnitude, phase and group delay responses. The second area is the stability study and can involve various techniques for determining relative stability.

The magnitude, phase and group delay responses of a 2-dimensional digital filter, outlined in (1.5) or (1.6), can be calculated as follows [2],

\[ H(\omega_1 T,\omega_2 T) = H(z_1,z_2) \big|_{z_1=e^{j\omega T}} \]

\[ = |H(\omega_1 T,\omega_2 T)| \ \phi(\omega_1 T,\omega_2 T) \]

where,

\[ |H(\omega_1 T,\omega_2 T)| = \text{magnitude response} \]

\[ \phi(\omega_1 T,\omega_2 T) = \text{phase response} \]

\[ \omega_1, \omega_2 = \text{frequency variables} \]

\[ T = \text{sampling interval} \]

\[ -\pi < \omega_1 T < \pi \]

and,

\[ \tau_i(\omega_1 T,\omega_2 T) = \frac{\delta \phi(\omega_1 T,\omega_2 T)}{\delta \omega T} \]

where,

\[ \tau_i(\omega_1 T,\omega_2 T) = \text{group delay} \]

\[ i = 1, 2 \]
These responses can be visualized as surfaces over the digital spectral plane.

The stability of a filter can be determined by numerous techniques. One technique based on the root maps of the filter's denominator polynomial is stated as follows [3],

Let
\[ H(z_1, z_2) = \frac{1}{B(z_1, z_2)} \]  

be a first quadrant recursive filter.

This filter is stable if and only if the following is true.

1) \( B(z_1, z_2) \neq 0 \) for \( |z_1| = 1 \) and \( |z_2| = 1 \)

2) \( B(a, z_2) \neq 0 \) for \( |z_2| > 1 \) for any \( a \) such that \( |a| = 1 \)

3) \( B(z_1, b) \neq 0 \) for \( |z_1| > 1 \) for any \( b \) such that \( |b| = 1 \)

The physical realization of this theorem is that singularities of the denominator polynomial have a magnitude less than one when one of the \( z \)-variables traverses the complex unit magnitude circle.

In addition to this theorem it is also possible to ascertain stability by generating the filter's impulse response and testing it for a finite convergence to zero. The impulse response of a filter of this type can be calculated using the following equation [2],

\[ h(n_1, n_2) = a_{00} \delta(n_1, n_2) + \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} b_{ij} y(n_1-i, n_2-j) \]  

(1.10)

Throughout this research, the previously stated equations and theorems were used to study the filters designed.

C. 2-Dimensional Recursive Digital Filter Design Techniques

As stated earlier, there are two different types of digital filters; non-recursive and recursive. Although non-recursive filters sometimes lend themselves to a more efficient application [4], recursive filters are generally less complex to realize [2]. Unlike non-recursive filters, a guarantee of a linear phase response is difficult to insure when designing recursive filters [2]. Since this is essential in image processing [5], then attaining a linear phase response when using recursive systems in the
filter's passband is paramount.

The major drawback to recursive filters has been their computationally intensive design techniques. Most existing design techniques center around either an iterative optimization technique [2,6-9], which cannot guarantee stability, or transformation methods [2,10,11].

In contrast, the direct design techniques, developed by this study, provide for inherently stable filters. The methods presented in this study employ the development of multi-dimensional, z-domain mathematical arguments for use in the Chebyshev or Butterworth functions to yield filter transfer functions.

D. Thesis Organization

In the following chapters each of the six design techniques' development will be illustrated and design examples given. Following the filters' presentation, their application in image processing will be examined, with various example images.

The program languages used for development in this study are exclusively Turbo PASCAL for the IBM PC programs and VAX FORTRAN for the DEC VAX 11-750/85 programs. The source code listing for plotting routines on the IBM PC, and impulse generation and plotting using VT241 terminals on a DEC VAX computer are shown in Appendix 1.
II. FIRST CHEBYSHEV LOW-PASS 2-DIMENSIONAL DIGITAL FILTER

In this chapter the first Chebyshev low-pass 2-dimensional digital filter's development and design examples will be presented.

A. Filter Development

A generalized form for the transfer function of a 2-dimensional IIR filter with no numerator singularities can be stated as follows,

\[ H(z_1, z_2) = \frac{1}{B(z_1, z_2)} \]  

(2.1)

where,

\[ B(z_1, z_2) = \text{finite order polynomial in } z_1, z_2 \]

By taking the square of the magnitude response of the above function and expressing it in terms of the Chebyshev function, the following can be attained,

\[ |H(e^{j\omega_1T}, e^{j\omega_2T})|^2 = \frac{1}{1 + (\varepsilon C_n(\text{Arg}))^2} \]

(2.2)

where,

\[ \varepsilon = (10^{-1.0 A_{max}})^{0.5} \]

(2.3)

\[ A_{max} = \text{Passband Ripple [dB]} \]

\[ C_n(\text{Arg}) = \cos(n \cos(\text{Arg})) \]

(2.4)

\[ n = \text{filter order} \]

\[ \text{Arg} = \text{function in } \cos(\omega_1T), \cos(\omega_2T) \]

The development of a 2-dimensional digital Chebyshev argument, Arg, must now be undertaken. By applying the appropriate restrictions on the Chebyshev function argument the following can be stated,

\[ \text{Arg} = f(\cos(\omega_1T), \cos(\omega_2T)) \]
such that $|\text{Arg}| < 1.0$ in the filter's passband

$|\text{Arg}| > 1.0$ in the filter's stopband

Consider the following relationship in the two digital spectral variables,

$$f(\omega_1 T, \omega_2 T) = \left(\omega_1 T\right)^2 + \left(\omega_2 T\right)^2$$

$$\frac{2}{(\gamma \pi)}$$

(2.5)

This relationship is a circularly symmetric function centered at,

$$\omega_1 T = \omega_2 T = 0.0$$

where,

$$|f(\omega_1 T, \omega_2 T)| = 1.0 \text{ for } (\omega_1 T)^2 + (\omega_2 T)^2 \leq (\gamma \pi)$$

representing the passband region and,

$$|f(\omega_1 T, \omega_2 T)| > 1.0 \text{ for } (\omega_1 T)^2 + (\omega_2 T)^2 > (\gamma \pi)$$

representing the stopband region, where,

$$\gamma = \text{normalized passband size parameter}$$

$$0.0 < \gamma \leq 1.0$$

The Taylor Series for the cosine function can be stated as follows,

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ...$$

(2.6)

or approximately,

$$x^2 \cong 2 - 2 \cos(x)$$

(2.7)

By using the result of (2.7) in equation (2.5), the following can be derived,

$$f(\cos(\omega_1 T), \cos(\omega_2 T)) = 2 - \cos(\omega_1 T) - \cos(\omega_2 T)$$

$$\frac{1}{1 - \cos(\gamma \pi)}$$

(2.8)

By making a further trigonometric substitution, (2.8) can be equivalently stated as follows,
\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = 2 - \cos(\omega_1 T) - \cos(\omega_2 T) \]  
\[ \frac{2}{2 \sin \left( \frac{\gamma \pi}{2} \right)} \]  

This function represents a suitable Chebyshev argument in the variables \( \cos(\omega_1 T) \) and \( \cos(\omega_2 T) \). For solution of the roots for the earlier stated transfer function, (2.2), it is necessary to define the Chebyshev argument as a function of the two z-domain variables. The following relationship provides this transformation,

\[ \cos(\omega_1 T) = \frac{z_1 + z_{-1}}{2} \]  

(2.10)

Substituting (2.10) into (2.9) an appropriate function in the z-domain variables is attained.

\[ \text{Arg} = 4 - z_1 - z_{-1} - z_2 - z_{-2} \]  
\[ \frac{2}{4 \sin \left( \frac{\gamma \pi}{2} \right)} \]  

(2.11)

It is now necessary to find the denominator roots of the transfer function stated in (2.2). Setting that expression equal to zero yields the following,

\[ 1 + (e^{C_2 \text{Arg}}) = 0 \]  

(2.12)

Letting,

\[ \cos^{-1}(\text{Arg}) = W = u + j v \]

produces the following,

\[ 1 + (e^{\cos(n(u + j v))}) = 0 \]  

(2.13)

or,

\[ \cos(n(u + j v)) = \frac{\pm j}{\epsilon} \]  

(2.14)
Expanding the complex valued cosine function into its real and imaginary parts yields the following,

\[ \cos(n \ u) \cosh(n \ v) - j \sin(n \ u) \sinh(n \ v) = + - j \epsilon \]  

(2.15)

By studying the real and imaginary parts of (2.15) separately the roots of the polynomial can be ascertained.

\[ \cos(n \ u) \cosh(n \ v) = 0 \]  

(2.16)

Since,

\[ \cosh(x) \leq 1.0 \text{ for any } x \]

Therefore,

\[ \cos(n \ u) = 0 \]

\[ n \ u = (2k + 1) \pi \text{ for } |k| = 0,1,2,... \]

\[ \frac{u_k}{2} = (2k + 1) \pi \text{ for } k = 0,1,2,... \frac{2n-1}{2n} \]  

(2.17)

Considering the imaginary part,

\[ \sin(n \ u) \sinh(n \ v) = + - 1 \]

(2.18)

Since,

\[ \sin(n \ u_k) = + - 1.0 \text{ for all } k \]

Therefore,

\[ \sinh(n \ v) = + - 1 \]

\[ \frac{v_k}{\epsilon} = \sinh \left( \frac{\epsilon}{n} \right) \text{ for all } k \]

(2.19)

Therefore, (2.11) and (2.12) lead to the following,
\[
\text{Arg} = \cos(W_k) \\
4 - z_1 - z_1 - z_2 - z_2 = \cos(W_k)
\]

From these roots the transfer function denominator polynomial must be developed. A direct solution for the \(z\)-domain variables in (2.21), to yield the polynomial factors, leads to an unrealizable filter. Although this filter is unrealizable, its responses can be generated. The magnitude response of an example is given in Figure 2.1 (computer program source code given in Appendix 2). In this figure, a circularly symmetric passband is present with ripple and large attenuation in the stopband. An alternate solution must be derived to attain a realizable filter.

Letting,

\[
z_j = e^{\xi_j}
\]

for the following expression,

\[
(z_1 + z_1) + (z_2 + z_2)
\]

yields,

\[
(z_1 + z_1) + (z_2 + z_2) = e^{\xi_1} + e^{\xi_1} + e^{\xi_2} + e^{\xi_2}
\]

Letting,

\[
\xi_1 = \xi_2 = \xi
\]

ensures that the solution is exact along \(\omega_1T = \omega_2T\).

Using this relation in (2.22) yields the following,

\[
(z_1 + z_1) + (z_2 + z_2) = 2(e^{\xi} + e^{-\xi})
\]

Using the result of (2.23) to solve for the other terms in (2.21) yields the following values for \(\xi\).

\[
\xi_k = \cosh \left(1 - \sin \left(\frac{\gamma \pi}{2}\right) \cos(W_k)\right)
\]
Type - Exact  \( n = 4 \)  \( A_{\text{max}} = 0.5 \)  \( \gamma = 0.5 \)

Figure 2.1: Magnitude Response for Exact Solution
It is necessary to represent (2.23) in the form of realizable polynomial factors. Through the filter's development it has been found that the following factors are sufficient,

\[
(z_1 + z_2 -2e)(z_1 + z_2 -2e) = 0
\]  

(2.25)

When studying the stability of these factors it was found that the first factor represented a stable factor and the second an unstable factor. Therefore the filter was constructed from the first factors as follows,

\[
H(z_1, z_2) = \frac{1}{\prod_{k=0}^{2n-1} (z_1 + z_2 -2e)^{-\xi_k}}
\]  

(2.26)

B. Design Examples

The previously illustrated filter design technique can be summarized as shown in Figure 2.2. The computer programs used to study the responses of this filter are given in Appendix 2. Figure 2.3 illustrates the magnitude response of an example filter. This response is clearly low-pass in nature and possesses a well defined passband, with ripple. The root trajectory map of this filter is given in Figure 2.4. This map clearly indicates a stable filter, since all trajectories are enclosed within the unit circle. Although not included in this report, the impulse response of this filter was also generated and converged to zero as expected. The group delays of the filter are also given in Figures 2.5 and 2.6. These delays do possess the necessary flat response in the passband of the filter to minimize signal distortion.

The input parameters to the design technique, as given in Figure 2.2, are Amax, n and γ. The Amax input parameter controls the amount of ripple present in the filter's passband and produces exact results. The passband size parameter, γ, works as expected to vary passband size. The larger the value of γ the larger the passband is in radial size. The filter order, n, also works as anticipated. As the order increases, the transition from passband to stopband becomes sharper and greater attenuation in the stopband is attained. Through examples, Table 2.1 summarizes response characteristics for this filter design method.
First Chebyshev Low-Pass 2-Dimensional Digital Filter Design

\[ H(z_1, z_2) = \frac{1}{B(z_1, z_2)} \]

\[ B(z_1, z_2) = \prod_{k=0}^{2n-1} (z_1 + z_2 - 2e^\xi_k) \]

\[ \xi_k = \cosh \left( 1 - \sin \left( \gamma \pi \right) \cos(W_k) \right) \]

\[ W_k = u_k + j v_k \]

\[ u_k = \left( 2k + 1 \right) \pi \]

\[ v_k = \left( \frac{1}{2n} - 1 \right) \left( \frac{1}{2n} - 1 \right) \]

\[ \epsilon = \begin{cases} 0.1 A_{\text{max}} & 0.5 \\ 10^{-1} & \end{cases} \]

\[ n = \text{filter order} = 0.5, 1, 1.5, 2.0, \ldots \]

\[ \gamma = \text{normalized passband size} \quad 0.0 < \gamma < 1.0 \]

\[ A_{\text{max}} = \text{passband ripple (dB)} \]

Figure 2.2: First Chebyshev Low-Pass Design Method
Type - Cheb \( n = 4 \) \( A_{\text{max}} = 1.0 \) \( \gamma = 0.5 \)

Figure 2.3: Example Filter Magnitude Response

Type - Cheb \( n = 4 \) \( A_{\text{max}} = 1.0 \) \( \gamma = 0.5 \)

Figure 2.4: Example Filter Root Trajectories

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Group Delay 1

![Group Delay 1](image1)

Type - Cheb \( n = 4 \) \( A_{\text{max}} = 1.0 \) \( \gamma = 0.5 \)

Figure 2.5: Example Filter Group Delay 1

Group Delay 2

![Group Delay 2](image2)

Type - Cheb \( n = 4 \) \( A_{\text{max}} = 1.0 \) \( \gamma = 0.5 \)

Figure 2.6: Example Filter Group Delay 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
## Filter Input Parameters

<table>
<thead>
<tr>
<th>n</th>
<th>A&lt;sub&gt;max&lt;/sub&gt;</th>
<th>γ</th>
<th>Radial Passband Size</th>
<th>Maximum Passband Gain</th>
<th>Maximum Stopband Attenuation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[dB]</td>
<td></td>
<td>[rad]</td>
<td>[rad]</td>
<td>[dB]</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.2</td>
<td>0.684</td>
<td>0.720</td>
<td>0.461</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.140</td>
<td>1.440</td>
<td>0.460</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.710</td>
<td>2.160</td>
<td>0.485</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.684</td>
<td>0.720</td>
<td>0.920</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.140</td>
<td>1.390</td>
<td>0.944</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.600</td>
<td>2.020</td>
<td>0.970</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.2</td>
<td>0.509</td>
<td>0.581</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.806</td>
<td>1.280</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.140</td>
<td>1.840</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.456</td>
<td>0.581</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.806</td>
<td>1.250</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.140</td>
<td>1.740</td>
<td>0.000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.2</td>
<td>0.456</td>
<td>0.581</td>
<td>0.493</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.684</td>
<td>1.250</td>
<td>0.500</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.030</td>
<td>1.820</td>
<td>0.480</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.455</td>
<td>0.581</td>
<td>0.986</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.684</td>
<td>1.250</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.010</td>
<td>1.740</td>
<td>0.950</td>
<td></td>
</tr>
</tbody>
</table>

Note - Radial Passband size measured from ω<sub>1</sub> T = ω<sub>2</sub> T = 0.0 to -3 dB point
-Magnitude Response scaled for 0 dB at ω<sub>1</sub> T = ω<sub>2</sub> T = 0

Table 2.1: Examples of First Chebyshev Filters

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
III. SECOND AND THIRD CHEBYSHEV LOW-PASS
2-DIMENSIONAL DIGITAL FILTERS

In the following sections, the second and third Chebyshev low-pass 2-dimensional digital filter's
development with design examples will be offered. An identical development to that which is used
for the first Chebyshev filter is employed, except, that a different Chebyshev argument is derived.
This argument was designed to offer greater dynamic range in the magnitude response of the filter
thereby providing larger attenuation in the stopband.

A. Second Chebyshev Filter Development

The development of another 2-dimensional digital Chebyshev argument, Arg, must be undertaken.
Consider the following function,

\[ f(u_1T, u_2T) = \frac{2 ((\omega_1T)^2 + (\omega_2T)^2) - (\gamma \pi)}{(\gamma \pi)^2} \]  

\( (3.1) \)

where,

\[ \gamma = \text{normalized passband size parameter} \]
\[ 0.0 < \gamma < 1.0 \]

As before, this new relationship is a circularly symmetric function. Unlike the earlier function, it
offers greater dynamic range over the digital spectral domain and, this will translate directly into
larger attenuation in the filter’s stopband.

Using the same Taylor Series approximation for the cosine function as used in the first
development, (3.1) can be expressed as the following,

\[ f(\cos(\omega_1T), \cos(\omega_2T)) = \frac{3 - 2 \cos(\omega_1T) - 2 \cos(\omega_2T) + \cos(\gamma \pi)}{1 - \cos(\gamma \pi)} \]  

\( (3.2) \)

By making a further trigonometric substitution, (3.2) can be equivalently stated as follows,
\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = \]
\[
\frac{2}{2 - \sin (\gamma \pi) - \cos(\omega_1 T) - \cos(\omega_2 T)}
\]
\[
\frac{2}{2 \sin (\gamma \pi)}
\]

(3.3)

This function is a second suitable Chebyshev argument in the variables \( \cos(\omega_1 T) \) and \( \cos(\omega_2 T) \). As before, for solution of the roots for the transfer function it is necessary to represent the Chebyshev argument as a function of the two z-domain variables. By use of (2.10) the following relationship is equivalent to (3.3),

\[ \text{Arg} = \frac{2}{2 - \sin (\gamma \pi) - z_1 - z_1 \cdot z_2 - z_2} \]

(3.4)

It is now necessary to find the denominator roots of the transfer function. Since this is the identical solution to that shown for the first Chebyshev filter, the results will only be stated.

Solving,

\[ 1 + (e C_\theta(\text{Arg})) = 0 \]

(3.5)
yields,

\[ \text{Arg} = \cos(W_k) = \cos(u_k + j v_k) \]

(3.6)

where,

\[ u_k = (2k + 1) \pi \text{ for } k = 0, 1, 2, \ldots 2n - 1 \]

(3.7)

\[ v_k = \sinh (e^{\frac{-1}{n}}) \text{ for all } k \]

(3.8)
As earlier, the transfer function denominator polynomial must be derived from these roots. The development of the first filter is again employed with the following results.

Letting,

\[
\begin{align*}
-1 & -1 - \xi & & -\xi \\
(z_1 + z_1) + (z_2 + z_2) = 2(c + c) 
\end{align*}
\]  
(3.9)

yields,

\[
\xi_k = \cosh \left(1 - \sin \left(\gamma \pi\right) \cos \left(W_k\right)\right)
\]
(3.10)

Using the stable polynomial factors developed earlier the filters transfer function can be stated as follows,

\[
H(z_1,z_2) = \frac{1}{\prod_{k=0}^{2n-1} (z_1 + z_2 - 2e^{-\xi_k})}
\]  
(3.11)

B. Third Chebyshev Filter Development

Following the earlier developments a third Chebyshev argument was studied and is stated as follows,

\[
f(\omega_1T,\omega_2T)) = \frac{2}{(\gamma \pi)} - \frac{2}{2((\omega_1T) + (\omega_2T))}
\]  
(3.12)

This relationship, the negative of the second Chebyshev argument, also represents an acceptable circularly symmetric function. By using the identical Taylor series approximation and trigonometric substitution as earlier, (3.12) can be expressed as follows,

\[
f(\cos(\omega_1T)\cos(\omega_2T)) = \frac{2}{2 \sin (\gamma \pi) - 2 + \cos(\omega_1T) + \cos(\omega_2T)}
\]  
(3.13)
or expressed, using (2.10), in terms of the $z$-domain variables,

$$\text{Arg} = \sin \left( \frac{\gamma \pi}{2} \right) - 2 + z_1 + z_1 + z_2 + z_2$$

$$\frac{-1}{2}$$

$$\frac{2}{2}$$

$$\frac{2 \sin \left( \frac{\gamma \pi}{2} \right)}{2}$$

(3.14)

For this filter, the roots of the denominator polynomial are equivalent to those stated in (3.6) to (3.8). Using these roots and the above Chebyshev argument, (3.14), in the earlier shown polynomial factor development, the following results were attained,

$$\xi_k = \cosh \left( 1 - \sin \left( \frac{\gamma \pi}{2} \right) \cos \left( \frac{W_k}{2} \right) \right)$$

(3.15)

with,

$$H(z_1, z_2) = 1$$

(3.16)

C. Design Examples

The previously illustrated filter design techniques can be summarized as shown in Figures 3.1 and 3.2. The computer programs used to study the responses of these filters are given in Appendix 3 and 4 for the second and third Chebyshev designs, respectively. Since the results of these two techniques are identical only examples for the second Chebyshev filter will be given. Figure 3.3 illustrates the magnitude response of an example filter. As in the previous technique, this response is clearly low-pass in nature and possesses a well defined passband, with ripple. The root trajectory map of this filter is given in Figure 3.4. This map clearly indicates a stable filter. Although not included in this report, the impulse response of this filter was also generated and converged to zero as expected. The group delays of the filter are also given in Figures 3.5 and 3.6. These delays do possess the necessary flat response in the passband of the filter. In comparison to the first Chebyshev filter, the
Second Chebyshev Low-Pass 2-Dimensional Digital Filter Design

\[
H(z_1,z_2) = \frac{1}{B(z_1,z_2)}
\]

\[
B(z_1,z_2) = \prod_{k=0}^{2n-1} (z_1 + z_2 - 2e^{-\xi_k})
\]

\[
\xi_k = \cosh \left( 1 - \sin \left( \gamma \pi \right) \cos \left( W_k \right) \right)^{\frac{1}{2}}
\]

\[
W_k = u_k + j v_k
\]

\[
u_k = \frac{(2k + 1)\pi}{2n}
\]

\[
\epsilon = \frac{0.1 A_{\text{max}}}{0.5}
\]

\[
\epsilon = \frac{10 - 1}{n}
\]

\[
n = \text{filter order} = 0.5, 1, 1.5, 2.0, ...
\]

\[
\gamma = \text{normalized passband size} 0.0 < \gamma < 1.0
\]

\[
A_{\text{max}} = \text{passband ripple (dB)}
\]

Figure 3.1: Second Chebyshev Low-Pass Design Method
Third Chebyshev Low-Pass 2-Dimensional Digital Filter Design

\[
H(z_1,z_2) = \frac{1}{B(z_1,z_2)}
\]

\[
B(z_1,z_2) = \prod_{k=0}^{2n-1} (z_1 + z_2 - 2\epsilon^k)
\]

\[
\xi_k = \cosh (1 - \sin (\gamma \pi) \sin (W_k))
\]

\[
W_k = u_k + j v_k
\]

\[
u_k = \frac{(2k + 1) \pi}{2n}
\]

\[
\epsilon = (10^{-(0.1 \text{ Amax})} - 1)
\]

\[n = \text{filter order} = 0.5, 1, 1.5, 2.0, ...\]

\[\gamma = \text{normalized passband size} 0.0 < \gamma < 1.0\]

\[\text{Amax} = \text{passband ripple (dB)}\]

Figure 3.2: Third Chebyshev Low-Pass Design Method
Magnitude Response (in dB)

Type - Cheb2  $n = 4$  $A_{max} = 1.0$  $\gamma = 0.5$

Figure 3.3: Example Filter Magnitude Response

Root Trajectories

Type - Cheb2  $n = 4$  $A_{max} = 1.0$  $\gamma = 0.5$

Figure 3.4: Example Filter Root Trajectories

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 3.5: Example Filter Group Delay 1

Figure 3.6: Example Filter Group Delay 2
dynamic range of the magnitude response is increased, as expected, but, the group delay maxima has also increased.

The input parameters to the design technique, as given earlier in Figure 3.1 and 3.2, are \( A_{\text{max}} \), \( n \) and \( \gamma \). The input parameters behave in the same manner as those described in the earlier method. Table 3.1 summarizes results attained from this filter design by varying the input parameters.
### Table 3.1: Examples of Second and Third Chebyshev Filters

<table>
<thead>
<tr>
<th>n</th>
<th>A_{max}</th>
<th>\gamma</th>
<th>Radial Passband Size</th>
<th>Maximum Passband Gain</th>
<th>Maximum Stopband Attenuation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Min. [rad]</td>
<td>Max. [rad]</td>
<td>[dB]</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.2</td>
<td>0.581</td>
<td>0.684</td>
<td>0.455</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>1.020</td>
<td>1.300</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>1.370</td>
<td>1.880</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.2</td>
<td>0.581</td>
<td>0.664</td>
<td>0.900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>0.806</td>
<td>1.260</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>1.300</td>
<td>1.840</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.2</td>
<td>0.456</td>
<td>0.581</td>
<td>0.467</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>0.806</td>
<td>1.260</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>1.030</td>
<td>1.740</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.2</td>
<td>0.456</td>
<td>0.581</td>
<td>0.930</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>0.684</td>
<td>1.160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>1.030</td>
<td>1.740</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.2</td>
<td>0.456</td>
<td>0.581</td>
<td>0.417</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>0.664</td>
<td>1.160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>0.868</td>
<td>1.740</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.2</td>
<td>0.456</td>
<td>0.581</td>
<td>0.826</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>0.664</td>
<td>1.160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>0.911</td>
<td>1.710</td>
</tr>
</tbody>
</table>

**Note** - Radial Passband size measured from \(\omega_1 T = \omega_2 T = 0.0\) to -3 dB point.
- Magnitude Response scaled for 0 dB at \(\omega_1 T = \omega_2 T = 0\)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
IV. BUTTERWORTH LOW-PASS 2-DIMENSIONAL DIGITAL FILTER

Now a filter using the Butterworth approximation will be developed and design examples will be given. This filter’s development is similar to that of the three previously discussed filters except that the Butterworth function is employed instead of the Chebyshev.

A. Filter Development

A generalized form for the transfer function of a 2-dimensional recursive filter with no numerator singularities can be stated as follows,

\[
H(z_1, z_2) = \frac{1}{B(z_1, z_2)} \tag{4.1}
\]

where,

\[B(z_1, z_2) = \text{finite order polynomial in } z_1, z_2\]

By taking the square of the magnitude response of the above function and expressing it in terms of the Butterworth approximation the following can be attained,

\[
|\frac{\sin(j\omega_1 T \sqrt{2})}{\sin(j\omega_2 T \sqrt{2})}| = 1 \tag{4.2}
\]

where,

\[n = \text{filter order} \]

\[\text{Arg} = \text{function in } \cos(\omega_1 T), \cos(\omega_2 T)\]

The development of a 2-dimensional digital Butterworth argument, Arg, must now be undertaken. By applying the appropriate restrictions on the Butterworth function argument the following can be stated,

\[\text{Arg} = f(\cos(\omega_1 T), \cos(\omega_2 T))\]

such that

\[|\text{Arg}| \leq 1.0 \text{ in the filter’s passband}\]

\[|\text{Arg}| > 1.0 \text{ in the filter’s stopband}\]
By considering the first Chebyshev filter's argument for use with this filter, and employing the same development, a Butterworth argument can be expressed as follows,

\[
\text{Arg} = \frac{4 - z_1 - z_1 - z_2 - z_2}{4 \sin (\gamma \pi)}
\]  

(4.3)

It is now necessary to find the denominator roots of the transfer function stated in (4.2). Setting that expression equal to zero yields the following,

\[
2n
\]

\[
1 + (\text{Arg}) = 0
\]

(4.4)

Letting,

\[
\text{Arg} = W = u + j \, v
\]

produces the following,

\[
2n
\]

\[
1 + (u + j \, v) = 0
\]

(4.5)

or,

\[
2n
\]

\[
(u + j \, v)^{-1} = -1
\]

(4.6)

Solving this equation for the appropriate roots of unity yields the following solution,

\[
u_k = \cos((2k + 1) \pi) \text{ for } k = 0, 1, 2, \ldots, 2n - 1
\]

(4.7)

\[
v_k = \sin((2k + 1) \pi) \text{ for } k = 0, 1, 2, \ldots, 2n - 1
\]

(4.8)

Therefore,

\[
\text{Arg} = W_k
\]

(4.9)

\[
\frac{4 - z_1 - z_1 - z_2 - z_2}{4 \sin (\gamma \pi)} = W_k
\]

(4.10)
From these roots, the transfer function denominator polynomial must be developed. The same denominator polynomial factors, as before, are employed to realize the following results,

$$\xi_k = \cosh \left( \frac{1 - \sin (\gamma \pi) W_k}{2} \right)$$

$$H(z_1, z_2) = \frac{1}{\prod_{k=0}^{2n-1} (z_1 + z_2 - 2e^{-\xi_k})}$$

with,

B. Design Examples

The previously illustrated filter design technique can be summarized as shown in Figure 4.1. The computer programs used to study the responses of this filter are given in Appendix 5. Figure 4.2 illustrates the magnitude response of an example filter. This response is clearly low-pass in nature and possesses a well-defined passband. The root trajectory map of this filter is given in Figure 4.3. This map clearly indicates a stable filter. Although not included in this report, the impulse response of this filter was also generated and converged to zero as expected. The group delays of the filter are also given in Figures 4.4 and 4.5. These delays do possess the necessary flat response in the passband of the filter. In comparison to the earlier shown Chebyshev filters, this filter does not possess as large a dynamic range in the magnitude response. It does, however, possess a more desirable group delay.

The input parameters to the design technique, as given earlier in Figure 4.1, are $n$ and $\gamma$. These parameters behave in an applicable manner as those described earlier. Through examples, Table 4.1 documents the response characteristics of filters designed by this method.
Butterworth Low-Pass 2-Dimensional Digital Filter Design

\[
H(z_1, z_2) = \frac{1}{B(z_1, z_2)}
\]

\[
B(z_1, z_2) = \prod_{k=0}^{2n-1} (z_1 + z_2 - 2e^{-\xi_k})
\]

\[
\xi_k = \cosh \left( \left( 1 - \sin \left( \gamma \pi \right) \right) W_k \right) - 1
\]

\[
W_k = u_k + j v_k
\]

\[
u_k = \cos \left( \left( 2k + 1 \right) \pi \right)\]

\[
v_k = \sin \left( \left( 2k + 1 \right) \pi \right)
\]

\[
n = \text{filter order} = 0.5, 1, 1.5, 2, \ldots
\]

\[
\gamma = \text{normalized passband size } 0.0 < \gamma < 1.0
\]

Figure 4.1: Butterworth Low-Pass Design Method
Figure 4.2: Example Filter Magnitude Response

Figure 4.3: Example Filter Root Trajectories

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 4.4: Example Filter Group Delay 1

Figure 4.5: Example Filter Group Delay 2
<table>
<thead>
<tr>
<th>n</th>
<th>( \gamma )</th>
<th>Radial Passband Size Min. [rad]</th>
<th>Radial Passband Size Max. [rad]</th>
<th>Maximum Stopband Attenuation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.2</td>
<td>0.805</td>
<td>0.867</td>
<td>40.8</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.440</td>
<td>1.740</td>
<td>18.5</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.400</td>
<td>3.090</td>
<td>8.10</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
<td>0.684</td>
<td>0.868</td>
<td>61.2</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.260</td>
<td>1.740</td>
<td>27.7</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>2.170</td>
<td>2.700</td>
<td>11.4</td>
</tr>
<tr>
<td>4</td>
<td>0.2</td>
<td>0.581</td>
<td>0.801</td>
<td>81.6</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1.140</td>
<td>1.740</td>
<td>36.3</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>1.940</td>
<td>2.640</td>
<td>14.9</td>
</tr>
</tbody>
</table>

Note - Radial Passband size measured from \( \omega_1 T = \omega_2 T = 0 \) to -3 dB point
-Magnitude Response scaled for 0 dB at \( \omega_1 T = \omega_2 T = 0 \)

Table 4.1: Examples of Butterworth Low-Pass Filters
V. CHEBYSHEV HIGH-PASS 2-DIMENSIONAL DIGITAL FILTER

Following the previous design techniques, a Chebyshev high-pass 2-dimensional digital filter, with examples, will now be introduced. This filter follows a similar development to that discussed in the earlier chapters, except, that a new Chebyshev argument will be developed.

A. Filter Development

The development of another 2-dimensional digital Chebyshev argument, Arg, must now be undertaken. In the case of developing a high-pass Chebyshev argument, it is necessary to satisfy different constraints with respect to the magnitude of the argument. These constraints are as follows,

\[ |f(\cos(\omega_1 T), \cos(\omega_2 T))| \leq 1.0 \text{ for } (\omega_1 T) + (\omega_2 T) \geq (\gamma \pi) \]

representing the passband region and,

\[ |f(\cos(\omega_1 T), \cos(\omega_2 T))| > 1.0 \text{ for } (\omega_1 T) + (\omega_2 T) < (\gamma \pi) \]

representing the stopband region, where,

\[ \gamma = \text{normalized stopband size parameter} \]
\[ 0.0 < \gamma < 1.0 \]

Consider the following relationship,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = \alpha (\cos(\omega_1 T) + \cos(\omega_2 T)) + \beta \]  \hspace{1cm} (5.1)

where,

\[ \alpha, \beta = \text{scaling parameters evaluated to satisfy Chebyshev argument requirements} \]

With the following constraint,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = 1.0 \text{ for } \omega_1 T = \omega_2 T = \pi \]

in (5.1) leads to,

\[ \beta = 2\alpha + 1 \]  \hspace{1cm} (5.2)

In addition,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = -1.0 \text{ for } \omega_1 T = \gamma \pi, \omega_2 T = 0.0 \]

in (5.1) leads to,
\[ \alpha (\cos(\gamma \pi) + 1) + \beta = -1 \quad (5.3) \]

Solving for the scaling parameters in (5.2) and (5.3) gives the following,

\[ \alpha = \frac{-2}{3 + \cos(\gamma \pi)} \quad (5.4) \]
\[ \beta = \frac{1 - 4}{3 + \cos(\gamma \pi)} \quad (5.5) \]

By using the result of (5.4) and (5.5) in (5.1), the following function can be stated,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = \frac{\cos(\gamma \pi) - 1 - 2 (\cos(\omega_1 T) + \cos(\omega_2 T))}{3 + \cos(\gamma \pi)} \quad (5.6) \]

This function represents a suitable Chebyshev argument in the variables \( \cos(\omega_1 T) \) and \( \cos(\omega_2 T) \).

By using the results of (2.10) this equation can be expressed in terms of the \( z \)-domain variables as follows,

\[ \text{Arg} = \frac{-1}{3 + \cos(\gamma \pi)} \quad (5.7) \]

It is now necessary to find the denominator roots of the transfer function. This is an identical solution to that shown earlier, so only the results will be summarized.

Solving,

\[ 1 + (\epsilon \ C_n(\text{Arg})) = 0 \quad (5.8) \]

yields,

\[ \text{Arg} = \cos(W_k) = \cos(u_k + j \ v_k) \quad (5.9) \]

where,

\[ u_k = \frac{(2 \ k + 1) \pi}{2 \ n} \quad \text{for} \ k = 0,1,2, \ldots, 2n - 1 \quad (5.10) \]
\[ v_k = \sinh \left( e^k \right) \text{ for all } k \]  \hspace{1cm} (5.11)

From these roots the transfer function denominator polynomial must be derived. Using a similar development to that of the earlier filters, this can be attained.

Letting,

\[ (z_1 + z_1) + (z_2 + z_2) = 2(e \cos \theta) \]  \hspace{1cm} (5.12)

yields,

\[ \xi_k = \cosh \left( (1 + \cos (\gamma \pi)) \sin (W_k - 1) \right) \]  \hspace{1cm} (5.13)

As before, (5.12) must be represented in the form of realizable polynomial factors. Unfortunately, the earlier developed polynomial factor was insufficient, producing an unstable filter. Therefore, an additional factor was derived to compensate for this instability. The two possible polynomial factors are stated as follows,

\[ (z_1 + z_2 -2e)(z_1 + z_2 -2e) = 0 \]  \hspace{1cm} (5.14)

\[ (z_1 + z_2 -2e)(z_1 + z_2 -2e) = 0 \]  \hspace{1cm} (5.15)

The selection of an appropriate factor depends on the value of \( \xi_k \).

The selection of the proper polynomial factor is summarized as follows, with the filter's transfer function.

\[ H(z_1, z_2) = \frac{1}{2n-1 \xi_k} \]  \hspace{1cm} \[ k = 0 \]

\[ \xi' = \xi_k \text{ if real component of } \xi_k \text{ is positive} \]

\[ = -\xi_k \text{ if real component of } \xi_k \text{ is negative} \]
B. Design Examples

The previously illustrated filter design technique can be summarized as shown in Figure 5.1. The computer programs used to study the responses of this filter are given in Appendix 6. Figure 5.2 illustrates the magnitude response of an example filter. This response is clearly high-pass in nature and possesses a well defined passband, with ripple. The root trajectory map of this filter is given in Figure 5.3. This map clearly indicates a stable filter. Although not included in this report, the impulse response of this filter was also generated and converged to zero as expected. The group delays of the filter are also given in Figures 5.4 and 5.5. These delays do possess the necessary flat response in the passband of the filter.

The input parameters to the design technique, as given earlier in Figure 5.1, are Amax, n and γ. The input parameter γ works as expected to vary the stopband size. The larger the value of γ, the larger the stopband is in radial size. The other input parameters behave as expected. Table 5.1 summarizes response characteristics obtained from this filter design.
Chebyshev High-Pass 2-Dimensional Digital Filter Design

\[ H(z_1, z_2) = \frac{1}{B(z_1, z_2)} \]

\[ B(z_1, z_2) = \prod_{k=0}^{2n-1} (z_1 + z_2 - 2e^{-\xi_k}) \]

\[ \xi_k = \cosh \left( \left(1 + \cos \left( \gamma \pi \right) \right) \sin \left( \frac{W_k}{2} \right) - 1 \right) \]

\[ W_k = u_k + j v_k \]

\[ u_k = \frac{(2k + 1)\pi}{2n} \]

\[ v_k = \frac{-1 - 1}{n} \cdot \frac{-1}{n} \cdot \sinh \left( \epsilon \right) \]

\[ \epsilon = \frac{(10 - 1)}{2n} \]

\[ n = \text{filter order} = 0.5, 1, 1.5, 2.0, ... \]

\[ \gamma = \text{normalized stopband size} \quad 0.0 < \gamma < 1.0 \]

\[ A_{\text{max}} = \text{passband ripple (dB)} \]

Figure 5.1: Chebyshev High-Pass Design Method
Figure 5.2: Example Filter Magnitude Response

Figure 5.3: Example Filter Root Trajectories

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 5.4: Example Filter Group Delay 1

Figure 5.5: Example Filter Group Delay 2
<table>
<thead>
<tr>
<th>Filter Input Parameters</th>
<th>Filter Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( A_{\text{max}} )</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note - Radial Stopband size measured from \( \omega_1 T = \omega_2 T = 0.0 \) to -3 dB point.
-Magnitude Response scaled for 0 dB at \( \omega_1 T = \omega_2 T = \pi \)

Table 5.1: Examples of Chebyshev High-Pass Filters

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
VI. GENERAL CHEBYSHEV 2-DIMENSIONAL DIGITAL FILTER

In this chapter, the development and design examples of the general Chebyshev 2-dimensional digital filter will be presented. This filter provides for direct design of either a low-pass, band-pass or high-pass filter by varying the appropriate input parameter. This filter follows a similar development to that discussed for the high-pass filter, except, that a new Chebyshev argument will be developed.

A. Filter Development

The development of another 2-dimensional digital Chebyshev argument, Arg, must now be undertaken. In the case of developing a general Chebyshev argument, it is necessary to satisfy different constraints with respect to the magnitude of the argument. These constraints are as follows,

\[ |f(\cos(\omega_1 T), \cos(\omega_2 T))| = \leq 1.0 \]

for \((\gamma_1 \pi) = \leq (\omega_1 T) + (\omega_2 T) = \leq (\gamma_2 \pi)\)

representing the passband region and,

\[ |f(\cos(\omega_1 T), \cos(\omega_2 T))| = > 1.0 \]

for \((\gamma_2 \pi) < (\omega_1 T) + (\omega_2 T) < (\gamma_1 \pi)\)

representing the stopband region, where,

\(\gamma_1 = \) normalized lower passband edge parameter

\(0.0 = \leq \gamma_1 = \gamma_2\)

\(\gamma_2 = \) normalized upper passband edge parameter

\(\gamma_1 < \gamma_2 = \leq 1.0\)

Consider the following relationship,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = \alpha (\cos(\omega_1 T) + \cos(\omega_2 T)) + 2 \beta \quad (6.1) \]

where,

\(\alpha, \beta = \) scaling parameters evaluated to satisfy Chebyshev argument requirements

With the following constraint,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = 1.0 \text{ for } \omega_1 T = \omega_2 T = \gamma_1 \pi \]
in (6.1) leads to,

\[ 2 \beta = 1 - 2 \alpha \cos(\gamma_1 \pi) \]  

(6.2)

In addition,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = -1.0 \text{ for } \omega_1 T = \omega_2 T = \gamma_2 \pi \]

in (6.1) leads to,

\[ 2 \alpha \cos(\gamma_2 \pi) + 2 \beta = -1 \]  

(6.3)

Solving for the scaling parameters in (6.2) and (6.3) yields the following,

\[ \alpha = \frac{1}{\cos(\gamma_1 \pi) - \cos(\gamma_2 \pi)} \]

(6.4)

\[ 2 \beta = \frac{-(\cos(\gamma_1 \pi) + \cos(\gamma_2 \pi))}{\cos(\gamma_1 \pi) - \cos(\gamma_2 \pi)} \]  

(6.5)

By using the result of (6.4) and (6.5) in (6.1) leads to the following function,

\[ f(\cos(\omega_1 T), \cos(\omega_2 T)) = \]

\[ -\cos(\gamma_1 \pi) - \cos(\gamma_2 \pi) + \cos(\omega_1 T) + \cos(\omega_2 T) \]

\[ \cos(\gamma_1 \pi) - \cos(\gamma_2 \pi) \]  

(6.6)

This function is a suitable Chebyshev argument in the variables \( \cos(\omega_1 T) \) and \( \cos(\omega_2 T) \). By using the results of (2.10), this equation can be expressed in terms of the z-domain variables as follows,

\[ \text{Arg} = -2 \left( \cos(\gamma_1 \pi) - \cos(\gamma_2 \pi) \right) + z_1 + z_1 + z_2 + z_2 \]

\[ 2 \left( \cos(\gamma_1 \pi) - \cos(\gamma_2 \pi) \right) \]  

(6.7)

It is now necessary to find the denominator roots of the transfer function. This is an identical solution to that shown earlier, so only the results will be summarized.

Solving,

\[ 1 + (e C_n(\text{Arg})) = 0 \]  

(6.8)

yields,

\[ \text{Arg} = \cos(W_k) = \cos(u_k + j v_k) \]  

(6.9)

where,
\[ u_k = \frac{(2k + 1) \pi}{2n} \text{ for } k = 0, 1, 2, ... 2n - 1 \]  
(6.10)

\[ v_k = \sinh \left( e^{-k} \right) \text{ for all } k \]  
(6.11)

From these roots, the transfer function denominator polynomial must be derived. Using an identical solution to that of the Chebyshev high-pass filter, the denominator polynomial factors can be formed as follows,

\[ \xi_k = \cosh \left( \cos(\gamma_1 \pi) \cos(W_k) + \cos(\gamma_2 \pi) \sin(W_k) \right) \]  
(6.12)

where,

\[ H(z_1, z_2) = \frac{1}{z_1 + z_2 - 2e} \]  
(6.13)

\[ \xi_k' = \xi_k \text{ if real component of } \xi_k \text{ is positive} \]
\[ = -\xi_k \text{ if real component of } \xi_k \text{ is negative} \]

B. Design Examples

The previously illustrated filter design technique can be summarized as shown in Figure 6.1. The computer programs used to study the responses of this filter are given in Appendix 7. Because of the general nature of this filter, its low-pass and high-pass responses will be studied separately.

Figure 6.2 illustrates the magnitude response of an example low-pass filter. This response is clearly low-pass in nature and possesses a well defined passband, with ripple. The root trajectory map of this filter is given in Figure 6.3. This map clearly indicates a stable filter. Although not included in this report, the impulse response of this filter was also generated and converged to zero as expected. The group delays of the filter are also given in Figures 6.4 and 6.5. These delays do possess the necessary flat response in the passband of the filter. In comparison to the earlier
General Chebyshev 2-Dimensional Digital Filter Design

\[ H(z_1, z_2) = \frac{1}{B(z_1, z_2)} \]

\[ B(z_1, z_2) = \prod_{k=0}^{2n-1} \left( z_1 + z_2 - 2 \varepsilon \right) \]

\[ \xi_k = \cosh \left( \cos(\gamma_1 \pi) \cos \left( \frac{W_k}{2} \right) + \cos(\gamma_2 \pi) \sin \left( \frac{W_k}{2} \right) \right) \]

\[ W_k = u_k + j v_k \]

\[ u_k = \frac{(2k + 1) \pi}{2n} \]

\[ v_k = \frac{-1}{n} \sinh (\varepsilon) \]

\[ \varepsilon = (0.1 \text{ Amax}, 0.5, 0, 1) \]

\[ n = \text{filter order} = 1, 2, 3, \ldots \]

\[ \gamma_1 = \text{normalized lower passband edge} 0.0 < \gamma_1 < \gamma_2 \]

\[ \gamma_2 = \text{normalized upper passband edge} \gamma_1 < \gamma_2 < 1.0 \]

\[ \text{Amax} = \text{passband ripple (dB)} \]

Figure 5.1: General Chebyshev Design Method
Type - Gen Cheb $n = 4$ $A_{\text{max}} = 0.5$ $\gamma_1 = 0.0$ $\gamma_2 = 0.4$

Figure 6.2: Example Low-Pass Filter Magnitude Response

Type - Gen Cheb $n = 4$ $A_{\text{max}} = 0.5$ $\gamma_1 = 0.0$ $\gamma_2 = 0.4$

Figure 6.3: Example Low-Pass Filter Root Trajectories
Type - Gen Cheb  \( n = 4 \) \( A_{\text{max}} = 0.5 \)  \( \gamma_1 = 0.0 \)  \( \gamma_2 = 0.4 \)

Figure 6.4: Example Low-Pass Filter Group Delay 1

Type - Gen Cheb  \( n = 4 \) \( A_{\text{max}} = 0.5 \)  \( \gamma_1 = 0.0 \)  \( \gamma_2 = 0.4 \)

Figure 6.5: Example Low-Pass Filter Group Delay 2

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
discussed Chebyshev low-pass techniques, this filter offers approximately the same response as attained from the first Chebyshev design.

The input parameters to the design technique, as given earlier in Figure 6.1, are $A_{\text{max}}$, $n$, $\gamma_1$, and $\gamma_2$. All input parameters function as expected. Table 6.1 summarizes results attained from this filter's low-pass design capabilities.

Figure 6.6 illustrates the magnitude response of an example high-pass filter. This response is high-pass in nature and possesses a well defined passband, with ripple. The root trajectory map of this filter is given in Figure 6.7. This root map clearly indicates a stable filter. The group delays of the filter are also given in Figures 6.8 and 6.9. These delays do possess the necessary flat response in the passband of the filter. In comparison to the earlier discussed Chebyshev high-pass techniques, this filter offers approximately the same response.

Table 6.2 summarizes results attained from this filter's high-pass design capabilities.
<table>
<thead>
<tr>
<th>n</th>
<th>A_{max}</th>
<th>\gamma_1</th>
<th>\gamma_2</th>
<th>Radial Passband Size Min. [rad]</th>
<th>Max. [rad]</th>
<th>Maximum Passband Gain [dB]</th>
<th>Maximum Stopband Attenuation [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>0.0</td>
<td>0.2</td>
<td>0.806</td>
<td>0.939</td>
<td>0.480</td>
<td>48.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.4</td>
<td>1.390</td>
<td>1.970</td>
<td>0.480</td>
<td>23.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.6</td>
<td>2.620</td>
<td>3.320</td>
<td>0.490</td>
<td>8.41</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.806</td>
<td>0.939</td>
<td>0.967</td>
<td>51.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.4</td>
<td>1.370</td>
<td>1.880</td>
<td>0.975</td>
<td>26.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.6</td>
<td>2.400</td>
<td>3.160</td>
<td>0.996</td>
<td>10.9</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.0</td>
<td>0.2</td>
<td>0.581</td>
<td>0.911</td>
<td>0.457</td>
<td>80.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.4</td>
<td>1.030</td>
<td>1.840</td>
<td>0.473</td>
<td>42.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.6</td>
<td>1.940</td>
<td>2.850</td>
<td>0.494</td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.581</td>
<td>0.911</td>
<td>0.910</td>
<td>83.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.4</td>
<td>1.030</td>
<td>1.840</td>
<td>0.950</td>
<td>45.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.6</td>
<td>1.820</td>
<td>2.770</td>
<td>0.987</td>
<td>22.3</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.0</td>
<td>0.2</td>
<td>0.456</td>
<td>0.911</td>
<td>0.496</td>
<td>112.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.4</td>
<td>0.911</td>
<td>1.820</td>
<td>0.496</td>
<td>62.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.6</td>
<td>1.590</td>
<td>2.740</td>
<td>0.500</td>
<td>31.2</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.456</td>
<td>0.806</td>
<td>0.990</td>
<td>115.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.4</td>
<td>0.911</td>
<td>1.740</td>
<td>0.990</td>
<td>65.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.6</td>
<td>1.490</td>
<td>2.730</td>
<td>1.000</td>
<td>33.9</td>
</tr>
</tbody>
</table>

Note - Radial Passband Size measured from \(\omega_1T = \omega_2T = 0.0\) to -3 dB point
-Magnitude Response scaled for 0 dB at \(\omega_1T = \omega_2T = 0\)

Table 6.1: Examples of General Chebyshev Low-Pass Filters
Figure 6.6: Example High-Pass Filter Magnitude Response

Figure 6.7: Example High-Pass Filter Root Trajectories
Type - Gen Cheb $n = 4 \ A_{\text{max}} = 0.5 \ \gamma_1 = 0.25 \ \gamma_2 = 1.0$

Figure 6.8: Example High-Pass Filter Group Delay 1

Type - Gen Cheb $n = 4 \ A_{\text{max}} = 0.5 \ \gamma_1 = 0.25 \ \gamma_2 = 1.0$

Figure 6.9: Example High-Pass Filter Group Delay 2
<table>
<thead>
<tr>
<th>n</th>
<th>Amax</th>
<th>γ1</th>
<th>γ2</th>
<th>Radial Stopband Size</th>
<th>Maximum Passband Gain</th>
<th>Maximum Stopband Attenuation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[rad]</td>
<td>[dB]</td>
<td>[dB]</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.2</td>
<td>1.0</td>
<td>0.342</td>
<td>0.496</td>
<td>3.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>1.0</td>
<td>1.140</td>
<td>0.490</td>
<td>10.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4</td>
<td>1.0</td>
<td>1.600</td>
<td>0.450</td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.2</td>
<td>1.0</td>
<td>0.570</td>
<td>0.990</td>
<td>5.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>1.0</td>
<td>1.140</td>
<td>0.976</td>
<td>12.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4</td>
<td>1.0</td>
<td>1.710</td>
<td>0.973</td>
<td>22.3</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>0.2</td>
<td>1.0</td>
<td>0.693</td>
<td>0.499</td>
<td>7.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.25</td>
<td>1.0</td>
<td>1.030</td>
<td>0.490</td>
<td>12.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>1.0</td>
<td>1.250</td>
<td>0.498</td>
<td>18.4</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.2</td>
<td>1.0</td>
<td>0.797</td>
<td>0.998</td>
<td>9.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.25</td>
<td>1.0</td>
<td>1.030</td>
<td>0.979</td>
<td>15.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>1.0</td>
<td>1.250</td>
<td>0.990</td>
<td>21.2</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>0.15</td>
<td>1.0</td>
<td>0.470</td>
<td>0.500</td>
<td>6.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.2</td>
<td>1.0</td>
<td>0.797</td>
<td>0.480</td>
<td>12.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.25</td>
<td>1.0</td>
<td>1.030</td>
<td>0.488</td>
<td>19.4</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.15</td>
<td>1.0</td>
<td>0.570</td>
<td>1.140</td>
<td>6.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.2</td>
<td>1.0</td>
<td>0.797</td>
<td>1.840</td>
<td>15.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.25</td>
<td>1.0</td>
<td>1.140</td>
<td>0.974</td>
<td>22.2</td>
</tr>
</tbody>
</table>

Note - Radial Stopband size measured from ω1T = ω2T = 0.0 to -3 dB point
-Magnitude Response scaled for 0 dB at ω1T = ω2T = π

Table 6.2: Examples of General Chebyshev High-Pass Filters
VII. FILTER IMPLEMENTATION

The implementation, through software, of 2 dimensional digital filters, designed by the previously discussed methods, was studied to illustrate their application in the field of image processing. The method of implementation will be discussed first, and, results attained from filtering some selected images will then be illustrated.

A. 2-Dimensional Recursive Digital Filter Implementation

The basic form of the filter transfer functions attained by use of the previously described design techniques is in the following form,

\[ H(z_1, z_2) = \frac{1}{\prod_{k=0}^{2n-1} (z_1 + z_2 + c_k)} \]  

where,

\[ c_k = \text{complex valued constant calculated as per specific design method and input parameters} \]

\[ n = \text{filter order} \]

It is necessary, for implementation, to express this transfer function as an expanded polynomial instead of the product of factors, as follows,

\[ H(z_1, z_2) = a_{00} \sum_{i=0}^{N} \sum_{j=0}^{N} b_{ij} z_1^i z_2^j \]  

where,

\[ a_{00} = \text{numerator coefficient (used for magnitude scaling)} \]

\[ N = 2n \]

\[ b_{ij} = \text{denominator polynomial coefficients} \]  

\[ (b_{00} = 1.0) \]

The conversion from (7.1) to (7.2) is accomplished by use of the computer program.
TRANSFER.PAS. The source code listing of this program is given in each methods' appendix, respectively.

This function (7.2) can now be expressed in a difference equation form, which can be implemented, as follows,

\[
y(n_1,n_2) = a_{00} x(n_1,n_2) - \sum_{i=0}^{N} \sum_{j=0}^{N} b_{ij} y(n_1-i,n_2-j) \quad (7.3)
\]

where,

- \(x(n_1,n_2)\) = 2-dimensional input sequence
- \(y(n_1,n_2)\) = 2-dimensional output sequence
- \(n_1,n_2\) = sequence indices

Through studying the form of the filter polynomials, (7.3) can be simplified to only consider the non-trivial filter coefficients. By considering this upper triangular region of recursion, the following can be stated,

\[
y(n_1,n_2) = a_{00} x(n_1,n_2) - \sum_{i=0}^{N} \sum_{j=0}^{N-i} b_{ij} y(n_1-i,n_2-j) \quad (7.4)
\]

In addition, through observing symmetry in the filter polynomial coefficients, a further simplification can be made. Since,

\[
b_{ij} = b_{ji}
\]

(7.4) can be computationally simplified to,

\[
y(n_1,n_2) = a_{00} x(n_1,n_2) - \sum_{i=1}^{N-i} b_{ii} y(n_1-i,n_2-i) \quad (7.5)
\]

\[
- \sum_{j=0}^{N-j-1} \sum_{k=j}^{N-j} b_{jk} (y(n_1-j,n_2-k) + y(n_1-k,n_2-j))
\]

B. Implementations in Image Processing

Image processing provides vast applications for the use of 2-dimensional digital filters. Low-pass
filters are used for smoothing images. This is done primarily for diminishing spurious effects that may be present in a digital image [12]. In contrast, high-pass filtering is done primarily for feature and edge enhancement [12].

For testing of the capabilities of filters designed using the previously developed techniques their implementation through software (program source code given in Appendix 8) was undertaken. The filters used for each image are given in Table 7.1 and the images are shown in Figures 7.1 through 7.6. Equation (7.4) was used in the implementation software. In Figures 7.1 and 7.2, various degrees of blurring were introduced into the digital images. As the filters’ passband size increased radially, the amount of blurring decreased, as expected. In Figure 7.3, the sign image was processed with various high-pass filters. As the size of the filters’ stopband increased radially, more information was removed from the original image. In the case of the upper right image, dynamic compression was achieved, due to a relatively small stopband. For the lower right image, all illumination effects are suppressed and transitions remain. For the piston image, identical filters to those implemented on the sign image were used. Due to partial dynamic range usage in the original image and full dynamic range scaling in the implementation software, the upper right image is an enhancement of the original image. Additionally, due to non-uniform illumination of the original image, the horizontal and vertical transitions were not sensed in the lower left image. In Figure 7.4, high-pass filters designed by the general Chebyshev filter method were implemented. As was done in the previous image varying radial sizes of stopband were used and are summarized in Table 7.1. For the chess board image, rising transitions were sensed, but, due to the filter order relative to the image size, falling transitions were detected to a smaller degree. With the skull image, dynamic compression was achieved in the upper right processed image and further compression was shown in the lower right image. In the case of the lower left image, most of the image information was removed and noise amplification was introduced.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Image</th>
<th>Filter Type</th>
<th>Location</th>
<th>n</th>
<th>( A_{\text{max}} )</th>
<th>( \gamma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Terminal</td>
<td>Cheb1</td>
<td>UL</td>
<td>4</td>
<td>1.0</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR</td>
<td>4</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>4</td>
<td>1.0</td>
<td>0.7</td>
</tr>
<tr>
<td>7.2</td>
<td>Terminal</td>
<td>Butter</td>
<td>UL</td>
<td>4</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR</td>
<td>4</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>4</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>7.3</td>
<td>Sign</td>
<td>Cheb HP</td>
<td>UL</td>
<td>4</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR</td>
<td>4</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>4</td>
<td>0.5</td>
<td>0.7</td>
</tr>
<tr>
<td>7.4</td>
<td>Chess</td>
<td>Gen Cheb</td>
<td>UL</td>
<td>3</td>
<td>1.0</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR</td>
<td>3</td>
<td>1.0</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>3</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td>Skull</td>
<td>Gen Cheb</td>
<td>UL</td>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UR</td>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LR</td>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**UL** = Upper Left  \hspace{1cm} **UR** = Upper Right  \hspace{1cm} **LL** = Lower Left  \hspace{1cm} **LR** = Lower Right

Table 7.1: Filter Input Parameters for Processed Images

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 7.1: Images Processed with First Chebyshev Low-Pass Filters

Figure 7.2: Images Processed with Butterworth Low-Pass Filters
Figure 7.3: Images Processed with Chebyshev
High-Pass Filters

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 7.4: Images Processed with General Chebyshev High-Pass Filters

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
VIII. CONCLUSIONS

It can be clearly stated that a viable method for the design of inherently stable, first quadrant, 2-dimensional, recursive digital filters has been developed and their efficient implementation in image processing has been illustrated. From the six different design methods presented, the ability to design a desired low-pass, band-pass or high-pass filter has been shown. The stability of any filter designed is guaranteed through its development and the transfer function polynomial is easily studied due to its factorability. In addition, the filters lend themselves to efficient implementation due to their upper triangular first quadrant recursion and coefficient symmetry. Also, since the degree of relative stability of these filters can be determined, they can be implemented in finite precision arithmetic systems.
APPENDIX 1

IBM PC Plotting and VAX Impulse
Response Generation Program Source Code
procedure graph2d(data:form;n:integer;title:form4);

(* sábado 13:40 *)

Procedure to Draw Line graphs

\begin{align*}
\text{begin} & \\
\text{for } I \text{ from } 1 \text{ to } n \text{ do } & \\
\text{end};
\end{align*}

\begin{align*}
\text{for } i = 1 \text{ to } n \text{ do } & \\
\text{begin} & \\
\text{if } xy[1,2] < \text{data}[1,1] \text{ then } xy[1,2] := \text{data}[1,1]; & \\
\text{if } xy[1,1] < \text{data}[1,1] \text{ then } xy[1,1] := \text{data}[1,1]; & \\
\text{if } xy[2,2] < \text{data}[2,1] \text{ then } xy[2,2] := \text{data}[2,1]; & \\
\text{if } xy[2,1] < \text{data}[2,1] \text{ then } xy[2,1] := \text{data}[2,1]; & \\
\text{end};
\end{align*}

**List of Variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>array containing points to be plotted</td>
</tr>
<tr>
<td>n</td>
<td>number of separate graphs contained in data</td>
</tr>
<tr>
<td>I, I</td>
<td>loop counters</td>
</tr>
<tr>
<td>xy, lastx, lasty</td>
<td>computational variables used during plot</td>
</tr>
<tr>
<td>range</td>
<td>array to store range of graph</td>
</tr>
<tr>
<td>flag2, temp2</td>
<td>variables used in program option selection</td>
</tr>
<tr>
<td>theta, diff</td>
<td>variable used to draw unit circle</td>
</tr>
</tbody>
</table>

\begin{align*}
\text{for } i = 1 \text{ to } n \text{ do } & \\
\text{begin} & \\
\text{if } xy[1,2] < \text{data}[1,1] \text{ then } xy[1,2] := \text{data}[1,1]; & \\
\text{if } xy[1,1] < \text{data}[1,1] \text{ then } xy[1,1] := \text{data}[1,1]; & \\
\text{if } xy[2,2] < \text{data}[2,1] \text{ then } xy[2,2] := \text{data}[2,1]; & \\
\text{if } xy[2,1] < \text{data}[2,1] \text{ then } xy[2,1] := \text{data}[2,1]; & \\
\text{end};
\end{align*}
flag2:=false;
temp2:=Upcase(temp2);
if temp2='Y' then
begin
  flag2:=true;
  if xy[1,1]>-1.5 then xy[1,1]:=-1.5;
  if xy[1,2]<1.5 then xy[1,2]:=1.5;
  if xy[2,1]>-1.0 then xy[2,1]:=-1.0;
  if xy[2,2]<1.0 then xy[2,2]:=1.0;
end;

{******************
  Initialize screen
******************}

range[1]:=xy[1,2]-xy[1,1];
range[2]:=xy[2,2]-xy[2,1];
if range[1]<1.5*range[2] then
begin
  range[1]:=1.5*range[2];
  diff:=xy[1,1]+range[1]-xy[1,2];
  xy[1,1]:=xy[1,1]-diff/2;
  xy[1,2]:=xy[1,2]+diff/2;
end;
c1sclr;
hires;
hirescolor(15);
gotoxy(31,1);
write(title);
for i:=1 to 5 do
begin
gotoxy(1,2+5*(i-1));
write(xy[2,2]-(i-1)*range[2]/4:7:3);
end;
for i := -1 to 7 do
begin
gotoxy(8+(i-1)*11,24);
write(xy[1,1]+(i-1)*range[1]/6:7:3);
end;
draw(75,10,639,10,15);
draw(639,10,639,175,15);
draw(639,175,75,175,15);
draw(75,175,75,10,15);
for i:=1 to 5 do
begin
draw(72,round(10*(i-1)*41.3),78,round(10*(i-1)*41.3),15);
if i<5 then
begin
  for j:=1 to 4 do
  begin
draw(74,round(10*(i-1)*41.3+j*8.26),76,round(10*(i-1)*41.3+j*8.26),15);
  end;
end;
end;
for i:=1 to 7 do
begin
draw(round(75+(i-1)*94),172,round(75+(i-1)*94),178,15);
if i>7 then
begin
for j:=1 to 5 do
draw(round(75+(i-1)*94+j*18.8),174,round(75+(i-1)*94+j*18.8),176,15);
end;
end;

loop to draw multiple graphs on same picture

for i:=1 to n do
begin
lastx:=round((data[1,1+(i-1)*npts]-xy[1,1])/range[1]*564.0+75);
lasty:=175-round((data[2,1+(i-1)*npts]-xy[2,1])/range[2]*165);

loop to draw each graph

for i:=2 to npts do
begin
x:=round((data[1,1+(i-1)*npts]-xy[1,1])/range[1]*564.0+75);
y:=175-round((data[2,1+(i-1)*npts]-xy[2,1])/range[2]*165);
draw(x,y,lastx,lasty,15);
lastx:=x;
lasty:=y;
end;
end;

draw unit circle

if flag2 then
begin
for i:=1 to 100 do
begin
theta:=(i-1)/100*2*pi;
x:=round((cos(theta)-xy[1,1])/range[1]*564.0+75);
y:=175-round((sin(theta)-xy[2,1])/range[2]*165);
plot(x,y,15);
end;
end;

for i:=1 to 200 do
begin
y:=175-round(-xy[2,1]/range[2]*165);
x:=round(((range[1])*(i-1)/199)/range[1]*564+75);
plot(x,y,15)
end;

for i:=1 to 175 do
begin
y:=175-round(range[2]*(i-1)/174/range[2]*165);
x:=round((-xy[1,1])/range[1]*564+75);
plot(x,y,15)
end;

(****************************
Halt for user initiated screen dump
****************************)

repeat until KeyPressed;
end;
procedure graph3d(data:form;xy:form2;title:=form4);

{**************************************************************************}
Procedure to draw a 3D surface on screen
**************************************************************************}

var
l,j,k,l1,l2,l3,l4,l5,l6,l7,l8,x,y,x2,y2 : integer;
zmax,zmin, fact, fact1, fact2, fact3: real;
dxy : array [1..2..1..npts] of integer;
rangex : array [1..3] of real;
input : char;
flag : boolean;

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

List of variables

Name               Use

l,j,k = Loop counters
zmax,zmin = Maximum and minimum values in z direction
fact, fact1, fact2, fact3 = Computational variables
dxy = array to store transformed 3D to 2D points
rangex = computational array storing range of numbers

**************************************************************************

begin

{**************
Initialize screen
**************}
clrscr;
writeIn;
writeIn(' Plot of ',title);
write(' Do you want hidden line removal (Y/N)?
readln(input);
if upcase(input) = 'Y' then
  flag:=true
else
  flag:=false;
hires;
hirescolor(15);

{***************
Find Zmax and Zmin
***************}

zmax:=data[3,1];
zmin:=data[3,1];
for l:=1 to npts do
begin
56   if zmax<data[3,1] then zmax:=data[3,1];
57   if zmin>data[3,1] then zmin:=data[3,1];
58 end;
59   if zmax=zmin then zmax:=zmax+1;
60
61  [----------------------------------------------------------]
62  Draw Initial screen - axis' and range
63  [----------------------------------------------------------]
64
65  gotoxy(21,1);
66  write(title);
67  gotoXY(73,8);
68  write(zmax:7:3);
69  gotoxy(73,18);
70  write(zmin:7:3);
71  gotoxy(1,20);
72  write(xy[2,1]:7:3);
73  gotoxy(71,20);
74  write(xy[1,1]:7:3);
75  gotoxy(40,25);
76  write(xy[2,2]:7:3);
77  gotoxy(30,25);
78  write(xy[1,2]:7:3);
79  draw(307,101,575,146,15);
80  draw(307,101,40,146,15);
81  draw(307,101,307,11,15);
82  draw(40,146,40,56,15);
83  draw(40,56,307,11,15);
84  draw(307,11,575,56,15);
85  draw(575,56,575,146,15);
86  draw(575,146,307,191,15);
87  draw(307,191,40,146,15);
88  range[1]:=xy[1,2]-xy[1,1];
89  range[2]:=xy[2,2]-xy[2,1];
90  range[3]:=zmax-zmin;
91
92  [----------------------------------------------------------]
93  Perform transformation of 3D points into 2D screen array
94  [----------------------------------------------------------]
95
96  fact:=xy[1,1]/range[1]-xy[2,1]/range[2];
97  fact1:=xy[1,1]/range[1]-xy[2,1]/range[2];
98  for i:=1 to npts do
99     begin
100        fact2:=data[2,1]/range[2]-data[1,1]/range[1];
101        fact3:=data[2,1]/range[2]-data[1,1]/range[1];
102        dxy[1,1]:=307+round(fact2*fact1*267);
103        dxy[2,1]:=101+round((fact3*fact1)*45-(data[3,1]-zmin)/range[3]*90);
104     end;
105
106  [----------------------------------------------------------]
107  Draw graph in x-z plane
108  [----------------------------------------------------------]
for i=2 to nptsx do begin
  j:=(i-1)*nptsy+1;
draw(dxy[1,j],dxy[2,j],dxy[1,j-nptsy],dxy[2,j-nptsy],15);
end;

{*******************************************************************************
  Draw graph in y-z plane
*******************************************************************************
}

for i=2 to nptsy do begin
draw(dxy[1,1],dxy[2,1],dxy[1,1-nptsx],dxy[2,1-nptsx],15);
end;

{*******************************************************************************
  Draw surface as a series of squares in xyz space
*******************************************************************************
}

for i=2 to nptsy do begin
  for j=2 to nptsx do begin
    k:=(j-1)*nptsy+1;
x:=dxy[1,k];
y:=dxy[2,k];
if flag then begin
  x2:=dxy[1,k-nptsy];
y2:=dxy[2,k-nptsy];
r1:=x2-dxy[1,k-1];
r2:=y2-dxy[2,k-1];
r3:=x2-dxy[1,k-nptsy];
r4:=y2-dxy[2,k-nptsy];
r5:=x-dxy[1,k-1];
r6:=y-dxy[2,k-1];
r7:=x-dxy[1,k-nptsy];
r8:=y-dxy[2,k-nptsy];
for l=1 to 15 do begin
  draw(x,y,round(dxy[1,k-1]+r1*l/15),round(dxy[2,k-1]+r2*l/15),0);
draw(x,y,round(dxy[1,k-nptsy]+r3*l/15),round(dxy[2,k-nptsy]+r4*l/15),0);
draw(x2,y2,round(dxy[1,k-1]+r5*l/15),round(dxy[2,k-1]+r6*l/15),0);
draw(x2,y2,round(dxy[1,k-nptsy]+r7*l/15),round(dxy[2,k-nptsy]+r8*l/15),0);
end;
end;
draw(dxy[1,k-1],dxy[2,k-1],dxy[1,k-nptsy-1],dxy[2,k-nptsy-1],15);
draw(dxy[1,k-nptsy],dxy[2,k-nptsy],dxy[1,k-nptsy-1],dxy[2,k-nptsy-1],15);
end;

end;
end;
{**************************************************
Pause for keyboard initiated screen dump
**************************************************}
repeat until KeyPressed;
end:
This is a program to generate the impulse response of a two dimensional filter from its transfer function.

INPUT - this program expects to read a disk file that contains the coefficients of the transfer function in expanded form. After the integer order with respect to z1 and z2, respectively, the real numbers representing the coefficients should be available in the following order:

\( c(0,0), c(0,1), c(0,2), \ldots c(0,n), c(1,0), \ldots c(1,n), \ldots c(m,n) \)

OUTPUT - the program will output an array of identical form containing the impulse response. If the response does not decay to zero the program will indicate it with a message.

written by Tim Kent
June 87

```fortran
real c(50,50),y(1000,1000)
integer m,n,a,b
character*20 filein,fileout
logical flag1,flag2,error

max=1000
small=1.0e-5

print *, 'Enter Input Filename'
read 10, filein
format(a20)
print *, 'Enter Output Filename'
read 10, fileout
open(1, file=filein, status='old')
open(2, file=fileout, status='new')

read(1,*) m
read(1,*) n
read(1,*) scale
do i=1,n+1
  do j=1,m+1
    read(1,*) c(i,j)
  end do
end do

flag1=.true.
error=.false.
i=1
jmax=0
```
55  do while (flag1)
56    flag1=.false.
57    flag2=.true.
58    J=1
59    do while (flag2)
60      flag2=.false.
61      output=0.0
62      if (l.eq.1.and.j.eq.1) output=scale
63      do k=2,m+1
64        k2=1-k+1
65        if (k2.gt.0) then
66          c
67          print*, j,j,' ',output,' ',c(l,k),',',y(j,k2)
68          output=output-c(l,k)*y(j,k2)
69          if (abs(y(j,k2)).gt.small) flag2=.true.
70        endif
71      end do
72      do l=2,n+1
73        l2=j+1
74        if (l2.gt.0) then
75          c
76          print*, l,l,' ',output,' ',c(l,l),',',y(l2,l)
77          output=output-c(l,l)*y(l2,l)
78          if (abs(y(l2,l)).gt.small) flag2=.true.
79        endif
80      end do
81      do k=2,m+1
82        do l=2,n+1
83          k2=1-k+1
84          l2=j+1
85          if (k2.gt.0.and.l2.gt.0) then
86            c
87            print*, l,l,' ',output,' ',c(l,k),',',y(l2,k2)
88            output=output-c(l,k)*y(l2,k2)
89            if (abs(y(l2,k2)).gt.small) flag2=.true.
90          endif
91      end do
92      end do
93      if (abs(output).gt.small) flag2=.true.
94      c
95      print*, j,j,output
96      if (abs(output).lt.small/100) output=0.0
97      y(j,l)=output
98      if (flag2) flag1=.true.
99      if (j.ge.jmax) then
100     jmax=j
101    else
102      flag2=.true.
103    endif
104     J=J+1
105     if (J.gt.jmax) then
106      error=.true.
107      flag2=.false.
108     endif
109     end do
110  l=l+1
if (l.gt.max) then
  error=.true.
  flagl=.false.
endif
end do
imax=1-1
if (error) print *, 'Response did not reach zero'
write(2,*) imax
write(2,*) jmax
do j=1,jmax
do i=1,imax
  write(2,*) y(j,i)
end do
end do
stop
end
This program reads data files written by IMPULSE.FOR and creates a REGIS graphics file that when typed to the terminal draws a surface representing the impulse response.

By Tim Kent June 87

Language Vax FORTRAN

`real data(500,500),xy(2,2),range(3)
integer dxy(2,250000)
character*20, filein, fileout

print*, 'Enter data filename'
read 10, filein
10 format(a20)
print*, 'Enter REGIS format output filename'
read 10, fileout
open(1, file=filein, status='old')
open(2, file=fileout, status='new')

write(2,*) char(27)//'P0p S(E) S(C0)'
read(1,*) nptsy
read(1,*) nptsx
npts=nptsy*nptsx

xy(1,1)=1
xy(1,2)=nptsx
xy(2,1)=1
xy(2,2)=nptsy
zmax=data(1,1)
zmin=data(1,1)
do i=1, nptsx
   do j=1, nptsy
      read(1,*), data(j,1)
      end do
   end do

close(1)

zmax=data(1,1)
zmin=data(1,1)
do i=1, nptsx
   do j=1, nptsy
      if (data(i,j).gt.zmax) zmax=data(i,j)
      if (data(i,j).lt.zmin) zmin=data(i,j)
   end do
end do

call draw(399,239,799,359,1)
call draw(399,239,0,359,1)
call draw(399,239,399,0,1)
call draw(0,359,0,70,1)"
call draw(0.70,399,0,1)
call draw(399,0,799,70,1)
call draw(799,70,799,359,1)
call draw(799,359,399,479,1)
call draw(399,479,0,359,1)

range(1)=xy(1,2)-xy(1,1)
range(2)=xy(2,2)-xy(2,1)
range(3)=zmax-zmin

fact=xy(1,1)/range(1)-xy(2,1)/range(2)
fact1=xy(1,1)/range(1)-xy(2,1)/range(2)
do l=1,nptsx
   do j=1,nptsy
      fact2=float(j)/range(2)-float(1)/range(1)
      fact3=float(j)/range(2)+float(1)/range(1)
      k=(k-1)*nptsy+j
      dxy(1,k)=399+int((fact2+fact)*399+0.5)
      dxy(2,k)=239+int((fact3+fact1)*120-(data(l,j)-zmin)/range(3)+239+0.5)
   end do
end do

do l=2,nptsx
   j=(j-1)*nptsy+1
   call draw(dxy(1,j),dxy(2,j),dxy(1,j-nptsy),dxy(2,j-nptsy),3)
end do

do l=2,nptsy
   call draw(dxy(1,l),dxy(2,l),dxy(1,l-1),dxy(2,l-1),3)
end do

do l=2,nptsy
   do j=2,nptsx
      k=(j-1)*nptsy+l
      call draw(dxy(1,k),dxy(2,k),dxy(1,k-1),dxy(2,k-1),3)
      call draw(dxy(1,k),dxy(2,k),dxy(1,k-nptsy),dxy(2,k-nptsy),3)
   end do
end do

write(2,50) zmax
50 format(' P[0,460] T''max = ',E15.8,'\n')
write(2,60) zmin
60 format(' P[590,460] T''min = ',E15.8,'\n')
close(2)
stop

end

subroutine draw(a,b,c,d,l)
integer a,b,c,d
write(2,20) a,b,l

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
109 20 format('P[\',14,'\',',14,'\'] M(\',11,\'))
110 write(2,30) c,d
111 30 format('V[\',14,'\',',14,'\'] ')
112 return
113 end
APPENDIX 2

First Chebyshev Low-Pass Filter
Response Program Source Code
program cheb2type newcosh;
{

******************************************************************************

* This program calculates the frequency, phase and delay *
* response of a two dimensional digital filter derived *
* from the second chebyshev argument *
*
* Program By - Tim Kent Date Nov.86 *
* Language - Turbo Pascal 3.0 *

Soltis. T. Kent, FI. Iter Theory - Dr. J.

******************************************************************************

const
ordmax=20;
nptsx=25;        {**- needed for graph3d.pas}
nptsy=25;        {**}
npts=625;        {**}
type
form =array [1..3,1..npts] of real;        {**}
form2 =array [1..2,1..2] of real;        {**}
form3 =array [1..2] of real;        {**}
form4 = string[40];
var
data,data2,deley :form;        {**}
xy :form2;        {**}
amax,eps,fact0,gamma,c1,x,y,s,t,wt1,wt2,mag,p1,p2, scale :real;
phase,deltal,delta2,delp1,delp2,n :real;
l,j,k,m,nextx,nexty :integer;
u,v,cu,cv,zu,zv :array [1..ordmax] of real;
z1,z2,t1,t2,dp,t3,t4 :form3;
flag :boolean;
temp :char;
title :string[40];

Subprograms
******************************************************************************

{$1 graph3d.pas}
procedure cadd(var z1,z2,z3:form3);
******************************************************************************

Procedure to simulate complex addition
******************************************************************************

begin
z1[1]=z2[1]+z3[1];
end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
procedure \texttt{cexp}(\texttt{var} \ x, y : \texttt{real});

\begin{verbatim}
85 \[ \begin{align*}
86 & \text{Procedure to simulate complex exponentiation} \\
87 & \text{*******************************************************************************} \\
88 & \text{Procedure to simulate complex exponentiation} \\
89 & \text{*******************************************************************************} \\
90 & \text{begin} \\
91 & \text{\quad} t := x; \\
92 & \text{\quad} s := y; \\
93 & \text{\quad} x := \text{exp}(t) \cdot \cos(s); \\
94 & \text{\quad} y := \text{exp}(t) \cdot \sin(s); \\
95 & \text{\end{verbatim}

procedure \texttt{cmult}(\texttt{var} \ z1, z2, z3 : \texttt{form3});

\begin{verbatim}
85 \[ \begin{align*}
86 & \text{Procedure to simulate complex multiplication} \\
87 & \text{*******************************************************************************} \\
88 & \text{Procedure to simulate complex multiplication} \\
89 & \text{*******************************************************************************} \\
90 & \text{begin} \\
93 & \text{\end{verbatim}

procedure \texttt{arccosh}(\texttt{var} \ z1, z2 : \texttt{form3});

\begin{verbatim}
85 \[ \begin{align*}
86 & \text{Procedure to calculate \texttt{arccosh} of \texttt{z}} \\
87 & \text{*******************************************************************************} \\
88 & \text{Procedure to calculate \texttt{arccosh} of \texttt{z}} \\
89 & \text{*******************************************************************************} \\
90 & \text{begin} \\
91 & \text{\quad} x, y, b, c, d, e, f, g, t, sign : \texttt{real}; \\
92 & \text{\quad} x := z1[1]; \\
93 & \text{\quad} y := z1[2]; \\
94 & \text{\quad} b := x^2 - y^2 - 1; \\
95 & \text{\quad} c := 2 \cdot x \cdot y; \\
96 & \text{\quad} d := \sqrt{(\text{sqrt}(b^2 + c^2) + b)/2}; \\
97 & \text{\quad} \text{\quad if } c > 0 \text{ then} \\
98 & \text{\quad} \quad \text{sign} := 1 \\
99 & \text{\quad} \text{\quad else} \\
100 & \text{\quad} \quad \text{sign} := -1; \\
101 & \text{\quad} e := y \cdot \text{sign} \cdot \text{sqrt}((\text{sqrt}(b^2 + c^2) - b)/2); \\
102 & \text{\quad} f := \text{ln}((\text{sqrt}(d^2 + e^2))); \\
103 & \text{\quad} \text{\quad if } (\text{abs}(e) \leq 10^{-100}) \text{ and } (d \leq 0.0) \text{ then} \\
104 & \text{\quad} \quad t := \pi \\
105 & \text{\quad} \text{\quad else} \\
106 & \text{\quad} \quad \text{begin} \\
107 & \text{\quad} \quad \text{\quad if } \text{abs}(d) \leq 10^{-100} \text{ then} \\
108 & \text{\quad} \quad \quad \text{begin} \\
\end{verbatim}
if e>0 then
    sign:=1
else
    sign:=-1;
t:=(pi/2)*sign;
end

begin
    t:=arctan(e/d);
    if d<0 then
        if e>0 then
            t:=t+pi
        else
            t:=t-pi;
        end;
    end;
g:=t;
end;

***************
End of subprograms
***************

***************
Mainline Program
***************

begin
    clrscr;

    writeln('Frequency Response - Type I - Chebyshev Argument #2');
    writeln;
    writeln('Enter order of filter (+ve < ,ordmax,'/'2) ; ');
    readln(n);
    m:=trunc(2*n);
    writeln('Enter passband ripple Amax ; ');
    readln(amax);
    writeln('Enter Gamma (0<gamma<1.0) ; ');
    readln(gamma);
    c1:=sin(gamma*pi/2)*sin(gamma*pi/2);
    writeln;
    eps:=sqrt(exp(0.1*amax*ln(10))-1);

    ***************
    Calculation of chebyshev poles
    ***************
for k:=1 to m do begin
  u[k]:=(2*k-1)*pi/2/n;
  v[k]:=fact0;
end;

Calculation of Cosine of chebyshev poles

for k:=1 to m do begin
  cu[k]:=0.5*cos(u[k])*(exp(v[k])+exp(-v[k]));
  cv[k]:=0.5*sin(u[k])*(exp(v[k])-exp(-v[k]));
end;

Calculation of Zsl — (Arcosh(z))

for k:=1 to m do begin
  cu[k]:=cl*cu[k];
  cv[k]:=cl*cv[k];
  z1[1]:=1-cu[k];
  z1[2]:=cv[k];
  arccosh(z1,22);
  zu[k]:=z2[1];
  zv[k]:=z2[2];
end;

Setting of x-y-axis limits for plotting

xy[1,1]:=-pi;
xy[1,2]:=-pi;
xy[2,1]:=-pi;
xy[2,2]:=-pi;

Selection of Table of Values

flag:=false;
write('Do you want a table of values for magnitude and phase (Y/N)?');
readln(temp);
temp:=upcase(temp);
if temp='Y' then begin
  flag:=true;
end;
217  if flag then
218  begin
219    writeln;
220    writeln('Frequency Response - Magnitude and Phase');
221    writeln(' } w*T  w*2*T [H(z1,z2)] <H(z1,z2)');
222    writeln(' (dB) (rad)');
223    writeln;
224  end;
225
226  [{**Calculation of constants in filter polynomial factors**}]
227  for i:=1 to m do
228  begin
229    zu[i]:=zu[i];
230    zv[i]:=zv[i];
231    exp(zu[i],zv[i]);
232    zu[i]:=-2.0*zu[i];
233    zv[i]:=-2.0*zv[i];
234  end;
235
236  [{**Loop to generate filter magnitude response**}]
237  for i:=1 to npts do
238  begin
239    for j:=1 to nptsy do
240    begin
241      wt1:=(1-l)*(xy[1,2]-xy[1,1])/(nptsx-1)+xy[1,1];
242      wt2:=(j-l)*(xy[2,2]-xy[2,1])/(nptsy-1)+xy[2,1];
243      z1[1]:=cos(-wt1);
244      z1[2]:=sin(-wt1);
245      z2[1]:=cos(-wt2);
246      z2[2]:=sin(-wt2);
247      dp[1]:=l.0;
248      dp[2]:=0.0;
249      cadd(t1,z1,z2);
250    end;
251
252  [{**Loop to multiple filter polynomial factors**}]
253  for k:=1 to m do
254  begin
255    t2[1]:=zu[k];
256    t2[2]:=zv[k];
257    cmult(t4,dp,t3);
258    dp[1]:=t4[1];
259    dp[2]:=t4[2];
260  end;
261
262
263
264
265
266
267
268
269
270
Calculation of magnitude in dB with dc scaling

```plaintext
k := (l-1)*nptsy-j;
if k=1 then scale := mag;
mag := 20*log(mag/scale)/log(10);
```

Calculation of phase from -2pi - 0

```plaintext
phase := -1.0*arctan(dp[2]/dp[1]);
if dp[1]<0.0 then
  phase := pi+phase
else
  begin
    if dp[2]<0.0 then phase := 2*pi+phase;
  end;
end;
data[1,k] := wt[l];
data[2,k] := wt[2];
data[3,k] := mag;
data2[1,k] := wt[l];
data2[2,k] := wt[2];
data2[3,k] := phase;
if flag then
  begin
    writeln(wt1:10:6,' ',wt2:10:6,' ',mag:14:6,' ',phase:14:6);
  end;
end;
data := 100.0;
figure := 4.;
figure := 0.0;
begin
  clrscr;
title := ' Magnitude Response (in dB) ';
  graph3d(data.xy,title);
  clrscr;
title := ' Phase Response ';
  graph3d(data2.xy,title);
  clrscr;
```

Simulation of derivative by linear difference equation

```plaintext
Calculation of group delays of filter
```

```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
delta1:=(xy[1,2]-xy[1,1])/nptsx;
delta2:=(xy[2,2]-xy[2,1])/nptsy;
for i=1 to nptsx do
begin
  wt1:=(i-1)*(xy[1,2]-xy[1,1])/(nptsx-1)*xy[1,1];
  for j=1 to nptsy do
  begin
    wt2:=(j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)*xy[2,1];
    k:=(i-1)*nptsy+j;
    nextx:=k+nptsy;
    nexty:=k+1;
    if nexty>nptsy then nexty:=nexty-2;
    if nextx>npts then nextx:=nextx-2*nptsy;
    data[1,k]:=wt1;
    dely[1,k]:=wt1;
    data[2,k]:=wt2;
    dely[2,k]:=wt2;
    delp1:=data2[3,nextx]+data2[3,k];
    delp2:=data2[3,nexty]+data2[3,k];
    if abs(delp1)>pi*1.2 then
      begin
        if delp1>0.0 then
def:=-delp1-2*pi
        else
def:=-delp1+2*pi
      end;
    if abs(delp2)>pi*1.2 then
      begin
        if delp2>0.0 then
def:=delp2-2*pi
        else
def:=delp2+2*pi
      end;
    data[3,k]:=def/deltal;
    dely[3,k]:=def/delta2;
    if j=nptsy then dely[3,k]:=-dely[3,k];
    if i=npts then data[3,k]:=-data[3,k];
  end;
end;
{**************************************************************************}
Scaling of vertical change elements for periodic wrap-around
**************************************************************************
if abs(delp1)>pi*1.2 then
begin
  if delp1>0.0 then
def:=-delp1-2*pi
  else
def:=-delp1+2*pi
end;
if abs(delp2)>pi*1.2 then
begin
  if delp2>0.0 then
def:=delp2-2*pi
  else
def:=delp2+2*pi
end;
data[3,k]:=def/deltal;
dely[3,k]:=def/delta2;
if j=nptsy then dely[3,k]:=-dely[3,k];
if i=npts then data[3,k]:=-data[3,k];
end;
{**************************************************************************}
Call to Graphing Subroutines
**************************************************************************
title:="Group Delay 1";
graph3d(data,xy,title);
cirscr;
title:="Group Delay 2";
graph3d(dely,xy,title);
clrsr;

***************
End of mainline program
***************
end.
program cheb2typelrootnewcosh;
{

This program calculates the root trajectory
of a two dimensional digital filter derived
from the second chebyshev argument

Program By - Tim Kent       Date Nov.86
Language - Turbo Pascal 3.0

Filter Theory - Dr. J. Soltis, T.Kent

Pay attention to the special characters and formatting, such as comments, variables, and functions, and ensure they are accurately transcribed into the text representation.
begin
  mag := z2[1]*z2[1]+z2[2]*z2[2];
  z1[1] := z2[1]/mag;
  z1[2] := z2[2]/mag;
end;

procedure cexp(var z1, z2: form3);

{Procedure to simulate complex exponentiation}

begin
  z1[1] := exp(z2[1])*cos(z2[2]);
  z1[2] := exp(z2[1])*sin(z2[2]);
end;

procedure cart(var z1: form3; theta: real);

{Procedure to perform cartesian to polar conversion}

begin
  z1[1] := cos(theta);
  z1[2] := sin(theta);
end;

procedure arccosh(var z1, z2: form3);

{Procedure to calculate arccosh of z}

var
  x, y, b, c, d, e, f, g, t, sign: real;

begin
  x := z1[1];
  y := z1[2];
  b := x*x-y*y-1;
  c := 2*x*y;
  d := x+sqrt((sqrt(b*b+c*c)+b)/2);
  if c>0 then
    sign := 1
  else
    sign := -1;
  e := y*sign*sqrt((sqrt(b*b+c*c)-b)/2);
  f := ln(sqrt(d^2+e^2));
  if ((abs(e)<1e-10)and(d<0.0)) then
109 \[ t := \pi \]
110 else
111 begin
112 if abs(d) < 1e-100 then
113 begin
114 if e > 0 then
115 \[ \text{sign} := 1 \]
116 else
117 \[ \text{sign} := -1; \]
118 \[ t := \pi / 2 \times \text{sign}; \]
119 end
120 else
121 begin
122 \[ t := \arctan(e / d); \]
123 if d < 0 then
124 if e > 0 then
125 \[ t := t + \pi \]
126 else
127 \[ t := t - \pi; \]
128 end;
129 end;
130 \[ g := t; \]
131 \[ z_2[1] := f; \]
132 \[ z_2[2] := g; \]
133 end;
134 135 136 [********************************]
137 End of subprograms
138 [********************************]
139 140 141 142 Mainline program
143 [********************************]
144 145 begin
146 clrscr;
147 148 [********************************]
149 Input filter specifications
150 [********************************]
151 152 writeln('Root Trajectory - Type 1 - Chebyshev Argument #2');
153 writeln;
154 writeln('Enter order of filter (+ve <', ordmax, '/2) : ');
155 readin(n);
156 \[ m := \text{trunc}(2 \times n); \]
157 writeln('Enter passband ripple A_{\text{max}} : ');
158 readin(amax);
159 writeln('Enter Gamma (0 < \gamma < 1.0) : ');
160 readin(gamma);
161 \[ c_l := \sin(\gamma \times \pi / 2) \times \sin(\gamma \times \pi / 2); \]
162 writeln;
\[
\text{eps} := \sqrt{\exp(0.1 \times \text{amax} \times \ln(I0)) - 1};
\]

Calculate Chebyshev poles

\[
\text{fact0} := -\ln(1/\text{eps} + \sqrt{1/\text{eps}^2 + 1}) / n;
\]

for \( k := 1 \) to \( m \) do

begin

\[ u[k] := (2k - 1) \times \pi / 2 / n; \]

\[ v[k] := \text{fact0}; \]

end;

Calculate cosine of Cheb poles

\[
\text{for } k := 1 \text{ to } m \text{ do }
\begin{align*}
\text{cu}[k] & := 0.5 \times \cos(u[k]) \times (\exp(v[k]) + \exp(-v[k])); \\
\text{cv}[k] & := -0.5 \times \sin(u[k]) \times (\exp(v[k]) - \exp(-v[k])); \\
\end{align*}
\]

end;

Calculate \( z \)sl - \( \text{arcosh}(z) \)

\[
\text{for } k := 1 \text{ to } m \text{ do }
\begin{align*}
\text{cu}[k] & := \text{cl} \times \text{cu}[k]; \\
\text{cv}[k] & := \text{cl} \times \text{cv}[k]; \\
\text{zl}[1] & := 1 - \text{cu}[k]; \\
\text{zl}[2] & := -\text{cv}[k]; \\
\text{arccosh}(z[1], z[2]); \\
\text{zu}[k] & := z[1]; \\
\text{zv}[k] & := z[2]; \\
\end{align*}
\]

end;

Loop to generate root trajectories for all factors

\[
\text{for } l := 1 \text{ to } m \text{ do }
\begin{align*}
\text{tl}[1] & := \text{zu}[l]; \\
\text{tl}[2] & := \text{zv}[l]; \\
\text{cexp}(t[2], t[1]); \\
\text{tl}[1] & := 2 \times t[2]; \\
\text{tl}[2] & := 2 \times t[2]; \\
\end{align*}
\]

Loop to generate points of trajectories

\[
\text{for } l := 1 \text{ to } m \text{ do }
\begin{align*}
\text{tl}[1] & := \text{zu}[l]; \\
\text{tl}[2] & := \text{zv}[l]; \\
\text{cexp}(t[2], t[1]); \\
\text{tl}[1] & := 2 \times t[2]; \\
\text{tl}[2] & := 2 \times t[2]; \\
\end{align*}
\]
for i:=1 to npts do
begin
theta:=(i-1)/(npts-1)*2*pi;
cart(z2,theta);
csub(t2,t1,z2);
cinv(z1,t2);
data[1,i+(i-1)*npts]:=z1[1];
data[2,i+(i-1)*npts]:=z1[2];
end;
end;

Call to graphing routine

clrs cr;
write ln;
title:=' Root Trajectories ';
graph2d(data,m,title);
clrs cr;

End of Mainline Program

end.

end.
program chebtype1;

{ ***********************************************
  # This program calculates the expanded transfer
  # function of a two dimensional digital filter derived
  # from the second chebyshev argument
  # Program By - Tim Kent  Date Jul.87
  # Language - Turbo Pascal 3.0
  # Filter Theory - Dr. J. Soltis , T. Kent
  ***********************************************}

const
ordmax=20;

var
num, amax, sum, eps, n, fact0, gamma, sign,
c1, x, y, s, t, w, t2, m, n, p, pl, p2, scale : real;
I, j, k, l, m, nextx, nexty, nextx, nexty : integer;
u, v, cu, cv, zu, zv : array [1..ordmax] of real;
z1, z2, tl, t2 : form3;
flag : boolean;
temp : char;
filename : string[13];
output : text;
a, b, c, d : array [0..ordmax, 0..ordmax] of real;

{********************
  Subprograms
  ********************}

procedure pause;

{****************************
  Procedure to temporarily halt program execution
  ****************************}

begin
  writeln;
  writeln('press any key to continue');
  repeat until keypressed;
  writeln;
end;

procedure cexp(var x, y : real);

{*************************
  Procedure to simulate complex exponentiation
  *************************}

begin
  writeln;
end;
55 \text{var} \\
56 \text{s, t : real;} \\
57 \text{begin} \\
58 \text{t:=x;} \\
59 \text{s:=y;} \\
60 \text{x:=exp(t)\#cos(s);} \\
61 \text{y:=exp(t)\#sin(s);} \\
62 \text{end;} \\
63 \text{procedure cmult(var z1,z2,z3:form3);} \\
64 \{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
65 \text{Procedure to simulate complex multiplication}
66 \{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
67 \text{begin} \\
68 \text{z1[1]:=z2[1]\#z3[1]-z2[2]\#z3[2];} \\
69 \text{z1[2]:=z2[1]\#z3[2]+z2[2]\#z3[1];} \\
70 \text{end;} \\
71 \text{procedure arccosh(var z1,z2:form3);} \\
72 \{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
73 \text{Procedure to calculate arccosh of z}
74 \{~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
75 \text{var} \\
76 \text{x,y,b,c,d,e,f,g,t,sign : real;} \\
77 \text{begin} \\
78 \text{x:=z1[1];} \\
79 \text{y:=z1[2];} \\
80 \text{b:=x\#x-y\#y-1;} \\
81 \text{c:=2\#x\#y;} \\
82 \text{d:=x\#sqrt((sqrt(b\#b)+c\#c))/2);} \\
83 \text{if c>0 then} \\
84 \text{sign:=1;} \\
85 \text{else} \\
86 \text{sign:=-1;} \\
87 \text{e:=y\#sign\#sqrt((sqrt(b\#b)+c\#c)-b)/2);} \\
88 \text{f:=ln(sqrt(d\#d+e\#e)));} \\
89 \text{if ((abs(e)<1e-100)and(d<0.0)) then} \\
90 \text{t:=pi} \\
91 \text{else} \\
92 \text{begin} \\
93 \text{if abs(d)<1e-100 then} \\
94 \text{begin} \\
95 \text{if e>0 then} \\
96 \text{sign:=1;} \\
97 \text{else} \\
98 \text{sign:=-1;} \\
99 \text{t:=pi/2\#sign;} 

else
begin
  t:=arctan(e/d);
  if d<0 then
    if e>0 then
      t:=t+pi
    else
      t:=t-pi;
  end;
end;
g:=t;
z2[1]:=f;
z2[2]:=g;
end;

(***************
End of subprograms
***************

***************
Mainline Program
***************

begin
cirscr;

(***************
Input filter design parameters
***************)
write('Expanded Transfer Function Generator');
write('Chebyshev Argument #2');
write('Enter order of filter (+ve < ordmax,\'/2 \': \');
readln(n);
m:=trunc(2*n);
write('Enter passband ripple Amax : \');
readln(amax);
write('Enter Gamma (0<gamma<1.0) : \');
readln(gamma);
c1:=sin(gamma*pl/2)*sin(gamma*pl/2);
write;
eps:=sqrt(exp(0.1*amax*ln(10))-1);

(***************
Calculation of chebyshev poles
***************)
fact0:=-ln(1/eps+sqrt(1/eps/eps+1))/n;
for k:=1 to m do
begin
  u[k]:=(2*k-1)*pl/2/n;
Calculation of Cosine of chebyshev poles

```
for k:=1 to m do
begin
  cu[k]:=0.5*cos(u[k])*(exp(v[k])+exp(-v[k]));
  cv[k]:=-0.5*sin(u[k])*(exp(v[k])-exp(-v[k]));
end;
```

Calculation of Zsl — \(( \text{Arccosh}(z) )\)

```
for k:=1 to m do
begin
  cu[k]:=c1*cu[k];
  cv[k]:=c1#cv[k];
  z1[1]=-1-cu[k];
  z1[2]=-cv[k];
  arccosh(z1[1],z2);
  zu[k]:=z2[1];
  zv[k]:=z2[2];
end;
```

Calculation of constants in filter polynomial factors

```
for l:=1 to m do
begin
  zu[l]:=zu[l];
  zv[l]:=zv[l];
  cexp(zu[l],zv[l]);
  zu[l]=-2.0*zu[l];
  zv[l]=-2.0*zv[l];
end;
```

Initialize arrays for polynomial expansion

```
for l:=0 to m do
for j:=0 to m do
begin
  a[l,j]:=0.0;
  b[l,j]:=0.0;
  c[l,j]:=0.0;
  d[l,j]:=0.0;
end;
```
end;
a[0,0]:=1.0;

{*******************************************************************************
Loop to generate expanded polynomial
*******************************************************************************}

for k:=1 to m do
begin
    for l:=1 to k do
        for j:=0 to k do
            begin
                l:=l-1;
                c[l,j]:=c[l,j]+a[l,j];
                d[l,j]:=d[l,j]+b[l,j];
                c[l,j]:=c[l,j]+a[l,j];
                d[l,j]:=d[l,j]+b[l,j];
            end;
    for t:=0 to k do
        for j:=0 to k do
            begin
                c[l,j]:=c[l,j]+a[l,j]*zv[k]-b[l,j]*zu[k];
                d[l,j]:=d[l,j]+a[l,j]*zu[k]+b[l,j]*zv[k];
            end;
    for l:=0 to k do
        for j:=0 to k do
            begin
                a[l,j]:=c[l,j];
                b[l,j]:=d[l,j];
                c[l,j]:=0.0;
                d[l,j]:=0.0;
            end;
end;
num:=1.0/a[0,0];
for l:=0 to m do
    for j:=0 to m do
        a[l,j]:=a[l,j]*num;
writeln(' Do you wish amplitude response?');
write(' scaled to 0 dB at w1T = w2T = 0.0 ? (Y/N)');
readln(temp);
temp:=upcase(temp);
if temp='Y' then
    begin
        sum:=0.0;
        for l:=0 to m do
            for j:=0 to m do
                sum:=sum+a[l,j];
        num:=sum;
    end;
if temp='N' then
    begin
        writeln(' Do you wish amplitude response?');
        write(' scaled to 0 dB at w1T = w2T = pi ? (Y/N)');
        readln(temp);
temp:=upcase(temp);
if temp='Y' then
begin
  sum:=0.0;
  sign:=-1.0;
  for l:=0 to m do
  begin
    sign:=-sign;
    for j:=0 to m do
    begin
      sum:=sum+sign*a[l,j];
    sign:=-sign;
    end;
  end;
  num:=sum;
end;
end;
writeln;
write('Enter Filename for output of coefficients ');
readln(filename);
assign(output, filename);
rewrite(output);
writeln(output,m);
writeln(output,m);
writeln(output,num:12:11);
for l:=0 to m do
for j:=0 to m do
begin
  writeln(l,j,a[l,j]:12:11);
  writeln(output,a[l,j]:12:11);
end;
close(output);
}
writeln;
write(num);
writeln;
for l:=0 to m do
for j:=0 to m do
writeln(a[l,j]);
}
end.
APPENDIX 3

Second Chebyshev Low-Pass Filter
Response Program Source Code
program cheb3type1;
{
  function cheb3type1;
  ~
  ~
  * This program calculates the frequency, phase and delay~
  ~
  * response of a two dimensional digital filter derived~
  ~
  * from the third chebyshev argument~
  ~
  * Program By - Tim Kent Date Nov.86~
  ~
  * Language - Turbo Pascal 3.0~
  ~
  * Filter Theory - Dr. J. Soltis , T. Kent~
  ~
  ~
  ~
}

const
ordmax=20;
nptsx=25; { *** needed for graph3d.pas}
nptsy=25; {***}
npts=625; {***}
type
form =array [1..3,1..npts] of real; {***}
form2 =array [1..2,1..2] of real; {***}
form4 = string [40];
form3 =array [1..2] of real;
var
data,data2,delay ;form;
xy :form2;
amax,eps, fact0, gamma, c1, x, y, s, t, wt1, wt2, mag, pl, p2, scale :real;
phase, delta1 , delta2 , delpl, delp2, n :real;
l, j, k, m, nextx, nexty :integer;
u, v, cu, cv, zu, zv :array [1..ordmax] of real;
z1,z2,t1,t2,dp,t3,t4 :form3;
flag :boolean;
temp :char;
title :string[40];

[**********
Subprograms
**********]

{** graph3d.pas}

procedure cadd(var z1,z2,z3:form3);

{********** Procedure to simulate complex addition
**********}

begin
z1[1]:=z2[1]+z3[1];
z2[2]:=z2[2]+z3[2];
end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
procedure cexp(var x, y: real);

Procedure to simulate complex exponentiation

begin
  t := x;
  s := y;
  x := exp(t) * cos(s);
  y := exp(t) * sin(s);
end;

procedure cmult(var z1, z2, z3: form3);

Procedure to simulate complex multiplication

begin
end;

procedure arccosh(var z1, z2: form3);

Procedure to calculate arccosh of z

begin
  x := z1[1];
  y := z1[2];
  b := x * x - y * y - 1;
  c := 2 * x * y;
  d := x * sqrt((sqrt(b * b + c * c) + b) / 2);
  if c >= 0 then
    sign := 1
  else
    sign := -1;
  e := y * sign * sqrt((sqrt(b * b + c * c) - b) / 2);
  f := ln(sqrt(d * d + e * e));
  if ((abs(e) < 1e-100) and (d < 0.0)) then
    t := pi
  else
    begin
      if abs(d) < 1e-100 then
        begin

      end
      else
        begin

      end
end;
If e>0 then
  sign:=1
else
  sign:=-1;
end

if e>0 then
  t:=pi/2*sign;
else
  begin
    t:=arctan(e/d);
    if d<0 then
      if e>0 then
        t:=t+pi;
      else
        t:=t-pi;
      end;
  end;
g:=t;

z2[1]:=f;
z2[2]:=g;
end;

End of subprograms

End of program

Input filter design parameters

Calculation of chebyshev poles
fact0 := ln((1/eps + sqrt(1/eps^2 + 1)))/n/2;
for k := 1 to m do
begin
u[k] := (2*k - 1)*pi/4/n;
v[k] := fact0;
end;

Calculation of Cosine of Chebyshev poles
begin
for k := 1 to m do
begin
cu[k] := 0.5*cos(u[k])*(exp(v[k]) + exp(-v[k]));
cv[k] := 0.5*sin(u[k])*(exp(v[k]) - exp(-v[k]));
t1[k] := cu[k];
t2[k] := cv[k];
cmult(t2, t1, t1);
cu[k] := t2[k];
cv[k] := t2[k];
end;
end;

Calculation of Zsl — (Arccosh(z))
begin
for k := 1 to m do
begin
cu[k] := cl*cu[k];
cv[k] := cl*cv[k];
z1[k] := 1 - cu[k];
z2[k] := cv[k];
arccosh(z1, z2);
zu[k] := z2[k];
zv[k] := z2[k];
end;
end;

Setting of x-y-axis limits for plotting
begin
xy[1,1] := 0.0;
xy[1,2] := pi;
xy[2,1] := 0.0;
xy[2,2] := pi;
end;

Selection of Table of Values
begin
flag := false;
write('Do you want a table of values for magnitude and phase (Y/N)?');
readln(temp);
end;
temp:=Upcase(temp);
if temp='Y' then
begin
  flag:=true;
end;
if flag then
begin
  writeln;
  writeln('Frequency Response - Magnitude and Phase');
  writeln(' w1*T w2*T [H(z1,z2)] [H(z1,z2)');
  writeln(' (dB) (rad)');
  writeln;
end;

Calculation of constants in filter polynomial factors

for l:=1 to m do
begin
  zu1[l]:=zu[l];
  zv1[l]:=zv[l];
cexp(zu1[l],zv1[l]);
  zu1[l]:=-2.0*zu1[l];
  zv1[l]:=-2.0*zv1[l];
end;

Loop to generate filter magnitude response

for l:=1 to nptsx do
begin
  for j:=1 to nptsy do
  begin
    wt1:=((j-1)*(xy[2,2]-xy[1,1])/(nptsy-1)+xy[1,1];
    wt2:=((j-1)*(xy[2,2]-xy[1,2])/(nptsy-1)+xy[2,1];
    z1[l]:=cos(-wt1);
    z2[l]:=sin(-wt1);
    z1[l]:=cos(-wt2);
    z2[l]:=sin(-wt2);
    dp[l]:=1.0;
    dp[2]:=0.0;
    cadd(t1,z1,z2);
  end;

Loop to multiple filter polynomial factors

for k:=1 to m do
begin
  t2[l]:=zu[k];
  t2[2]:=zv[k];
  cadd(t3,t1,t2);
cmult(t4,dp,t3);
dp[1]=t4[1];
dp[2]=t4[2];
end;

{*****************************************************************************
  Calculation of magnitude in dB with dc scaling
*****************************************************************************}
k:=(1-1)*npts*j;
if k=1 then scale:=mag;
  mag:=-20*ln(mag/scale)/ln(10);

{*****************************************************************************
  Calculation of phase from -2pi - 0
*****************************************************************************}
phase:=-1.0*arctan(dp[2]/dp[1]);
if dp[1]<0.0 then
  phase:=-pl+phase
else
  begin
    if dp[2]<0.0 then phase:=-2*pl+phase;
  end;
data[1,k]:=wt1;
data[2,k]:=wt2;
data[3,k]:=mag;
data2[1,k]:=wt1;
data2[2,k]:=wt2;
data2[3,k]:=phase;
if flag then
  begin
    writeln(wt1:10:6,' ',wt2:10:6,' ',mag:14:6,' ',phase:14:6);
  end;
data[1,k]:=wp;
data[2,k]:=wp;
data[3,k]:=wp;
data2[1,k]:=wp;
data2[2,k]:=wp;
data2[3,k]:=wp;
writein;

{*****************************************************************************
  Call to graphing procedure
*****************************************************************************}
c1rscr;
title:=' Magnitude Response (in dB)';
graph3d(data,xy,title);
c1rscr;
title:=' Phase Response ';
graph3d(data2,xy,title);
c1rscr;

{*****************************************************************************
  Calculation of group delays of filter
*****************************************************************************}
Simulation of derivative by linear difference equation

\[ \text{delta1} = \frac{(xy[1,2]-xy[1,1])}{nptsx}; \]
\[ \text{delta2} = \frac{(xy[2,2]-xy[2,1])}{nptsy}; \]
for \( i = 1 \) to \( nptsx \) do
begin
\[ \text{wt1} = \frac{(i-1) \times (xy[1,2]-xy[1,1])}{nptsx-1} + xy[1,1]; \]
for \( j = 1 \) to \( nptsy \) do
begin
\[ \text{wt2} = \frac{(j-1) \times (xy[2,2]-xy[2,1])}{nptsy-1} + xy[2,1]; \]
\[ k = (1-1) \times nptsy + j; \]
\[ \text{nextx} = k + nptsy; \]
\[ \text{nexty} = k + 1; \]
if \( \text{nexty} > nptsy \) then \( \text{nexty} = \text{nexty} - 2; \)
if \( \text{nextx} > npts \) then \( \text{nextx} = \text{nextx} - 2 \times nptsy; \)
data[1,k] = wt1;
delay[1,k] = wt1;
data[2,k] = wt2;
delay[2,k] = wt2;
delp1 = data2[3,\text{nextx}] + data2[3,k];
delp2 = -data2[3,\text{nexty}] + data2[3,k];
end;
end;

Scaling of vertical change elements for periodic wrap-around

\[ \text{delp1} = \text{delp1} / \text{delta1}; \]
\[ \text{delay}[3,k] = \text{delp2} / \text{delta2}; \]
if \( j \times npts \) then \( \text{delay}[3,k] = \text{delay}[3,k]; \)
if \( i \times nptsx \) then \( \text{data}[3,k] = \text{data}[3,k]; \)
if \( \text{data}[3,k] < 10.0 \) then \( \text{data}[3,k] = \text{data}[3,k] + 10 \times \text{delta1}; \)
if \( \text{delay}[3,k] < 10.0 \) then \( \text{delay}[3,k] = \text{delay}[3,k] + 10 \times \text{delta2}; \)
end;
end;

Call to Graphing Subroutines
title: 'Group Delay 1';
graph3d(data xy, title);
clrscr;
title: 'Group Delay 2';
graph3d(dely xy, title);
clrscr;

End of mainline program

end.
program cheb3type1root;
{
    ******************************
    *
    * This program calculates the root trajectory *
    * of a two dimensional digital filter derived *
    * from the third chebyshev argument *
    *
    * Program By - Tim Kent      Date Nov.86 *
    * Language - Turbo Pascal 3.0 *
    *
    * Filter Theory - Dr. J. Soltis , T. Kent *
    *
    ******************************}

const
ordmax=20;
npts=150;  (**-Needed for graph2d)
nptsord=3000;
type
form =array [1..2,1..nptsord] of real;  (**
form3 =array [1..2] of real;
form4 = string [40];

var
data :form;  (**
amax,eps,fact0,gamma,cl,x,y,s,t,p1,p2,theta,n :real;
l,j,k,m,l :integer;
u,v,cu,cv,zu,zv :array [1..ordmax] of real;
z1,z2,t1,t2 :form3;
title :string[40];

{************
Subprograms
************}

{file graph2d.pas}
procedure csub(var z1,z2,z3:form3);
{
    Procedure to simulate complex subtraction
    **************************************
begin
    z1[1]:=z2[1]-z3[1];
z1[2]:=z2[2]-z3[2];
end;

procedure cmult(var z1,z2,z3:form3);
{
    Procedure to simulate complex multiplication
}
procedure clnv(var zl,z2:form3);
{**********************************************************************
Procedure to simulate complex inversion
**********************************************************************}
var
mag:real;
begin
mag:=z2[1]*z2[1]+z2[2]*z2[2];
zl[1]:=z2[1]/mag;
zl[2]:=-z2[2]/mag;
end;

procedure cexp(var zl,z2:form3);
{**********************************************************************
Procedure to simulate complex exponentiation
**********************************************************************}
begin
zl[1]:=exp(z2[1])*cos(z2[2]);
zl[2]:=exp(z2[1])*sin(z2[2]);
end;

procedure cart(var zl:form3; thetarreal);
{**********************************************************************
Procedure to perform cartesian to polar conversion
**********************************************************************}
begin
zl[1]:=cos(theta);
zl[2]:=sin(theta);
end;

procedure arccosh(var zl,z2:form3);
{**********************************************************************
Procedure to calculate arccosh of z
**********************************************************************}
var
x,y,b,c,d,e,f,g,t,sign:real;
begin
x:=zl[1];
\begin{verbatim}
109  y:=z[2];
110  b:=x*x-y*y-1;
111  c:=2*x*y;
112  d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
113  if c>0 then
114     sign:=1
115  else
116     sign:=-1;
117  e:=y*sign*sqrt((sqrt(b*b+c*c)-b)/2);
118  f:=ln(sqrt(d#d+e*e));
119  if ((abs(e)<le-100)and(d<0.0)) then
120     t:=pi
121  else
122     begin
123        if abs(d)<le-100 then
124          begin
125             if e>0 then
126                sign:=1
127             else
128                sign:=-1;
129             t:=pi/2*sign;
130          end
131        else
132          begin
133             t:=arctan(e/d);
134             if d<0 then
135               if e>0 then
136                  t:=t+pi
137               else
138                  t:=t-pi;
139          end;
140     end;
141  g:=t;
142  z2[1]:=f;
143  z2[2]:=g;
144  end;
145
146  (***************
147  End of subprograms
148  ********************
149  ****************************
150  ****************************
151  ****************************
152  Mainline program
153  **************************
154  **************************
155
156  begin
157     clrscr;
158
159  (*************************
160  Input filter specifications
161  **************************
162
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\end{verbatim}
 whiteln('Root Trajectory - Type 1 - Chebyshev Argument #3'); 
164 writeln; 
165 write('Enter order of filter (+ve ordmax, ordmax/2); '); 
166 readln(n); 
167 m:='trunc(2^n); 
168 write('Enter passband ripple Amax: '); 
169 readln(amax); 
170 write('Enter Gamma (0<gamma<1.0): '); 
171 readln(gamma); 
172 cl:='sin(gamma*pi/2)*sin(gamma*pi/2); 
173 writeln; 
174 eps:='sqrt(exp(0.1*amax*ln(10))-1); 
175 [**********] 
176 Calculate chebyshev poles 
177 [**********] 
178 fact0:='ln(1/eps+sqrt(1/eps/eps+1))/n/2; 
179 for k:='1 to m do 
180 begin 
181 u[k]:='(2*k-1)*pi/4/n; 
182 v[k]:='fact0; 
183 end; 
184 [**********] 
185 Calculate cosine of cheb poles 
186 [**********] 
187 for k:='1 to m do 
188 begin 
189 cu[k]:='0.5*cos(u[k])*(exp(v[k])+exp(-v[k])); 
190 cv[k]:='0.5*sin(u[k])*(exp(v[k])-exp(-v[k])); 
191 t1[1]:='cu[k]; 
192 t1[2]:='cv[k]; 
193 cmult(t2,t1,t1); 
194 cu[k]:='t2[1]; 
195 cv[k]:='t2[2]; 
200 end; 
201 [**********] 
202 Calculate zsis - arccosh(z) 
203 [**********] 
204 for k:='1 to m do 
205 begin 
206 cu[k]:='cl*cu[k]; 
207 cv[k]:='cl*cv[k]; 
208 z1[1]:='1-cu[k]; 
209 z1[2]:='cv[k]; 
210 arccosh(z1,z2); 
211 zu[k]:='z2[1]; 
212 zv[k]:='z2[2]; 
215 end; 
216
Loop to generate root trajectories for all factors

```plaintext
for 1:=1 to m do
begin
  t1[1]:=zu[1];
  t1[2]:=zv[1];
  cexp(t2,t1);
  t1[1]:=2*t2[1];
  t1[2]:=2*t2[2];
end;
```

Loop to generate points of trajectories

```plaintext
for 1:=1 to npts do
begin
  theta:=(1-1)/(npts-1)*2*pi;
  cart(z2,theta);
  csub(t2,t1,z2);
  cinv(z1,t2);
  data[1,1+(1-1)*npts]:=z1[1];
  data[2,1+(1-1)*npts]:=z1[2];
end;
end;
```

Call to graphing routine

```plaintext
cirscr;
title:=" Root Trajectories ";
graph2d(data,m,title);
cirscr;
```

End of Mainline Program

```plaintext
eend.
```
program cheb3type1;
{
 ************************************************************
 *
 * This program calculates the expanded transfer
 * function of a two dimensional digital filter derived
 * from the third chebyshev argument
 *
 * Program By - Tim Kent Date Jul 87
 *
 * Language - Turbo Pascal 3.0
 *
 * Filter Theory - Dr. J. Soltis , T. Kent
 *
 ************************************************************}

const
ordmax=20;
type
form3 = array [1..2] of real;
var
num,emax,sum,eps,n, fact0, gamma, sign,
ct, x,y,s,t, wt1, wt2, msg, pl, p2, scale : real;
j, k, l, m, nextx, nexty : integer;
u, v, cu, cv, zu, zv : array [1..ordmax] of real;
z1, z2, tl, t2 : form3;
flag : boolean;
temp : char;
filename : string[13];
output : text;
a, b, c, d : array [0..ordmax, 0..ordmax] of real;

{ Subprograms

**********

procedure pause;

{ ***********************
 Procedure to temporarily halt program execution
 ***********************}

begin
 writeln;
 writeln('press any key to continue');
 repeat until keypressed;
 writeln;
 end;

procedure cexp(var x,y:real);

{ ***********************
 Procedure to simulate complex exponentiation
 ***********************}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
55  var
56    s,t:real;
57  begin
58    t:=x;
59    s:=y;
60    x:=exp(t)*cos(s);
61    y:=exp(t)*sin(s);
62  end;
63
64  procedure cmult(var z1,z2,z3:form3);
65  {***********************************************************************
66  Procedure to simulate complex multiplication
67  ***********************************************************************}
68  begin
69    z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
70    z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
71  end;
72
73  procedure arccosh(var z1,z2:form3);
74  {***********************************************************************
75  Procedure to calculate arccosh of z
76  ***********************************************************************}
77  begin
78    var
79      x,y,b,c,d,e,f,g,t,sign:real;
80    begin
81      x:=z1[1];
82      y:=z1[2];
83      b:=x^2-y*y-1;
84      c:=2*x*y;
85      d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
86      if c>0 then
87        sign:=1
88      else
89        sign:=-1;
90      e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
91      f:=ln(sqrt(d*d+e*e));
92      if ((abs(e)<le-100)and(d<0.0)) then
93        t:=pi
94      else
95        begin
96          if abs(d)<le-100 then
97            begin
98              if e>0 then
99                sign:=1
100               else
101                  sign:=-1;
102              t:=pi/2*sign
103            end
104          end
105        end
106    end
107  end;
else
begin
  t:=arctan(e/d);
  if d<0 then
    if e>0 then
      t:=t+π
    else
      t:=t-π;
  end;
end;

g:=t;
z2[1]:=f;
z2[2]:=g;

End of subprograms

Mainline Program

begin
  clrscr;

  Input filter design parameters

write('Expanded Transfer Function Generator');
write('Chebyshev Argument #3');
write:
write('Enter order of filter (+ve < ordmax,/2) : ');
readin(n);
m:=trunc(2*n);
write('Enter passband ripple Amax : ');
readin(amax);
write('Enter Gamma (0<gamma<1.0) : ');
readin(gamma);
c:=sin(gamma*pi/2)*sin(gamma*pi/2);
write:
eps:=sqrt(exp(0.1*amax*ln(10))-1);

Calculation of chebyshev poles

fact0:=ln(1/eps+sqrt(1/eps/eps+1))/n/2;
for k:=1 to m do
begin
  u[k]:=(2*k-1)*π/4/n:
v[k] := fact0;
end;

(*-----------------------------
Calculation of Cosine of chebyshev poles
*)

for k := 1 to m do
begin
 cu[k] := 0.5*cos(u[k])*(exp(v[k])+exp(-v[k]));
 cv[k] := -0.5*sin(u[k])*(exp(v[k])-exp(-v[k]));
 t[i][1] := cu[k];
 t[i][2] := cv[k];
cmult(t2,t1,t1);
cu[k] := t2[1];
cv[k] := t2[2];
end;

(*-----------------------------
Calculation of Zsl --- (Arccosh(z))
*)

for k := 1 to m do
begin
 cu[k] := c1*cu[k];
 cv[k] := c1*cv[k];
z[i][1] := 1 - cu[k];
z[i][2] := -cv[k];
arccosh(z[i][1],z2);
zu[k] := z2[1];
zv[k] := z2[2];
end;

(*-----------------------------
Calculation of constants in filter polynomial factors
*)

for i := 1 to m do
begin
 zu[i] := zu[i];
 zv[i] := zv[i];
cexp(zu[i],zv[i]);
zu[i] := -2.0*zu[i];
zv[i] := -2.0*zv[i];
end;

(*-----------------------------
Initialize arrays for polynomial expansion
*)

for i := 0 to m do
for j := 0 to m do

begin
  a[1,0]:=0.0;
  b[1,0]:=0.0;
  c[1,0]:=0.0;
  d[1,0]:=0.0;
end;
num:=-1.0/at0.0;

{*******************************************************************************
 Loop to generate expanded polynomial
*******************************************************************************}

for k:=1 to m do
begin
  for l:=1 to k do
  begin
    c[1,l]:=c[1,l]+a[1,l];
    d[1,l]:=d[1,l]+b[1,l];
    c[l,1]:=c[l,1]+a[l,1];
    d[l,1]:=d[l,1]+b[l,1];
  end;
  for l:=0 to k do
  begin
    c[1,l]:=c[1,l]+a[l,0]*zv[l]-b[1,l]*zu[l];
    d[1,l]:=d[l,0]+a[1,l]*zu[l]+b[1,l]*zv[l];
  end;
  for l:=0 to k do
  begin
    a[1,l]:=c[1,l];
    b[1,l]:=d[1,l];
    c[1,l]:=0.0;
    d[1,l]:=0.0;
  end;
end;

num:=1.0/a[0,0];
for i:=0 to m do
begin
  for j:=0 to m do
  begin
    a[i,j]:=a[i,j]*num;
    writeln('Do you wish amplitude response?');
    write(' scaled to 0 dB at wiT = w2T = 0.0? (Y/N)');
    readln(temp);
    temp:=upcase(temp);
    if temp='Y' then
    begin
      sum:=0.0;
      for i:=0 to m do
      begin
        for j:=0 to m do
        begin
          sum:=sum+a[i,j];
        end;
        num:=sum;
      end;
    end;
  end;
end;
if temp = 'N' then
begin
  writeln(' Do you wish amplitude response');
  writeln(' scaled to 0 dB at w1T = w2T = pi ? (Y/N)');
  readln(temp);
  temp := upcase(temp);
if temp = 'Y' then
begin
  sum := 0.0;
  sign := -1.0;
  for I := 0 to m do
    begin
      sign := -sign;
      for J := 0 to m do
        begin
          sum := sum + sign*a[I, J];
          sign := -sign;
        end;
    end;
  num := sum;
end;
end;
write In;
write(' Enter Filename for output of coefficients ');
readln(filename);
assign(output, filename);
rewrite(output);
writeln(output, m);
writeln(output, m);
writeln(output, num:12:11);
for I := 0 to m do
  for J := 0 to m do
    begin
      writeln(I, J, a[I, J]:12:11);
    end;
close(output);

writeln;
writeln(num);
writeln;
for I := 0 to m do
  for J := 0 to m do
    writeln(a[I, J]);
}
end.
APPENDIX 4

Third Chebyshev Low-Pass Filter Response Program Source Code
program cheb4type1;

{  *****************************************************************
    *    *
    *  This program calculates the frequency, phase and delay  *
    *  response of a two dimensional digital filter derived    *
    *  from the fourth (summer86) chebyshev argument           *
    *    *
    *  Program By - Tim Kent  Date Jun 87                      *
    *  Language - Turbo Pascal 3.0                            *
    *  Filter Theory - Dr. J. Soltis, T. Kent                 *
    *    *
    ******************************************************************

const
ordmax=20;
nptsx=25; (**- needed for graph3d.pas)
nptsy=25; (**)
npts=625; (**)
type
form  =array [1..3,1..npts] of real; (**) 
form2 =array [1..2,1..2] of real; (**) 
form4 = string[40];
form3 =array [1..2] of real;

var

data,data2,delysform; (**)
xy sform2; (**)
amax,eps,fact0,gamma,c1,wt1,wt2,mag,p1,p2,scale :real;
phase,delta1,delta2,delpl,delp2,n :real;
l,j,k,m,nextx,nexty :integer;
u,v,cu,cv,zu,zv :array [1..ordmax] of real;
z1,z2,t1,t2,dp,t3,t4 :form3;
flag :boolean;
temp :char;
title :form4;


******
Subprograms
*******

{$I graph3d.pas}

procedure cadd(var zl,z2,z3:form3);

{***********************************************
 Procedure to simulate complex addition
 ***********************************************}

begin
zl[I]:=z2[I]+z3[I];
zl[2]:=z2[2]+z3[2];
end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
procedure cexp(var x,y:real);

(*---------------------------------------------------------------
  Procedure to simulate complex exponentiation
  ---------------------------------------------------------------*)

var
  t,s :real;
begin
  t:=x;
  s:=y;
  x:=exp(t)*cos(s);
  y:=exp(t)*sin(s);
end;

procedure cmult(var z1,z2,z3:form3);

(*---------------------------------------------------------------
  Procedure to simulate complex multiplication
  ---------------------------------------------------------------*)

begin
  z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
  z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
end;

procedure arccosh(var z1,z2:form3);

(*---------------------------------------------------------------
  Procedure to calculate arccosh of z
  ---------------------------------------------------------------*)

var
  x,y,b,c,d,e,f,g,t,s :real;
begin
  x:=z1[1];
  y:=z1[2];
  b:=x*x-y*y-1;
  c:=2*x*y;
  d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
  if c>0 then
    sign:=1
  else
    sign:=-1;
  e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
  f:=ln(sqrt(d*d+e*e));
  if ((abs(e)<le-100)and(d<0.0)) then
    t:=pi;
  else
    begin
      if abs(d)<le-100 then
        begin
          if abs(e)<le-100 then
            begin
            end
          end
        end
      end
    end
  end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
If e > 0 then
  sign := 1
else
  sign := -1;
end

t := π/2 * sign;
end
else
begin
  t := arctan(e/d);
  if d < 0 then
    if e > 0 then
      t := t + π
    else
      t := t - π;
  end;
end;
g := t;
z2[1] := f;
z2[2] := g;
end;

BEGIN
  CLEARSCR;
  [*********]
  Input filter design parameters
  [*********]
  writeln('Frequency Response - Chebyshev Argument #4');
  writeln;
  writeln('Enter order of filter (+ve < ordmax, \pi/2) : '); readln(n);
  m := trunc(2*n);
  writeln('Enter passband ripple Amax : '); readln(amax);
  readln(gamma);
  cl := sin(gamma*pi/2)*sin(gamma*pi/2);
  writeln;
  eps := sqrt(exp(0.1*amax*ln(10))-1);
  [*********]
  Calculation of chebyshev poles
  [*********]
```
163  fact0:=-ln(1/eps+sqrt(1/eps/eps+1))/n/2;
164  for k:=1 to m do
165     begin
166      u[k]:=(2*k-1)*pi/4/n;
167      v[k]:=fact0;
168     end;
169  [******************************************]
170  Calculation of sine of chebyshev poles
171  [******************************************]
172  for k:=1 to m do
173     begin
174       cu[k]:=0.5*sin(u[k])*(exp(v[k])+exp(-v[k]));
175       cv[k]:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
176       t1[1]:=cu[k];
177       t1[2]:=cv[k];
178       cmult(t2,t1,t1);
179       cu[k]:=t2[1];
180       cv[k]:=t2[2];
181     end;
182
183  [******************************************]
184  Calculation of Zsl --- ( Arccosh(z))
185  [******************************************]
186  for k:=1 to m do
187     begin
188       cu[k]:=c1*cu[k];
189       cv[k]:=c1*cv[k];
190       z1[1]:=1-cu[k];
191       z1[2]:=-cv[k];
192       arccosh(z1,z2);
193       zu[k]:=z2[1];
194       zv[k]:=z2[2];
195     end;
196
197  [******************************************]
198  Setting of x-y-axis limits for plotting
199  [******************************************]
200  xy[1,1]:=0.0;
201  xy[1,2]:=pi;
202  xy[2,1]:=0.0;
203  xy[2,2]:=pi;
204
205  [******************************************]
206  Selection of Table of Values
207  [******************************************]
208  flag:=false;
209  write('Do you want a table of values for magnitude and phase (Y/N)');
210  readin(temp);
```
temp:=Upcase(temp);
If temp='Y' then
begin
flag:=true;
end;
If flag then
begin
writeln;
writeln('Frequency Response - Magnitude and Phase');
writeln(' w1*T  w2*T  [H(z1,z2)]  <H(z1,z2)');
writeln(' (dB)  (rad)');
writeln;
end;

Calculation of constants in filter polynomial factors

for i=1 to m do
begin
zu[i]:=zu[i];
zv[i]:=zv[i];
exp(zu[i],zv[i]);
zu[i]:=-2.0*zu[i];
zv[i]:=-2.0*zv[i];
end;

Loop to generate filter magnitude response

for l=1 to nptsx do
begin
for j=1 to nptsy do
begin
wt1:=(l-1)*(xy[1,2]-xy[1,1])/(nptsx-1)+xy[1,1];
wt2:=(j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)+xy[2,1];
z1[1]:=cos(-wt1);
z1[2]:=sin(-wt1);
z2[1]:=cos(-wt2);
z2[2]:=sin(-wt2);
dp[1]:=1.0;
dp[2]:=0.0;
cadd(t1,z1,z2);
end;
end;

Loop to multiple filter polynomial factors

for k=1 to m do
begin
t2[1]:=zu[k];
t2[2]:=zv[k];
cadd(t3,t1,t2);
cmult(t4,dp.t3);
dp[1]:=t4[1];
dp[2]:=t4[2];
end;

{*******************************************************************************
Calculation of magnitude in dB with dc scaling
*******************************************************************************}
k:=(l-1)*nptsy+J;
mag:=(dp[1]*dp[1]+dp[2]*dp[2]);
if k=l then scale:=mag;
mag:=-20*ln(mag/scale)/ln(10);

{*******************************************************************************
Calculation of phase from -2pi - 0
*******************************************************************************}
phase:=-1.0*arctan(dp[2]/dp[1]);
if dp[1]<0.0 then
  phase:=-pi+phase
else
  begin
    if dp[2]<0.0 then phase:=-2*pi+phase;
  end;
data[1,k]:=wt1;
data[2,k]:=wt2;
data[3,k]:=mag;
data2[1,k]:=wt1;
data2[2,k]:=wt2;
data2[3,k]:=phase;
if flag then
  begin
    writeln(wt1:10:6,' ',wt2:10:6,' ',mag:14:6,' ',phase:14:6);
  end;

{*******************************************************************************
Call to graphing procedure
*******************************************************************************}
clrscr;
title:=" Magnitude Response (In dB)";
graph3d(data,xy,title);
clrscr;
title:=" Phase Response ";
graph3d(data2,xy,title);
clrscr;

{*******************************************************************************
Calculation of group delays of filter
*******************************************************************************}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Simulation of derivative by linear difference equation

\[
\Delta \text{val} = \frac{(y_{2,2} - y_{2,1})}{n_{ptsy}}; \\
\Delta^2 \text{val} = \frac{(y_{2,2} - y_{2,1})}{n_{ptsy}}; \\
\text{for } l = 1 \text{ to } n_{ptsx} \text{ do} \\
\begin{align*}
\text{wt1} &= (l-1) \frac{(y_{1,2} - y_{1,1})}{(n_{ptsx}-1)} + y_{1,1}; \\
\text{for } j = 1 \text{ to } n_{ptsy} \text{ do} \\
\text{wt2} &= (j-1) \frac{(y_{2,2} - y_{2,1})}{(n_{ptsy}-1)} + y_{2,1}; \\
k &= (l-1)n_{ptsy} + j; \\
\text{nextx} &= k + n_{ptsy}; \\
\text{nexty} &= k + 1; \\
\text{if } \text{nexty} > n_{ptsy} \text{ then } \text{nexty} &= \text{nexty} - 2; \\
\text{if } \text{nextx} > n_{pts} \text{ then } \text{nextx} &= \text{nextx} - 2n_{ptsy}; \\
\text{data}[1,k] &= \text{wt1}; \\
\text{dey}[1,k] &= \text{wt1}; \\
\text{data}[2,k] &= \text{wt2}; \\
\text{dey}[2,k] &= \text{wt2}; \\
\text{delp1} &= \text{data}[3, \text{nextx}] + \text{data}[3,k]; \\
\text{delp2} &= \text{data}[3, \text{nexty}] + \text{data}[3,k]; \\
\end{align*}
\]

Scaling of vertical change elements for periodic wrap-around

\[
\text{if } \text{abs(delp1)} > \text{pi} = 1.50 \text{ then} \\
\begin{align*}
\text{begin} \\
\text{if } \text{delp1} > 0.0 \text{ then } \\
\text{delp1} &= \text{delp1} - 2 \times \text{pi} \\
\text{else} \\
\text{delp1} &= \text{delp1} + 2 \times \text{pi}; \\
\text{end}; \\
\text{if } \text{abs(delp2)} > \text{pi} = 1.50 \text{ then} \\
\begin{align*}
\text{begin} \\
\text{if } \text{delp2} > 0.0 \text{ then } \\
\text{delp2} &= \text{delp2} - 2 \times \text{pi} \\
\text{else} \\
\text{delp2} &= \text{delp2} + 2 \times \text{pi}; \\
\text{end}; \\
\text{data}[3,k] &= \text{delp1}/\text{deltal}; \\
\text{dey}[3,k] &= \text{delp2}/\text{deltal}; \\
\text{if } j > n_{ptsy} \text{ then } \text{dey}[3,k] &= \text{dey}[3,k]; \\
\text{if } l > n_{ptsx} \text{ then } \text{dey}[3,k] &= \text{dey}[3,k]; \\
\text{if } \text{data}[3,k] < -10.0 \text{ then } \text{data}[3,k] &= \text{data}[3,k] + 2 \times \text{pi}/\text{deltal}; \\
\text{if } \text{dey}[3,k] < -10.0 \text{ then } \text{dey}[3,k] &= \text{dey}[3,k] + 2 \times \text{pi}/\text{deltal}; \\
\text{end}; \\
\end{align*}
\]

Call to Graphing Subroutines

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
clscr;
title:=' Group Delay 1';
graph3d(data,xy,title);
clscr;
title:=' Group Delay 2';
graph3d(dely,xy,title);
clscr;

End of mainline program

end.
program cheb4typeIroot;
{

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

This program calculates the root trajectory of a two dimensional digital filter derived from the fourth chebyshev argument

Program By - Tim Kent Date Jun87

Language - Turbo Pascal 3.0

Filter Theory - Dr. J. Soltis, T. Kent

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

const
ordmax=20;

npts=150; [**-Needed for graph2d]
nptsord=3000;

program cheb4typeIroot;
{

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

This program calculates the root trajectory of a two dimensional digital filter derived from the fourth chebyshev argument

Program By - Tim Kent Date Jun87

Language - Turbo Pascal 3.0

Filter Theory - Dr. J. Soltis, T. Kent

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

const
ordmax=20;

npts=150; [**-Needed for graph2d]
nptsord=3000;

type

form =array [1..2,1..nptsord] of real; [**]
form3 =array [1..2] of real;
form4 = string[40];

var
data :form; [**]

amax,eps,fact0, gamma, cl,x,y,s,t,p1,p2,theta,n :real;
l,j,k,m,l :integer;
u,v, cu,cv,zu,zv :array [1..ordmax] of real;
z1,z2,t1,t2 :form3;
title :form4;

{************

Subprograms

************}

{$1 graph2d.pas}

procedure csub(var z1,z2,z3:form3);
{

Procedure to simulate complex subtraction

**************}

begin

z1[1]:=z2[1]-z3[1];

z1[2]:=z2[2]-z3[2];

end;

procedure cmult(var z1,z2,z3:form3);
{

Procedure to simulate complex multiplication

```plaintext
procedure clnv(ver z1,z2:form 3);
{*****************************************************************************
  Procedure to simulate complex Inversion
*****************************************************************************}
begin
  mag := z2[1]*z2[1]+z2[2]*z2[2];
  z1[1] := z2[1]/mag;
end;

procedure cexp(var z1,z2:form 3);
{*****************************************************************************
  Procedure to simulate complex exponentiation
*****************************************************************************}
begin
  z1[1] := exp(z2[1])*cos(z2[2]);
  z1[2] := exp(z2[1])*sin(z2[2]);
end;

procedure cart(var zl:form 3; theta:real);
{*****************************************************************************
  Procedure to perform cartesian to polar conversion
*****************************************************************************}
begin
  z1[1] := cos(theta);
  z1[2] := sin(theta);
end;

procedure arccosh(var zl,z2:form 3);
{*****************************************************************************
  Procedure to calculate arccosh of z
*****************************************************************************}
var
  x,y,b,c,d,e,f,g,t,sign :real;
begin
  x := zl[1];
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
y:=z[2];
b:=x*y-y-1;
c:=2*x*y;
d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
if c>0 then
  sign:=1
else
  sign:=-1;
e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
f:=ln(sqrt(d*d+e*e));
if ((abs(e)<le-100)and(d<0.0)) then
  t:=pi
else
begin
  if abs(d)<le-100 then
    begin
      if e>0 then
        sign:=1
      else
        sign:=-1;
      t:=pi/2*sign;
    end
  else
    begin
      t:=arctan(e/d);
      if d<0 then
        if e>0 then
          t:=t+pi
        else
          t:=-t-pl;
    end;
end;
g:=t;
z2[1]:=t;
z2[2]:=g;
end;

{**********************************************************
** End of subprograms
**********************************************************
}

Mainline program

begin
clrscr;

{**********************************************************
** Input filter specifications
**********************************************************
}
writein('Root Trajectory - Chebyshev Argument #4');
write('Enter order of filter (+ve < ordmax, \text{ordmax} = 1/2): ');
readln(n);
m:=trunc(2*n);
write('Enter passband ripple \text{Amax}; ');
readln(amax);
write('Enter Gamma (0 < \gamma < 1.0): ');
readln(gamma);
c1:=sin(gamma*pi/2)*sin(gamma*pi/2);
writein;
eps:=-sqrt(exp(0.1*amax*ln(10))-1);

Calculate chebyshev poles

fact0:=-ln(1/eps+sqrt(1/eps/eps+1))/n/2;
for k:=1 to m do
begin
u[k]:=(2*k-1)*pi/4/n;
v[k]:=fact0;
end;

Calculate sine of cheb poles

for k:=1 to m do
begin
cu[k]:=0.5*sin(u[k])*(exp(v[k])+exp(-v[k]));
cv[k]:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
t[1]:=cu[k];
t[2]:=cv[k];
cmult(t2,t1,t1);
cu[k]:=t2[1];
cv[k]:=t2[2];
end;

Calculate zs - arccosh(z)

for k:=1 to m do
begin
cu[k]:=c1*cu[k];
cv[k]:=c1*cv[k];
z[1]:=1-cu[k];
z[2]:=-cv[k];
arccosh(z1,z2);
zu[k]:=z2[1];
zu[k]:=z2[2];
end;
Loop to generate root trajectories for all factors

```
for i=1 to m do begin
  t1[i]:=zu[i];
  t1[2]:=zv[i];
  cexp(t2,t1);
  t1[i]:=2*t2[i];
  t1[2]:=2*t2[2];
end;
```

Loop to generate points of trajectories

```
for i=1 to npts do begin
  theta:=(1/(npts-1)*2*pi;
  cart(z2,theta);
  csub(t2,t1,z2);
  clinv(z1,t2);
  data[1,i+(i-1)*npts]:=z1[i];
  data[2,i+(i-1)*npts]:=z1[2];
end;
```

Call to graphing routine

```
c1rscr;
title:=' Root Trajectories ';
graph2d(data,m,title);
c1rscr;
```

End of Mainline Program

```
end.
```
program cheb4type;
{
*****************************************************************************
  *
  * This program calculates the expanded transfer function of a two
dimensional digital filter derived from the fourth chebyshev argument
  *
  * Program By - Tim Kent       Date Jul.87
  *
  * Language - Turbo Pascal 3.0
  *
  * Filter Theory - Dr. J. Soltis, T. Kent
  *
*****************************************************************************
}
const
ordmax=20;
type
form3 =array [1..2] of real;
var
num,ammax,sum,eps,n, fact0, gamma,sign,
c1,x,y,s,t,wt1,wt2, mag, p1, p2, scale : real;
l,j,k,i,m,nextx,nexty : integer;
u,v, cv, zu, zv : array [1..ordmax] of real;
z1,z2,t1,t2 : form3;
flag :boolean;
temp :char;
filename :string[13];
output :text;
a,b,c,d : array [0..ordmax,0..ordmax] of real;

Subprograms
************
procedure pause;
{
***********************************************************************
Procedure to temporarily halt program execution
***********************************************************************
}
begin
  writeln;
  writeln('press any key to continue');
  repeat until keypressed;
  writeln;
end;
procedure cexp(var x,y:real);
{*****************************************************************************
Procedure to simulate complex exponentiation
*****************************************************************************
}
55  var
56    s,t : real;
57 begin
58    t:=x;
59    s:=y;
60    x:=exp(t)*cos(s);
61    y:=exp(t)*sin(s);
62 end;
63
64 procedure cmult(var z1,z2,z3: form3);
65 {*****************************************************
66 Procedure to simulate complex multiplication
67 *****************************************************}
68 begin
69    z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
70    z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
71 end;
72
73 procedure arccosh(var z1,z2:form3);
74 {*****************************************************
75 Procedure to calculate arccosh of z
76 *****************************************************}
77 var
78    x,y,b,c,d,e,f,g,t,sign: real;
79 begin
80    x:=z1[1];
81    y:=z1[2];
82    b:=x*x-y*y-1;
83    c:=2*x*y;
84    d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
85    if c>=0 then
86       sign:=1
87    else
88       sign:=-1;
89    e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
90    f:=ln(sqrt(d*d+e*e));
91    if ((abs(e)<le-100)and(d<0.0)) then
92       t:=pi
93    else
94       begin
95          if abs(d)<le-100 then
96             begin
97                if e>=0 then
98                   sign:=1
99                else
100                   sign:=-1;
101                t:=pi/2*sign;
102             end
103          end
104 end;
105
106 procedure cmult(var z1,z2,z3: form3);
107 {*****************************************************
108 Procedure to simulate complex multiplication
109 *****************************************************}
110 begin
111    z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
112    z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
113 end;
114
115 procedure arccosh(var z1,z2:form3);
116 {*****************************************************
117 Procedure to calculate arccosh of z
118 *****************************************************}
119 var
120    x,y,b,c,d,e,f,g,t,sign: real;
121 begin
122    x:=z1[1];
123    y:=z1[2];
124    b:=x*x-y*y-1;
125    c:=2*x*y;
126    d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
127    if c>=0 then
128       sign:=1
129    else
130       sign:=-1;
131    e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
132    f:=ln(sqrt(d*d+e*e));
133    if ((abs(e)<le-100)and(d<0.0)) then
134       t:=pi
135    else
136       begin
137          if abs(d)<le-100 then
138             begin
139                if e>=0 then
140                   sign:=1
141                else
142                   sign:=-1;
143                t:=pi/2*sign;
144             end
145          end
146 end;
147
148 procedure cmult(var z1,z2,z3: form3);
149 {*****************************************************
150 Procedure to simulate complex multiplication
151 *****************************************************}
152 begin
153    z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
154    z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
155 end;
156
157 procedure arccosh(var z1,z2:form3);
158 {*****************************************************
159 Procedure to calculate arccosh of z
160 *****************************************************}
else
begin
    t:=arctan(e/d);
    if d<0 then
        if e>0 then
            t:=t+pi
        else
            t:=t-pi;
    end;
end;
g:=t;
z2[1]:=f;
z2[2]:=g;
end;

end;

End of subprograms

Mainline Program

begin
clrscr;

Input filter design parameters

writeln('Expanded Transfer Function Generator');
writeln(' Chebyshev Argument #4');
writeln;
write('Enter order of filter (+ve <',ordmax,'/2) : ');
readln(n);
m:=trunc(2*n);
write('Enter passband ripple Amax : ');
readln(amax);
write('Enter Gamma (0<gamma<1.0) : ');
readln(gamma);
c1:=sin(gamma*pl/2)*sin(gamma*pl/2);
writeln;
eps:=sqrt(exp(0.1*amax*ln(10))-1);

Calculation of chebyshev poles

fact0:=-ln(1/eps+sqrt(1/eps/eps+1))/n/2;
for k:=1 to m do
begin
    u[k]:=(2*k-1)*pi/4/n;
end;
 Calculation of sine of chebyshev poles

for k = 1 to m do
begin
  cu[k] = 0.5 * sin(u[k]) * (exp(v[k]) + exp(-v[k]));
  cv[k] = 0.5 * cos(u[k]) * (exp(v[k]) - exp(-v[k]));
  t1[1] = cu[k];
  t1[2] = -cv[k];
  cmult(t2, t1, t1);
  cu[k] = t2[1];
  cv[k] = t2[2];
end;

 Calculation of Zsi --- (Arccosh(z))

for k = 1 to m do
begin
  cu[k] = cu[k] * cu[k];
  cv[k] = cv[k] * cv[k];
  z1[1] = 1 - cu[k];
  z1[2] = cv[k];
  arccosh(z1, z2);
  zu[k] = z2[1];
  zv[k] = z2[2];
end;

 Calculation of constants in filter polynomial factors

for i = 1 to m do
begin
  zu[i] = zu[i] * zu[i];
  zv[i] = zv[i] * zv[i];
  cexp(zu[i], zv[i]);
  zu[i] = -2.0 * zu[i];
  zv[i] = -2.0 * zv[i];
end;

 Initialize arrays for polynomial expansion

for i = 0 to m do
for j = 0 to m do

217     begin
218     a[I,J]:=0.0;
219     b[I,J]:=0.0;
220     c[I,J]:=0.0;
221     d[I,J]:=0.0;
222     end;
223     a[0,0]:=1.0;
224
225     {*****************************************************************************
226     Loop to generate expanded polynomials
227     *****************************************************************************}
228
229     for k:=1 to m do begin
230         for l:=1 to k do begin
231             for j:=0 to k do begin
232                 I:=I-1;
233                 c[I,J]:=c[I,J]+a[I,J];
234                 d[I,J]:=d[I,J]+b[I,J];
235             end;
236         end;
237         for I:=0 to k do begin
238             for J:=0 to k do begin
239                 c[I,J]:=c[I,J]+a[I,J]*zv[k]+b[I,J]*zu[k];
240                 d[I,J]:=d[I,J]+a[I,J]*zu[k]+b[I,J]*zu[k];
241             end;
242         end;
243     end;
244
245     num:=1.0/a[0,0];
246     for I:=0 to m do begin
247         for J:=0 to m do begin
248             a[I,J]:=a[I,J]*num;
249         end;
250     end;
251     writeln('Do you wish amplitude response?');
252     writeln('scaled to 0 dB at w1T = w2T = 0.0? (Y/N)');
253     readln(temp);
254     temp:=upcase(temp);
255     if temp='Y' then begin
256         sum:=0.0;
257         for I:=0 to m do begin
258             for J:=0 to m do begin
259                 sum:=sum+a[I,J];
260             end;
261         end;
262         num:=sum;
If temp='N' then
begin
writeln(' Do you wish amplitude response?');
write(' scaled to 0 dB at w1T = w2T = pi? (Y/N)');
readln(temp);
temp:=upcase(temp);
if temp='Y' then
begin
  sum:=0.0;
  sign:=-1.0;
  for i:=0 to m do
  begin
    sign:=-sign;
    for j:=0 to m do
    begin
      sum:=sum+sign*a[i,j];
      sign:=-sign;
    end;
    end;
  num:=sum;
end;
end;
writeln;
write('Enter Filename for output of coefficients ');
readln(filename);
assign(output,filename);
rewrite(output);
writeln(output,m);
writeln(output,m);
writeln(output,num:12:11);
for i:=0 to m do
  for j:=0 to m do
  begin
    writeln(i,j,a[i,j]:12:11);
  end;
close(output);
{ writeln;
  writeln(num);
  writeln;
  for i:=0 to m do
  for j:=0 to m do
  writeln(a[i,j]);
}
end.
APPENDIX 5

Butterworth Low-Pass Filter
Response Program Source Code
program bttr2type1;
{

This program calculates the frequency, phase and delay response of a two dimensional digital filter derived from the Butterworth argument **newcosh**

Program By - Tim Kent Date Jun 87

Language - Turbo Pascal 3.0

Filter Theory - Dr. J. Soltis

const
ordmax=20;
nptsx=25;           (*** needed for graph3d.pas)  
nptsy=25;            (**)
npts=625;           (**)

type
form = array [1..3,1..npts] of real;   (***)
form2 = array [1..2,1..2] of real;    (***)
form4 = string[40];
form3 = array [1..2] of real;

var
data,data2,delysform;         (***)
xy :form2;              (***)
emax,eps,fact0,gamma,cl,x,y,s,t,wt1,wt2,mag,p1,p2,scalereal;
phase,deltal,delta2,delp1,delp2,n :real;
l,j,k,m,nextx,nexty :integer;
u,v,cu,cv,zu,zv :array [1..ordmax] of real;
z1,z2,t1,t2,dp,t3,t4 :form3;
flag :boolean;
temp :char;
title :string[40];

Subprograms
**********

procedure cadd(var z1,z2,z3:form3);
{
Procedure to simulate complex addition
**********

begin
z1[1]:=z2[1]+z3[1];
z1[2]:=z2[2]+z3[2];
end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
procedure cexp(var x,y:real);

Procedure to simulate complex exponentiation

begin
  t:=x;
  s:=y;
  x:=exp(t)*cos(s);
  y:=exp(t)*sin(s);
end;

procedure cmult(var z1,z2,z3:form3);

Procedure to simulate complex multiplication

begin
  z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
  z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
end;

procedure arccosh(var z1,z2:form3);

Procedure to calculate arccosh of z

begin
  x:=z1[1];
  y:=z1[2];
  b:=x*x-y*y-1;
  c:=2*x*y;
  d:=x+sqrt((sqrt(b+b+c+c)+b)/2);
  if c>0 then
    sign:=1
  else
    sign:=-1;
  e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
  if (abs(e)<1e-100)and(d<0.0) then
    t:=pi
  else
    begin
      if abs(d)<1e-100 then
        begin
          if e>0 then
sign:=1
else
  sign:=-1;
t:=pi/2*sign;
end
else
  begin
    t:=arctan(e/d);
    if d<0 then
      if e>0 then
        t:=t+pi
      else
        t:=t-pi;
    end;
  end;
g:=t;
end;

begin
  clrscr;
  writeln;
  writeln('Enter order of filter (+ve < ordmax./2); ');
  readln(n);
m:=trunc(2*n);
  writeln('Enter Gamma (0<gamma<1.0); ');
  readln(gamma);
c1:=1-cos(gamma*pi);

for k:=1 to m do
  begin
    x:=1-c1*cos((2*k+1)*pi/n/2);
    y:=-c1*sin((2*k+1)*pi/n/2);
  end;

End of subprograms

Mainline Program

begin
  clrscr;
  writeln;
  writeln('Frequency Response - Type I - Butterworth Argument #2');
  writeln;
  writeln('Enter order of filter (+ve < ordmax./2); ');
  readln(n);
m:=trunc(2*n);
  writeln('Enter Gamma (0<gamma<1.0); ');
  readln(gamma);
c1:=1-cos(gamma*pi);

for k:=1 to m do
  begin
    x:=1-c1*cos((2*k+1)*pi/n/2);
    y:=-c1*sin((2*k+1)*pi/n/2);
  end;
163  z1[1]:=x;
164  z1[2]:=y;
165  arccosh(z1,z2);
166  zu[k]:=z2[1];
167  zv[k]:=z2[2];
168  end;
169
170  (*---------------------------------------------
171  Setting of x-y-axis limits for plotting
172  ***********************************************)
173  xy[1,1]:=0.0;
174  xy[1,2]:=pi;
175  xy[2,1]:=0.0;
176  xy[2,2]:=pi;
177
178  (*---------------------------------------------
179  Selection of Table of Values
180  ***********************************************)
181  flag:=false;
182  write('Do you want a table of values for magnitude and phase (Y/N)?');
183  readln(temp);
184  temp:=upcase(temp);
185  if temp='Y' then
186     begin
187       flag:=true;
188     end;
189  if flag then
190     begin
191       writeln;
192       writeln('Frequency Response - Magnitude and Phase');
193       writeln(' |w1*T w2*T [H(z1,z2)] <H(z1,z2)| (dB) (rad)');
194       writeln;
195     end;
196
197  (*---------------------------------------------
198  Calculation of constants in filter polynomial factors
199  ***********************************************)
200  for i:=1 to m do
201     begin
202       if zu[i]<0.0 then
203           begin
204             zu[i]:=-zu[i];
205             zv[i]:=-zv[i];
206           end;
207           cexp(zu[i],zv[i]);
208           zu[i]:=-2.0*zu[i];
209           zv[i]:=-2.0*zv[i];
210           end;
211       end;
212   (*---------------------------------------------
213   Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Loop to generate filter magnitude response

for i=1 to nptsx do
    begin
        for j=1 to nptsy do
            begin
                wt1:=(i-1)*(xy[1,2]-xy[1,1])/(nptsx-1)*xy[1,1];
                wt2:=(j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)*xy[2,1];
                z1[1]:=cos(-wt1);
                z1[2]:=sin(-wt1);
                z2[1]:=cos(-wt2);
                z2[2]:=sin(-wt2);
                dp[1]:=1.0;
                dp[2]:=0.0;
                cadd(t1,z1,z2);
            end;
        end;
    end;

Loop to multiple filter polynomial factors

for k=1 to m do
    begin
        t2[1]:=zu[k];
        t2[2]:=zv[k];
        cadd(t3,t1,t2);
        cmult(t4,dp,t3);
        dp[1]:=t4[1];
        dp[2]:=t4[2];
    end;

Calculation of magnitude in dB with dc scaling

k:=(l-1)*nptsy+j;
if k=1 then scale:=mag;
mag:=-20*ln(mag/scale)/ln(10);

Calculation of phase from -2pi - 0

phase:=-1.0*arctan(dp[2]/dp[1]);
if dp[1]<0.0 then
    phase:=-pi+phase
else
    begin
        if dp[2]<0.0 then phase:=-2*pi+phase;
    end;
data[1,k]:=wt1;
data[2,k]:=wt2;
data[3,k]:=mag;
data2[1,k]:=wt1;
data2[2,k]:=wt2;
data2[3,k]:=phase;
if flag then
  begin
    writeln(wt1:10:6,'',wt2:10:6,'',mag:14:6,'',phase:14:6);
  end;
end;
end:
end;

{**************************************************************************
Call to graphing procedure
**************************************************************************}
cls cr;
title:=' Magnitude Response (in dB)';
graph3d(data,xy,title);
cls cr;
title:=' Phase Response ';
graph3d(data2,xy,title);
cls cr;

{**************************************************************************
Calculation of group delays of filter
**************************************************************************}
cls cr;

{**************************************************************************
Simulation of derivative by linear difference equation
**************************************************************************}
delta1:=(xy[1,2]-xy[1,1])/nptsx;
delta2:=(xy[2,2]-xy[2,1])/nptsy;
for l:=1 to nptsx do
  begin
    wt1:=(l-1)*(xy[1,2]-xy[1,1])/(nptsx-1)*xy[1,1];
    for j:=1 to nptsy do
      begin
        wt2:=(j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)*xy[2,1];
        k:=(l-1)*nptsy+j;
        ntxy:=k+nptsy;
        nxly:=k+1;
        if nxly>nptsy then nxly:=nxly-2;
        if ntxy>npts then ntx:=ntx-2*nptsy;
        data[1,k]:=wt1;
        dely[1,k]:=wt1;
        data[2,k]:=wt2;
        dely[2,k]:=wt2;
        delp1:=-data2[3,ntx]+data2[3,k];
        delp2:=-data2[3,nxly]+data2[3,k];
      end;
  end;
}
{**************************************************************************
Scaling of vertical change elements for periodic wrap-around
**************************************************************************}
If abs(delp1)>pi*1.2 then
begin
  if delp1>0.0 then
delp1:=delp1-2*pi
else
delp1:=delp1+2*pi;
end;
if abs(delp1)>pi*1.2 then
begin
  if delp1>0.0 then
delp1:=delp1-2*pi
else
delp1:=delp1+2*pi;
end;
data[3,k]:=delp1/delta1;
dely[3,k]:=delp2/delta2;
if j=nptsy then dely[3,k]:=-dely[3,k];
if l=nptsx then data[3,k]:=-data[3,k];
end;
end;

Call to Graphing Subroutines

clrscr;
title:=' Group Delay 1 ';
graph3d(data,xy,title);
clrscr;
title:=' Group Delay 2 ';
graph3d(dely,xy,title);
clrscr;

End of mainline program

end.
program btt2root;
{
 *********************************************************************************************
  **
  ** This program calculates the root trajectory
  ** of a two dimensional digital filter derived
  ** from the second Butterworth argument
  **
  ** Program By - Tim Kent Date Jan.86
  ** Language - Turbo Pascal 3.0
  **
  ** Filter Theory - Dr. J. Soltis
  **
 *********************************************************************************************
}

const
ordmax=20;
 npts=150; (*-Needed for graph2d)
nptsord=3000;
type
  form = array [1..2,1..nptsord] of real; (**)
  form3 = array [1..2] of real;
  form4 = string[40];
var
data : form;
 amax,eps,n, fact0 , gamma, ci,x,y,s,t,p1,p2,theta :real;
j,k,m,l : integer;
u,v, cu,cv,zu,zv : array [1..ordmax] of real;
z1,z2,t1,t2 : form3;
title : string[40];

(*---------------
* Subprograms
*---------------)

(*$1 graph2d.pas)

procedure csub(var z1,z2,z3:form3);
{
  *********************************************************************************************
  Procedure to simulate complex subtraction
  *********************************************************************************************
}
begin
  z1[1]:=z2[1]-z3[1];
  z1[2]:=z2[2]-z3[2];
end;

procedure clnv(var z1,z2:form3);
{
  Procedure to simulate complex inversion
}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
55 //**************************************************************************
56 var
57 mag :real;
58 begin
59 mag := z2[1]*z2[1]+z2[2]*z2[2];
60 z1[1] := z2[1]/mag;
62 end;
63
64 procedure cexp(var z1,z2:form3);
65 {**************************************************************************
66 Procedure to simulate complex exponentiation
67**************************************************************************}
68 begin
69 z1[1] := exp(z2[1])*cos(z2[2]);
70 z1[2] := exp(z2[1])*sin(z2[2]);
71 end;
72
73 procedure cart(var z1:form3; theta:real);
74 {**************************************************************************
75 Procedure to perform cartesian to polar conversion
76**************************************************************************}
77 begin
78 z1[1] := cos(theta);
79 z1[2] := sin(theta);
80 end;
81
82 procedure arccosh(var z1,z2:form3);
83 {**************************************************************************
84 Procedure to calculate arccosh of z
85**************************************************************************}
86 var  
87 x,y,b,c,d,e,f,g,t,sign :real;
88 begin
89 x := z1[1];
90 y := z1[2];
91 b := x*x-y*y-1;
92 c := 2*x*y;
93 d := x+sqrt((sqrt(b+b*c*c)+b)/2);
94 if c>0 then
95 sign := 1
96 else
97 sign := -1;
98 e := y*sign*sqrt((sqrt(b+b*c*c)-b)/2);
99 f := ln(sqrt(d+d*e*e));
100 if ((abs(e)<le-100)and(d<0.0)) then

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
t:=pi
else
begin
  if abs(d)<1e-100 then
    begin
      if e>0 then
        sign:=1
      else
        sign:=-1;
      t:=pi/2*sign;
    end
  else
    begin
      t:=arctan(e/d);
      if d<0 then
        if e>0 then
          t:=t+pi
        else
          t:=t-pi;
    end;
g:=t;
z2[1]:=f;
z2[2]:=g;
end;

{***********************
End of subprograms
***********************

Mainline program
***********************}

begin
  clrscr;

{************************
Input filter specifications
************************

writeln('Root Trajectory - Type I - Butterworth Argument #2');
writeln;
write('Enter order of filter (+ve <',ordmax,'/2) : ');
readln(n);
m:=trunc(2*n);
write('Enter Gamma (0<gamma<1.0) : ');
readln(gamma);
c1:=1-cos(gamma*pi);

{************************
Calculate zsis - arccosh(z)
************************

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
for k:=1 to m do
begin
  x:=c*cos((2*k+1)*pi/n/2);
  y:=c*sin((2*k+1)*pi/n/2);
  z1[k]:=-x;
  z1[k]:=-y;
  arccosh(z1,z2);
  zv[k]:=z2[k];
end;

{**********************************************************
Loop to generate root trajectories for all factors
**********************************************************}
for i:=1 to m do
begin
  if zv[i]<0.0 then
  begin
    zv[i]:=-zv[i];
  end;
  t1[i]:=zv[i];
  t2[i]:=cexp(t2,t1);
  t1[i]:=2*t2[i];
  t2[i]:=2*t2[i];
end;

{**********************************************************
Loop to generate points of trajectories
**********************************************************}
for i:=1 to npts do
begin
  theta:=(i-1)/(npts-1)*2*pi;
  cart(z2,theta);
  csub(t2,t1,z2);
  cinv(z1,t2);
  data[1,i+(i-1)*npts]:=z1[i];
  data[2,i+(i-1)*npts]:=z2[i];
end;

{**********************************************************
Call to graphing routine
**********************************************************}
crscr;
write('title:="Root Trajectories";
graph2d(data,m,title);
217  clrscr;
218
219  {                        
220  End of Mainline Program  
221  }                        
222
223  end.
224
225
program butter;

{* This program calculates the expanded transfer function of a two dimensional digital filter derived from the second butterworth argument *
* Program By - Tim Kent Date Jul.87 *
* Language - Turbo Pascal 3.0 *
* Filter Theory - Dr. J. Soltis , T. Kent *
*
******************************************************************************}

const
ordmax=20;
type
form3 =array [1..2] of real;
var
num,max,sum,eps,n,fact0,gamma,sign,
c1,x,y,s,t,wt1,wt2,mag,p1,p2,scalereal;
l,j,k,m,nextx,nexty :integer;
z1,z2,t1,t2 :form3;
flag :boolean;
tempt :char;
filename :string[13];
output :text;
a,b,c,d :array [0..ordmax,0..ordmax] of real;

******************************************************************************
Subprograms
******************************************************************************
procedure pause;
begin
writeln;
writeln('press any key to continue');
repeat until keypressed;
writeln;
end;

procedure cexp(var x,y:real);
******************************************************************************
Procedure to simulate complex exponentiation
******************************************************************************
begin
{**************
exp(x+yi)=exp(x)*(cos(y)+isin(y))
**************}
writeln;

var s, t : real;
begin
  t := x;
  s := y;
  x := exp(t)*cos(s);
  y := exp(t)*sin(s);
end;

procedure cmult(var z1, z2, z3: form3);
{******************************************
  Procedure to simulate complex multiplication
  ******************************************}
begin
end;

procedure arccosh(var z1, z2: form3);
{******************************************
  Procedure to calculate arccosh of z
  ******************************************}
var x, y, b, c, d, e, f, t, sign: real;
begin
  x := z1[1];
  y := z1[2];
  b := x*x - y*y - 1;
  c := 2*x*y;
  d := x + sqrt((sqrt(b*b) - b*x + c))/2;
  if c > 0 then
    sign := 1
  else
    sign := -1;
  e := y + sqrt((sqrt(b*b) + b*x + c))/2;
  f := ln(sqrt(d + e + sqrt(e*e)));
  if ((abs(e) < 1e-100) and d < 0.0) then
    t := pi
  else
    begin
      if abs(d) < 1e-100 then
        begin
          if e < 0 then
            sign := 1
          else
            sign := -1;
        end;
      t := pi/2*sign;
    end;
```plaintext
else
begin
    t:=arctan(e/d);
    if d<0 then
        if e>0 then
            t:=t+pi
        else
            t:=t-pi;
    end;
end;

g:=t;
if f<0 then
    z2[1]:=f;
else
    z2[2]:=g;
end;

[*********************
End of subprograms
*********************]

Mainline Program

begin
clrscr;

[*********************
Input filter design parameters
*********************]
write('Expanded Transfer function Generator');
write('Butterworth Argument #2');
write;
write('Enter order of filter (+ve <:ordmax./2) : ');
readin(n);
m:=trunc(2*n);
write('Enter Gamma (0<gamma<1.0) : ');
readin(gamma);
c:=1-cos(gamma*pi);

[*********************
Calculation of Zsl — (Arccosh(z))
*********************]
for k:=1 to m do
begin
    x:=1-c1*cos((2*k+1)*pi/n/2);
y:=c1*sin((2*k+1)*pi/n/2);
z1[1]:=x;
z1[2]:=y;
arccosh(z1,z2);
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Calculation of constants in filter polynomial factors

for i:=1 to m do
begin
  if zu[i]<0.0 then
  begin
    zu[i] := -zu[i];
    zv[i] := -zv[i];
  end;
  cexp(zu[i],zv[i]);
  zu[i] := -2.0*zu[i];
  zv[i] := -2.0*zv[i];
end;

for i:=0 to m do
for j:=0 to m do
begin
  a[i,j] := 0.0;
  b[i,j] := 0.0;
  c[i,j] := 0.0;
  d[i,j] := 0.0;
end;
a[0,0] := 1.0;

for k:=1 to m do
begin
  for i:=1 to k do
  for j:=0 to k do
  begin
    l := l-1;
    c[i,j] := c[i,j] + a[i,j];
    d[i,j] := d[i,j] + b[i,j];
    c[j,i] := c[j,i] + a[j,i];
    d[j,i] := d[j,i] + b[j,i];
  end;
end;
for i:=0 to k do
for j:=0 to k do
begin
  c[i,j] := c[i,j] + zu[k]*a[i,j]*zu[k] - b[i,j]*zv[k];
  d[i,j] := d[i,j] + zu[k]*b[i,j]*zu[k];
for I = 0 to k do
for J = 0 to k do
begin
o[I, J] := c[I, J];
b[I, J] := d[I, J];
c[I, J] := 0.0;
d[I, J] := 0.0;
end;
end;
um := 1.0 / a[0, 0];
for I = 0 to m do
for J = 0 to m do
a[I, J] := a[I, J] * num;
write(' Do you wish amplitude response?');
write(' scaled to 0 dB at w1T = w2T = 0.0? (Y/N)');
readin(temp);
temp := upcase(temp);
if temp = 'Y' then
begin
sum := 0.0;
for I = 0 to m do
for J = 0 to m do
sum := sum + a[I, J];
num := sum;
end;
if temp = 'N' then
begin
write(' Do you wish amplitude response?');
write(' scaled to 0 dB at w1T = w2T = pi? (Y/N)');
readin(temp);
temp := upcase(temp);
if temp = 'Y' then
begin
sum := 0.0;
sign := 1.0;
for I = 0 to m do
begin

sign := -sign;
for J = 0 to m do
begin

sum := sum + sign * a[I, J];
sign := -sign;
end;
end;
end;
num := sum;
end;
write(' Enter Filename for output of coefficients ');
readin(filename);
assign(output, filename);
rewrite(output);
writeln(output, m);
write(output,m);
write(output,num:12:11);
for i:=0 to m do
  for j:=0 to m do
    begin
      write(i,j,a[i,j]:12:11);
      write(output,a[i,j]:12:11);
      end;
    close(output);

    begin
      write;
      write(num);
      write;
      for i:=0 to m do
        for j:=0 to m do
          write(a[i,j]);
    end.
  end.
APPENDIX 6

Chebyshev High-Pass Filter
Response Program Source Code
program cheb_hp3;
{
   #########################################################################
   #
   # This program calculates the frequency, phase and delay
   # response of a two dimensional digital filter derived
   # from the third chebyshev high-pass argument
   #
   # Program By - Tim Kent Date Jul 87
   # Language - Turbo Pascal 3.0
   #
   # Filter Theory - Dr. J. Soltis, T. Kent
   #
   #########################################################################
   constant
   ordmax=20;
nptsx=25; (* needed for graph3d.pas *)
nptsy=25; (* )
npts=625; (* )
type
form =array [1..3,1..npts] of real; (* )
form2 =array [1..2,1..2] of real; (* )
form4 =string[40]; (* )
form3 =array [1..2] of real; (* )
var
data, data, data.delay :form; (* )
xy :form2; (* )
amax, eps, fact0, gamma, cl, x, y, s, t, wt1, wt2, mag, pl, p2, scale :real;
phase, delta1, delta2, delpl, delp2, n :real;
l, j, k, m, nextx, nexty :integer;
u, v, cu, cv, zu, zv :array [1..ordmax] of real;
z1, z2, t1, t2, dp, t3, t4 :form3;
flag :boolean;
temp :char;
title :form4;

{ ************
Subprograms
{ ************}
(* $1 graph3d.pas *)
procedure cadd(var z1, z2, z3: form3);
(* Procedure to simulate complex addition
{ ************ *)
begin
z1[1] := z2[1] + z3[1];
end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
55 56 procedure cexp(var x,y:real);
57 58 {******************************************************************************
59 Procedure to simulate complex exponentiation
60 ******************************************************************************}
61 62 var
63  s,t :real;
64 begin
65  t:=x;
66  s:=y;
67  x:=exp(t)*cos(s);
68  y:=exp(t)*sin(s);
69 end;
70 71 procedure cmult(var z1,z2,z3:form3);
72 73 {******************************************************************************
74 Procedure to simulate complex multiplication
75 ******************************************************************************}
76 77 begin
78  z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
79  z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
80 end;
81 82 procedure arccosh(var z1,z2:form3);
83 84 {******************************************************************************
85 Procedure to calculate arccosh of z
86 ******************************************************************************}
87 88 var
89  x,y,b,c,d,e,f,g,t,sign :real;
90 91 begin
92  x:=z1[1];
93  y:=z1[2];
94  b:=x*y-y*x-1;
95  c:=2*x*y;
96  d:=x+sqrt((sqrt(b^2+b*c+c^2)+b)/2);
97  if c>0 then
98    sign:=1
99  else
100    sign:=-1;
101  e:=y*sign*sqrt((sqrt(b^2+b*c+c^2)-b)/2);
102  f:=ln(sqrt(d^2*e*e));
103  if ((abs(e)<1e-100)and(d<0.0)) then
104    t:=pi
105  else
106    begin
107      if abs(d)<1e-100 then
108        begin
109          end
if e > 0 then
    sign = 1
else
    sign = -1;
t = pi/2*sign;
end
else
    begin
        t = arctan(e/d);
        if d < 0 then
            if e > 0 then
                t = t + pi
            else
                t = t - pi;
        end;
    end;
g = t;
z[2][1] = f;
z[2][2] = g;
end;

End of subprograms

Mainline Program

begin
    clrscr;

    writeln('Frequency Response - High Pass Chebyshev Argument #3');
    writeln;
    writeln('Enter order of filter (+ve < order_max, '/2') ; ');
    readln(n);
    m := trunc(2^n);
    writeln('Enter passband ripple Amax ; ');
    readln(amax);
    writeln('Enter Gamma (0<gamma<1.0) ; ');
    readln(gamma);
    cl := 1.0*cos(gamma*pi/2)*cos(gamma*pi/2);
    writeln;
    eps := sqrt(exp(0.1*amax*ln(10))-1);

Calculation of chebyshev poles

for k:=1 to m do 
begin 
  \( u[k] := (2k-1) \frac{\pi}{n} \);  
  \( v[k] := \text{fact0} \);  
  writeln(lst,'u,v',u[k],v[k]); 
end;

Calculation of Cosine of Chebyshev poles

for k:=1 to m do 
begin 
  \( c_u[k] := 0.5 \sin(u[k]) \times (\exp(v[k]) + \exp(-v[k])); \)  
  \( c_v[k] := 0.5 \cos(u[k]) \times (\exp(v[k]) - \exp(-v[k])); \)  
  \( t1[1] := c_u[k] \);  
  \( t1[2] := c_v[k] \);  
  cmult(t2[1],t1[1],t1[2]); 
  \( c_u[k] := t2[1] \);  
  \( c_v[k] := t2[2] \);  
  writeln(lst,'sinsquared',c_u[k],c_v[k]); 
end;

Calculation of Zsl -- (Arccosh(z))

for k:=1 to m do 
begin 
  \( c_u[k] := c1 \times c_u[k] \);  
  \( c_v[k] := c1 \times c_v[k] \);  
  \( zl[1] := c_u[k] - 1.0 \);  
  \( zl[2] := c_v[k] \);  
  \( \text{arccosh}(zl[1],zl[2]); \)  
  \( z_u[k] := zl[1] \);  
  \( z_v[k] := zl[2] \);  
  writeln(lst,'precosh',zl[1],zl[2],cosh,zl[1],zl[2]); 
end;

Setting of x-y-axes limits for plotting

XY[1,1] := pi;  
XY[1,2] := pi;  
XY[2,1] := pi;  
XY[2,2] := pi;

Selection of Table of Values
flag:=false;
write('Do you want a table of values for magnitude and phase (Y/N)?');
readln(temp);
temp:=Upcase(temp);
if temp='Y' then
begin
flag:=true;
end;
if flag then
begin
 writeln('Frequency Response - Magnitude and Phase');
 writeln(' \omega \rightarrow T \quad w_2 \rightarrow T \quad |H(z_1,z_2)| \quad \angle(H(z_1,z_2))');
 writeln(' (dB) \quad (\text{rad})');
 end;

{*******************************************************************************
Calculation of constants in filter polynomial factors
*******************************************************************************}
for i:=1 to m do
begin
  if (zu[i]<0.0) then zu[i]:=-zu[i];
cexp(zu[i],zv[i]);
zu[i]:=-2.0*zu[i];
zv[i]:=-2.0*zv[i];
  ( writeln('fact ',zu[i],zv[i]));
end;

{*******************************************************************************
Loop to generate filter magnitude response
*******************************************************************************}
for i:=nptsx downto 1 do
begin
  for j:=nptsy downto 1 do
begin
    wt1:=(i-1)*(xy[1,2]-xy[1,1])/(nptsx-1)+xy[1,1];
    wt2:=(j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)+xy[2,1];
    z1[i]:=cos(-wt1);
    z1[i]:=sin(-wt1);
    z2[i]:=cos(-wt2);
    z2[i]:=sin(-wt2);
    dp[1]:=1.0;
    dp[2]:=0.0;
cadd(tl,z1,z2);
end;
end;

{*******************************************************************************
Loop to multiple filter polynomial factors
*******************************************************************************}
for k:=1 to m do
begin

Calculation of magnitude in dB with dc scaling

\[
\begin{align*}
\text{k} & := (l-1) \times \text{nptsy} + j; \\
\text{mag} & := \sqrt{\text{dp}[1] \times \text{dp}[1] + \text{dp}[2] \times \text{dp}[2]}; \\
\text{if} & \text{ k} = (\text{nptsx} - 1) \times \text{nptsy} + \text{nptsy} \text{ then } \text{scale} := \text{mag}; \\
& \text{mag} := -20 \times \ln(\text{mag}/\text{scale}) / \ln(10); \\
\end{align*}
\]

Calculation of phase from $-2\pi - 0$

\[
\begin{align*}
\text{phase} & := -1.0 \times \arctan(\text{dp}[2]/\text{dp}[1]); \\
\text{if} & \text{ dp}[1] < 0.0 \text{ then } \\
& \text{phase} := -\pi + \text{phase}; \\
\text{else} & \begin{align*}
& \text{if} \text{ dp}[2] < 0.0 \text{ then } \text{phase} := -2\pi + \text{phase}; \\
& \text{end}; \\
\end{align*} \\
\text{data}[1,k] & := \text{wt}; \\
\text{data}[2,k] & := \text{wt}; \\
\text{data}[3,k] & := \text{mag}; \\
\text{data}[21,k] & := \text{wt}; \\
\text{data}[22,k] & := \text{wt}; \\
\text{data}[23,k] & := \text{phase}; \\
& \text{if flag then } \\
& \begin{align*}
& \text{writeln}(\text{wt1}:10:6,' ',\text{wt2}:10:6,' ',\text{mag}:14:6,' ',\text{phase}:14:6); \\
& \text{end}; \\
& \text{end}; \\
\end{align*} \\
\text{end}; \\
\text{end}; \\
\text{end}; \\
\end{align*}
\]

Call to graphing procedure

\[
\begin{align*}
& \text{clrscr}; \\
& \text{title} := "\text{Magnitude Response (in dB)}"; \\
& \text{graph3d(data,xy,title)}; \\
& \text{clrscr}; \\
& \text{title} := "\text{Phase Response}"; \\
& \text{graph3d(data2,xy,title)}; \\
& \text{clrscr}; \\
\end{align*}
\]
Calculation of group delays of filter

Simulation of derivative by linear difference equation

delta1:=(xy[1,2]-xy[1,1])/nptsx;
delta2:=(xy[2,2]-xy[2,1])/nptsy;
for i=1 to nptsx do
begin
    wt1:=(i-1)*(xy[1,2]-xy[i,1])/(nptsx-1)*xy[i,1];
    for j=1 to nptsy do
    begin
        wt2:=(j-1)*(xy[2,2]-xy[2,j])/(nptsy-1)*xy[2,j];
        k:=(i-1)*nptsy+j;
        nextx:=k+nptsy;
        nexty:=k+1;
        if nexty>nptsy then nexty:=nexty-2;
        if nextx>npts then nextx:=nextx-2*nptsy;
        data[1,k]:=wt1;
        dely[1,k]:=wt1;
        data[2,k]:=wt2;
        dely[2,k]:=wt2;
        delpl:=-data2[3,nextx]+data2[3,k];
        delp2:=-data2[3,nexty]+data2[3,k];
    end;
end;
end;

Scaling of vertical change elements for periodic wrap-around

if abs(delpl)>pi*1.50 then
begin
    if delpl>0.0 then
    delpl:=delpl-2*pi
    else
    delpl:=delpl+2*pi;
end;
if abs(delp2)>pi*1.50 then
begin
    if delp2>0.0 then
    delp2:=delp2-2*pi
    else
    delp2:=delp2+2*pi;
end;
data[3,k]:=delpl/deltal;
dely[3,k]:=delp2/delta2;
if j=npts then dely[3,k]:=-dely[3,k];
if l=npts then data[3,k]:=-data[3,k];
if data[3,k]<-10.0 then data[3,k]:=data[3,k]+2*pi/deltal;
if dely[3,k]<-10.0 then dely[3,k]:=dely[3,k]+2*pi/delta2;
end;
Call to Graphing Subroutines

clrscr;
title: 'Group Delay 1';
graph3d(data.xy.title);
clrscr;
title: 'Group Delay 2';
graph3d(dely.xy.title);
clrscr;

End of mainline program

end.
program cheb_hp3root;

{
   ############################################################################
   #
   # This program calculates the root trajectory
   # of a two dimensional digital filter derived
   # from the third chebyshev high-pass argument
   #
   # Program By - Tim Kent       Date Jul 87
   # Language - Turbo Pascal 3.0
   #
   # Filter Theory - Dr. J. Soltis , T. Kent
   #
   ############################################################################
   
const
ordmax=30;
npts=100;                     (**-Needed for graph2d)
nptsord=3000;

type
form  =array [1..10,npts ord] of real;  (**)
form3 =array [1..2] of real;
form4 =string[40];

var
data :form;          (**)
amax,eps,acto,gamma,c1,x,y,s,t,p1,p2,theta,n :real;
l,j,k,m,1 :integer;
u,v,uc,vz,uz,vz :array [1..ordmax] of real;
z1,z2,t1,t2 :form3;
title :form4;

{************
Subprograms
************}

{$1 graph2d.pas}
procedure csub(var z1,z2,z3:form3);
{MNttWNNKMKMXItNMttMNNNNMNNXNNHHMKNtlNMMMItMMMNttWft}
begin
   z1[1]:=z2[1]-z3[1];
   z1[2]:=z2[2]-z3[2];
end;

procedure cmult(var z1,z2,z3:form3);
{MNttWNNKMKMXItNMttMNNNNMNNXNNHHMKNtlNMMMItMMMNttWft}
begin
   z1[1]:=z2[1]*z3[1];
   z1[2]:=z2[2]*z3[2];
end;

procedure graph2d;
{MNttWNNKMKMXItNMttMNNNNMNNXNNHHMKNtlNMMMItMMMNttWft}
begin
   graph2d.pas
end;

procedure graph2d;
{MNttWNNKMKMXItNMttMNNNNMNNXNNHHMKNtlNMMMItMMMNttWft}
begin

end;

procedure clnv(var z1, z2: form3);

{******************************************************}
{Procedure to simulate complex inversion}
{******************************************************}

var
mag : real;

begin
mag := z2[1]*z2[1] + z2[2]*z2[2];
z1[1] := z2[1]/mag;
end;

procedure cexp(var z1, z2: form3);

{******************************************************}
{Procedure to simulate complex exponentiation}
{******************************************************}

begin
z1[1] := exp(z2[1])*cos(z2[2]);
z1[2] := exp(z2[1])*sin(z2[2]);
end;

procedure cart(var z1: form3; theta: real);

{******************************************************}
{Procedure to perform cartesian to polar conversion}
{******************************************************}

begin
z1[1] := cos(theta);
z1[2] := sin(theta);
end;

procedure arccosh(var z1, z2: form3);

{******************************************************}
{Procedure to calculate arccosh of z}
{******************************************************}

var
x, y, b, c, d, e, f, g, t, sign : real;

begin
  x := z1[1];
y := z[2];
b := x*x - y*y - 1;
c := 2*x*y;
d := x*x*(sqrt(b*b + c*c) + b)/2;
if c > 0 then
  sign := 1
else
  sign := -1;
e := y + sign*sqrt((sqrt(b*b + c*c) - b)/2);
f := ln(sqrt(d#d + e*e));
if ((abs(e) <= 100) and (d < 0.0)) then
  t := pi
else
  begin
    If abs(d) <= 100 then
      begin
        if e > 0 then
          sign := 1
        else
          sign := -1;
        t := pi/2*sign;
      end
    else
      begin
        t := atan(e/d);
        if d < 0 then
          if e > 0 then
            t := t + pi
          else
            t := t - pi;
        end;
  end;
g := t;
f := z[1];
g := f;
g := z[2];
end;

[****************************
End of subprograms
**************************]

Mainline program

begin
clrscr;

[****************************
Input filter specifications
****************************]
writeln('Root Trajectory - High Pass Chebyshev Argument #3');
writeln:
write('Enter order of filter (+ve , ordmax , ordmax/2) ; ');
readln(n);
m:=tronc(2*n);
write('Enter passband ripple Amax ; ');
readln(amax);
write('Enter Gamma (0<gamma<1.0) ; ');
readln(gamma);
c1:=1.0*cos(gamma*1.0/2)*cos(gamma*1.0/2);
write;
eps:=sqrt(exp(0.1*amax*ln(10))-1);
******
Calculate chebyshev poles
******
factO:=ln(1/eps+sqrt(1/eps/eps+1))/n/2;
for k:=1 to m do
begin
u[k]:=(2*k-1)*pl/4/n;
vl[k]:=factO;
{ writeln(lst.'u,v ',u[k],v[k]); }
end;
******
Calculate cosine of cheb poles
******
for k:=1 to m do
begin
cuk:=0.5*sin(u[k])*(exp(v[k])+exp(-v[k]));
cvk:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
t1[1]:=cuk[k];
t1[2]:=cvk[k];
cmult(t2,t1,t1);
cuk[k]:=t2[1];
cvk[k]:=t2[2];
{ writeln(lst.'sinesquared ',cuk[k],cvk[k]); }
end;
******
Calculate zsis - arcosh(z)
******
for k:=1 to m do
begin
cuk[k]:=c1*cuk[k];
cvk[k]:=c1*cvk[k];
z1[1]:=cuk[k]-1.0;
z1[2]:=cvk[k];
arccosh(z1[2]);
zu[k]:=z1[1];
zv[k]:=z2[2];
{ writeln(lst,'precosh ',z1[1],z1[2],',cosh',z2[1],z2[2]); }
end;

{*******************************************************************************
Loop to generate root trajectories for all factors
*******************************************************************************}
for l:=1 to m do
begin
if (zu[l]<0.0) then zu[l]:=-zu[l];
t1[l]:=zu[l];
t2[l]:=zv[l];
exp(t2,t1);
t1[l]:=2*t2[l];
t2[l]:=2*t2[l];
{ writeln(lst,' fact ',t1[l],t1[l]); }
end;

{*******************************************************************************
Loop to generate points of trajectories
*******************************************************************************}
for l:=1 to npts do
begin
theta:=(l-1)/(npts-1)*2*pi;
cart(z2,theta);
csub(t2,t1,z2);
cinv(z1,t2);
data[1,l+(l-1)*npts]:=z1[l];
data[2,l+(l-1)*npts]:=z2[l];
end;
end;

{*******************************************************************************
Call to graphing routine
*******************************************************************************}
c1rscr;
title:=' Root Trajectories';
graph2d(data,m,title);
c1rscr;

{*******************************************************************************
End of Mainline Program
*******************************************************************************}
end.
program cheb_hp3;

{                                                                                                                   
  ***************************************************************************************************************** 
  * This program calculates the expanded transfer function of a two dimensional digital filter derived from the third chebyshev high-pass argument * 
  * Program By - Tim Kent Date Jul.87 * 
  * Language - Turbo Pascal 3.0 * 
  * Filter Theory - Dr. J. Soltis, T. Kent * 
  *****************************************************************************************************************} 

const
ordmax=20;
type
form3 =array [1..2] of real;
var
num,sm,eps,n,fact0,gamma,sign,
c1,x,y,s,t,wt1,wt2,mag,p1,p2,scale :real;
l,j,k,m,nextx,nexty:integer;
u,v,cu,cv,zu,zv :array [1..ordmax] of real;
z1,z2,t1,t2 :form3;
flag :boolean;
temp :char;
filename :string[13];
output :text;
a,b,c,d :array [0..ordmax,0..ordmax] of real;

{******************************************************
Subprograms
******************************************************}

procedure pause;

{******************************************************
Procedure to temporarily halt program execution
******************************************************}
begin
  writeln;
  writeln('press any key to continue');
  repeat until keypressed;
  writeln;
end;

procedure cexp(var x,y:real);

{******************************************************
Procedure to simulate complex exponetiation
******************************************************}
55 var
56 s,t :real;
57 begin
58 t:=x;
59 s:=y;
60 x:=exp(t)*cos(s);
61 y:=exp(t)*sin(s);
62 end;
63 procedure cmult(var z1,z2,z3:form3);
64 {***************************************************************
65 Procedure to simulate complex multiplication
66 ***************************************************************}
67 begin
68 z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
69 z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
70 end;
71 procedure arccosh(var z1,z2:form3);
72 {***************************************************************
73 Procedure to calculate arccosh of z
74 ***************************************************************}
75 var
76 x,y,b,c,d,e,f,g,t,sign :real;
77 begin
78 x:=z1[1];
79 y:=z1[2];
80 b:=2*x*y;
81 c:=2*x*y;
82 d:=x+sqrt((sqrt((sqrt(b*b+c*c)+b)/2));
83 if c>=0 then
84 sign:=1
85 else
86 sign:=-1;
87 e:=y+sign*sqrt((sqrt((sqrt(b*b+c*c)-b)/2));
88 f:=ln(sqrt(d*d+e*e));
89 if ((abs(e)<le-100)and(d<0.0)) then
90 t:=pl
91 else
92 begin
93 if abs(d)<le-100 then
94 begin
95 sign:=-1;
96 e:=y-sign*sqrt((sqrt((sqrt((sqrt(b*b+c*c)-b)/2));
97 f:=ln(sqrt(d*d+e*e));
98 if ((abs(e)<le-100)and(d<0.0)) then
99 t:=pl/2*sign
100 if (abs(y)<le-100) then
101 begin
102 if (y>0) then
103 sign:=1
104 else
105 sign:=-1;
106 t:=pl/2*sign
107
else
begin
  t:=arctan(e/d);
  if d<0 then
    if e>0 then
      t:=t+pi
    else
      t:=t-pi;
  end;
end;

:g=t;

z[1]:=f;
z[2]:=g;

end;

{**********************************************************************
End of subprograms
**********************************************************************}

Mainline Program
{************************************************************************

begin
clrscr;

{************************************************************************
Input filter design parameters
************************************************************************}
write('Expanded Transfer Function Generator');
write(' High Pass Chebyshev Argument #3');
write(' Enter order of filter (+ve <',ordmax,')/2 > '); readln(n);
m:= trunc(2*n):
write(' Enter passband ripple Amax: '); readln(amax);
write(' Enter Gamma (0<gamma<1.0): '); readln(gamma);
cl:=1.0+cos(gamma*pi/2)*cos(gamma*pi/2):
write(' Enter Gamma (0<gamma<1.0): '); readln(gamma);
eps:=sqrt(exp(0.1*amax*ln(10))-1);

{************************************************************************
Calculation of chebyshev poles
************************************************************************}

for k:=1 to m do
begin
  u[k]:(2*k-1)*pi/4/n;
end;
Calculation of Cosine of Chebyshev poles

for k:=1 to m do
begin
  cu[k]:=0.5*sin(u[k])*(exp(v[k])+exp(-v[k]));
  cv[k]:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
  t[l][1]:=cu[k];
  t[l][2]:=cv[k];
  cmult(t2,t1,t1);
  cu[k]:=t2[1];
  cv[k]:=t2[2];
  writeln('sinsquared ',cu[k],cv[k]);
end;

Calculation of Zsl — (Arccosh(z))

for k:=1 to m do
begin
  cu[k]:=cl*cu[k];
  cv[k]:=cl*cv[k];
  z[l][1]:=cu[k]-1.0;
  z[l][2]:=cv[k];
  arccosh(z1,z2);
  zu[k]:=z2[1];
  zv[k]:=z2[2];
  writeln('precosh ',z[l][1],z[l][2],'cosh ',z2[1],z2[2]);
end;

Calculation of constants in filter polynomial factors

for l:=1 to m do
begin
  if (zu[l]<0.0) then zu[l]:=zu[l];
  zu[l]:=zu[l];
  zv[l]:=zv[l];
  cexp(zu[l],zv[l]);
  zu[l]:=-2.0*zu[l];
  zv[l]:=-2.0*zv[l];
end;

Initialize arrays for polynomial expansion
for \( i = 0 \) to \( m \) do
  for \( j = 0 \) to \( m \) do
    begin
      \( a[i,j] := 0.0 \);
      \( b[i,j] := 0.0 \);
      \( c[i,j] := 0.0 \);
      \( d[i,j] := 0.0 \);
    end;
    \( a[0,0] := 1.0 \);

{ Loop to generate expanded polynomial }

for \( k = 1 \) to \( m \) do
  begin
    for \( i = 1 \) to \( k \) do
      for \( j = 0 \) to \( k \) do
        begin
          \( a[i,j] := a[i-1,j]+b[i,j]; \)
          \( d[i,j] := d[i-1,j]+b[i,j]; \)
          \( c[i,j] := c[i,j]+a[i,j]; \)
          \( d[i,j] := d[i,j]+b[i,j]; \)
        end;
    for \( i = 0 \) to \( k \) do
      for \( j = 0 \) to \( k \) do
        begin
          \( a[i,j] := c[i,j]*num; \)
          \( b[i,j] := d[i,j]; \)
          \( c[i,j] := 0.0; \)
          \( d[i,j] := 0.0; \)
        end;
    \( num := 1.0/a[0,0]; \)
  end;

\( num := 1.0/a[0,0]; \)
for \( i = 0 \) to \( m \) do
  for \( j = 0 \) to \( m \) do
    begin
      \( a[i,j] := a[i,j]*num; \)
    end;

writeln('Do you wish amplitude response?');
write(' scaled to 0 dB at \( w_1T = w_2T = 0.0 \)? (Y/N)');
readln(temp);
temp := upcase(temp);
if temp = 'Y' then
  begin
    \( sum := 0.0; \)
    for \( i = 0 \) to \( m \) do
      begin
        \( a[i,j] := a[i,j]*num; \)
      end;
for j:=0 to m do 
  sum:=sum+a[l,j];
num:=sum;
end;
if temp='N' then begin
  writeln('Do you wish amplitude response?');
  writeln('scaled to 0 dB at w1T = w2T = pi? (Y/N)');
  readin(temp);
  temp:=upcase(temp);
  if temp='Y' then begin
    sum:=0.0;
    sign:=-1.0;
    for l:=0 to m do begin
      sign:=-sign;
      for j:=0 to m do begin
        sum:=sum+sign*a[l,j];
        sign:=-sign;
      end;
    end;
    num:=sum;
  end;
end;
writeln;
write('Enter filename for output of coefficients ');
readin(filename);
assign(output,filename);
rewrite(output);
writeln(output,m);
writeln(output,m);
writeln(output,num:12:11);
for l:=0 to m do 
  for j:=0 to m do
  begin
    { writeln(l,j,a[l,j]:12:11); }
    writeln(output,a[l,j]:12:11);
  end;
close(output);

{ writeln;
  writeln(num);
  writeln;
  for l:=0 to m do
    for j:=0 to m do
      writeln(a[l,j]);
}
APPENDIX 7

General Chebyshev Filter
Response Program Source Code
program chebgen;
[

This program calculates the frequency, phase and delay
response of a two dimensional digital filter derived
from the fifth chebyshev argument (general)

Program By - Tim Kent Date July 87
Language - Turbo Pascal 3.0

Filter Theory - Dr. J. Soltis, T. Kent

const
ordmax = 20;
nptsx = 25; (** needed for graph3d.pas
nptsy = 25;
npts = 625;

type
form = array [1..3,1..npts] of real; (**
form4 = string[40];
form2 = array [1..2,1..2] of real; (**
form3 = array [1..2] of real;

var
data, data2, delay : form; (**
xy : form2; (**
amax, eps, fact0, gamma1, gamma2, c1, x, y, s, t, wt1, wt2, mag, pl, p2, scale : real;
phase, delta1, delta2, delta1, delta2, delta3, delta4 : real;
1, j, k, m, nextx, nexty : integer;
u, v, cu, cv, zu, zv, su, sv : array [1..ordmax] of real;
z1, z2, t1, t2, dp, t3, t4 : form3;
flag : boolean;
temp : char;
title : form4;

Subprograms

procedure cadd(var z1, z2, z3 : form3);

procedure to simulate complex addition

begin
z[1]: = z2[1]*z3[1];
z[2]: = z2[2]*z3[2];
end;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
procedure cexp(var x,y:real);

{ ***************
Procedure to simulate complex exponentiation
 ***************}

var
s,t :real;

begin
  t:=x;
s:=y;
x:=exp(t)*cos(s);
y:=exp(t)*sin(s);
end;

procedure cmult(var z1,z2,z3:form3);

{ ***************
Procedure to simulate complex multiplication
 ***************}

begin
  z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
end;

procedure arccosh(var z1,z2:form3);

{ ***************
Procedure to calculate arccosh of z
 ***************}

var
x,y,b,c,d,e,f,t,signtreal;

begin
  x:=z1[1];
y:=z1[2];
b:=x*x-y*y-1;
c:=2*x*y;
d:=x+sqrt((sqrt(b*b+c*c)-b)/2);
  if c>0 then
    sign:=1
  else
    sign:=-1;
e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
f:=ln(sqrt(d*d+e*e));
  if ((abs(e)<1e-100)and(d<0.0)) then
    t:=pi
  else
    begin
      if abs(d)<1e-100 then
        begin

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
If $e > 0$ then
  \[ \text{sign} = 1 \]
else
  \[ \text{sign} = -1 \]
  \[ t = \pi/2 \text{sign} \]
end
else
begin
  \[ t = \arctan(e/d) \]
  if $d < 0$ then
    if $e > 0$ then
      \[ t = t + \pi \]
    else
      \[ t = t - \pi \]
    end;
  end;
\[ g = t \]
\[ z_2[1] = f \]
\[ z_2[2] = g \]
end;

End of subprograms

***************
End of subprograms

***************
Mainline Program

***************

begin
clrscr;

***************
Input filter design parameters

***************

\[ \text{write}(\text{'Frequency Response - General Chebyshev Argument'}) \]
\[ \text{write}(); \]
\[ \text{write}(\text{'Enter order of filter (+ve <', ordmax, ',/2) : '}) \]
\[ \text{readln}(n) \];
\[ m = \text{trunc}(2 \times n) \];
\[ \text{write}(\text{'Enter passband ripple Amax ; '}) \]
\[ \text{readln}(amax) \];
\[ \text{readln}(gamma1) \];
\[ \text{readln}(gamma2) \];
\[ \text{write}(\text{'Enter Gamma1 (0<gamma1<1.0) ; '}) \]
\[ \text{readln}(gamma1) \];
\[ \text{write}(\text{'Enter Gamma2 (0<gamma2<1.0) ; '}) \]
\[ \text{readln}(gamma2) \];
\[ \text{gamma1} = \text{cos}(gamma1 \times \pi) \];
\[ \text{gamma2} = \text{cos}(gamma2 \times \pi) \];
\[ \text{write}(); \]
\[ \text{eps} = \text{sqrt}((\text{exp}(0.1 \times \text{amax} \times \text{ln}(10))) - 1) \];
Calculation of Chebyshev poles

```
163 fact0:=-ln(1/eps+sqrt(1/eps/eps+1))/n/2;
164 for k:=1 to m do
165 begin
166   u[k]:=(2*k-1)*p1/4/n;
167   v[k]:=fact0;
168 end;
```

Calculation of Cosine of Chebyshev poles

```
170 for k:=1 to m do
171 begin
172   cu[k]:=0.5*cos(u[k])*(exp(v[k])+exp(-v[k]));
173   cv[k]:=-0.5*sin(u[k])*(exp(v[k])-exp(-v[k]));
174   su[k]:=0.5*sqrt(u[k])*(exp(v[k])+exp(-v[k]));
175   sv[k]:=-0.5*sqrt(u[k])*(exp(v[k])-exp(-v[k]));
176   t1[1]:=cu[k];
177   t1[2]:=cv[k];
178   cmult(t2,t1,t1);
179   cu[k]:=t1[1];
180   cv[k]:=t1[2];
181   t1[1]:=su[k];
182   t1[2]:=sv[k];
183   cmult(t2,t1,t1);
184   su[k]:=t1[1];
185   sv[k]:=t1[2];
186 end;
```

Calculation of \( Z_{sl} = \text{Arccosh}(z) \)

```
195 for k:=1 to m do
196 begin
197   cu[k]:=gamma1*cu[k]+gamma2*su[k];
198   cv[k]:=gamma1*cv[k]+gamma2*sv[k];
199   z1[1]:=cu[k];
200   z1[2]:=cv[k];
201   arccosh(z1,z2);
202   zu[k]:=z2[1];
203   zv[k]:=z2[2];
204 end;
```

Setting of x-y-axis limits for plotting

```
215 xy[1,1]:=0.0;
216 xy[1,2]:=p1;
```
xy[2,1]:=0.0;
xy[2,2]:=p1;
if gamma1<1.0 then
begin
xy[1,1]:=-p1;
xy[2,1]:=-p1;
end;

Selection of Table of Values
*******************************************************************************
flag:=false;
write('Do you want a table of values for magnitude and phase (Y/N)?');
readln(temp);
temp:=Upcase(temp);
if temp='Y' then
begin
flag:=true;
end;
if flag then
begin
write ln;
write ln('Frequency Response - Magnitude and Phase');
write ln('w1*T w2*T [H(z1,z2)] <H(z1,z2)');
write ln('dB rad');
write ln;
end;

*******************************************************************************
Calculation of constants in filter polynomial factors
*******************************************************************************
for l:=1 to m do
begin
if (zu[l]<0.0) then
begin
zu[l]:=-zu[l];
zw[l]:=-zw[l];
end;
exp(zu[l],zw[l]);
zu[l]:=-2.0*zu[l];
zw[l]:=-2.0*zw[l];
end;

Loop to generate filter magnitude response
*******************************************************************************
for l:=1 to nptsx do
begin
for j:=1 to nptsy do
begin
wt1:=(1-1)*(xy[1,2]-xy[1,1])/(nptsx-1)*xy[1,1];
end;
wt2 := (j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)*xy[2,1];
zt[1] := cos(-wt1);
z[2] := sin(-wt1);
z2[1] := cos(-wt2);
z2[2] := sin(-wt2);
dp[1] := 1.0;
dp[2] := 0.0;
cadd(t1, z1, z2);

/* Loop to multiple filter polynomial factors*/

for k := 1 to m do
begin
  t2[1] := zv[k];  
t2[2] := zv[k];  
  cadd(t3, t1, t2);  
  cmult(t4, dp, t3);  
  dp[1] := t4[1];  
  dp[2] := t4[2];
end;

/* Calculation of magnitude in dB with dc scaling*/

k := (i-1)*nptsy+j;
if k = 1 then scale := mag;
  mag := -20*ln(mag/scale)/ln(10);

/* Calculation of phase from -2pi - 0*/

phase := -1.0*arctan(dp[2]/dp[1]);
if dp[1] < 0.0 then phase := -pi + phase
else begin
  if dp[2] < 0.0 then phase := -2*pi + phase
  end;
data[1, k] := wt1;
data[2, k] := wt2;
data[3, k] := mag;
data2[1, k] := wt1;
data2[2, k] := wt2;
data2[3, k] := phase;
if flag then
begin
  writeln(wt1:10:6, ' wt2:10:6, ' mag:14:6, ' phase:14:6);
end;
end;
end;
writein:

(******************************************************************************
Call to graphing procedure
******************************************************************************)
cilsr;
title: 'Magnitude Response (in dB)';
graph3d(data,xy,title);
cilsr;
title: 'Phase Response';
graph3d(data2,xy,title);
cilsr;

******************************************************************************
Calculation of group delays of filter
******************************************************************************

(******************************************************************************
Simulation of derivative by linear difference equation
******************************************************************************)
del1:=(xy[1,2]-xy[1,1])/nptsx;
del2:=(xy[2,2]-xy[2,1])/nptsy;
for i:=1 to nptsx do
  begin
    wt1:=(i-1)*(xy[1,2]-xy[1,1])/(nptsx-1)*xy[1,1];
    for j:=1 to nptsy do
      begin
        wt2:=(j-1)*(xy[2,2]-xy[2,1])/(nptsy-1)*xy[2,1];
        k:=(i-1)*nptsy+j;
        nextx:=k+nptsy;
        nexty:=k+1;
        if nexty>nptsy then nexty:=nexty-2;
        if nextx>npts then nextx:=nextx-2*nptsy;
        data[1,k]:=wt1;
        data[2,k]:=wt2;
      end;
    data2:=data2[3,nextx]+data2[3,k];
  end;

(******************************************************************************
Scaling of vertical change elements for periodic wrap-around
******************************************************************************)

if abs(del1)>p1 then
  begin
    if del1>0.0 then
      del1:=del1-2*p1
    else
      del1:=del1+2*p1;
  end;
379 if abs(delp2)>pl*1.2 then
380 begin
381 if delp2>0.0 then
382 delp2:=delp2-2*pi
383 else
384 delp2:=delp2+2*pi;
385 end;
386 data[3,k]:=delp1/deltal;
387 delay[3,k]:=delp2/delta2;
388 if j-nptsy then delay[3,k]:=delay[3,k];
389 if l-nptsx then data[3,k]:=data[3,k];
390 end;
391 end;
392
393 {*******************************
394 Call to Graphing Subroutines
395 ********************************}
396
397 clrscr;
398 title:='   Group Delay 1';
399 graph3d(data,xy,title);
400 clrscr;
401 title:='   Group Delay 2';
402 graph3d(delay,xy,title);
403 clrscr;
404
405 {*******************************
406 End of mainline program
407 ********************************}
408
410 end.
411
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
program chebgen_root;

{ This program calculates the root trajectory of a two dimensional digital filter derived from the fifth chebyshev argument (general) 
Program By - Tim Kent Date July 87
Language - Turbo Pascal 3.0
Filter Theory - Dr. J. Soltis, T.Kent 

const
ordmax=20;
npts=150; {**-Needed for graph2d}
nptsord=3000;
type
form = array [1..2,1..nptsord] of real; {**}
form3 = array [1..2] of real;
form4 = string [40];

var
data :form;
emax,eps, fact0, gamma1, gamma2, cl, x, y, s, t, pl, p2, theta, n : real;
I, j, k, m, l : integer;
u, v, cu, cv, zu, zv, su, sv : array [1..ordmax] of real;
z1, z2, t1, t2 : form3;
title : form4;

{ Subprograms
**********
{ $1 graph2d.pas

procedure csub(var z1, z2, z3: form3);
begin
z1[1] := z2[1] - z3[1];
end;

procedure cmult(var z1, z2, z3: form3);

Procedure to simulate complex multiplication

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
begin
z1[1]=z2[1]*z3[1]-z2[2]*z3[2];
end;

begin
    mag:=z2[1]*z2[1]+z2[2]*z2[2];
    z1[1]:=z2[1]/mag;
    z1[2]:=-z2[2]/mag;
end;

begin
    z1[1]:=exp(z2[1])*cos(z2[2]);
    z1[2]:=exp(z2[1])*sin(z2[2]);
end;

begin
    z1[1]:=cos(theta);
    z1[2]:=sin(theta);
end;

begin
    var
    end;
x:=z[i];
y:=z[i+1];
b:=x*y-x+y-1;
c:=2*x*y;
d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
if c>0 then
  sign:=1
else
  sign:=-1;
e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
f:=ln(sqrt(d*d+e*e));
if (abs(e)<le-100 and (d<0.0)) then
  t:=pi
else
  if abs(d)<le-100 then
    begin
      if e>0 then
        sign:=1
      else
        sign:=-1;
      t:=pi/2*sign;
    end
  else
    begin
      t:=arctan(e/d);
      if d<0 then
        if e>0 then
          t:=t+pi
        else
          t:=t-pi;
      end;
  end;
g:=t;
2[i]:=f;
2[i+1]:=g;
end;

{**********************************
End of subprograms
**********************************

********************************************************
Mainline program
********************************************************

begin
  clrscr;
({**********************************
Input filter specifications
**********************************

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
writeIn('Root Trajectory - Type I - General Chebyshev Argument ');
writeIn;
write('Enter order of filter (+ve ',ordmax,'/2) : ');
readIn(n);
m:=trunc(2*n);
write('Enter passband ripple Amax : ');
readIn(amax);
write('Enter Gamma1 (0<gamma<1.0) : ');
readIn(gamma1);
write('Enter Gamma2 (0<gamma<1.0) : ');
readIn(gamma2);
gamma1:=cos(gamma1*pi);
gamma2:=cos(gamma2*pi);
writeIn;
eps:=sqrt(exp(0.1*amax*ln(10))-1);
{*******************************
Calculation of chebyshev poles
*******************************}
fact0:=-ln(1/eps+sqrt(1/eps/eps+1))/n/2;
for k:=1 to m do
begin
u[k]:=(2*k-1)*pl/4/n;
v[k]:=fact0;
end;
{*******************************
Calculation of Cosine of chebyshev poles
*******************************}
for k:=1 to m do
begin
cu[k]:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
cv[k]:=-0.5*sin(u[k])*(exp(v[k])-exp(-v[k]));
su[k]:=0.5*sin(u[k])*(exp(v[k])+exp(-v[k]));
sv[k]:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
t[1]:=cu[k];
t[2]:=cv[k];
cmult(t2,t1,t1);
cu[k]:=t2[1];
cv[k]:=t2[2];
t[1]:=su[k];
t[2]:=sv[k];
cmult(t2,t1,t1);
su[k]:=t2[1];
sv[k]:=t2[2];
end;
{*******************************
Calculation of Zsi — ( Arccosh(z))
*******************************}
for k := 1 to m do
begin
  cu[k] := gamma1*cu[k] + gamma2*sv[k];
  cv[k] := gamma1*cv[k] + gamma2*sv[k];
  z1[1] := cu[k];
  z1[2] := cv[k];
  erccosh(z1, z2);
  zu[k] := z2[1];
  zv[k] := z2[2];
end;

Loop to generate root trajectories for all factors

for l := 1 to m do
begin
  if (zu[l] < 0.0) then
begin
    zu[l] := -zu[l];
    zv[l] := -zv[l];
  end;
  t1[l] := zu[l];
  t1[2] := zv[l];
  cexp(t2, t1);
  t1[1] := 2 * t2[l];
  t1[2] := 2 * t2[2];
end;

Loop to generate points of trajectories

for i := 1 to npts do
begin
  theta := (i-1)/(npts-1)*2*pi;
  cart(z2, theta);
  csub(t2, t2[1], z2);
  cinv(z1, z2);
  data[1, i*(1-1)*npts] := z1[1];
  data[2, i*(1-1)*npts] := z1[2];
end;

Call to graphing routine

c1scr;
title := 'Root Trajectories';
graph2d(data, m, title);
c1scr;

End of Mainline Program
end.

I program chebgen_tran_func;

This program calculates the expanded transfer function of a two dimensional digital filter derived from the fifth chebyshev argument (general)

Program By - Tim Kent Date Jul.87

Language - Turbo Pascal 3.0

Filter Theory - Dr. J. Soltis, T. Kent

II «

const

ordmax=20;
type

form3 = array [1..2] of real;

var

num,ammax,sum,eps,n,fact0,gamma1,gamma2,sign,
c1,x,y,s,t,w1,w2,pi,p2,scale : real;
l,j,k,m,nextx,nexty : integer;
u,v,cu,cv,su,sr,zu,zv array [1..ordmax] of real;
z1,z2,t1,t2 : form3;
flag:boolean;
temp:char;
filename : string[13];
output: text;
a,b,c,d array [0..ordmax,0..ordmax] of real;

Subprograms

procedure pause;

Procedure to temporarily halt program execution

begin

writeLn;
writeLn('press any key to continue');
repeat until keypressed;
writeLn;
end;

procedure cexp(var x,y:real);

Procedure to simulate complex exponentiation

begin


55 var
56 s,t :real;
57 begin
58 t:=x;
59 s:=y;
60 x:=exp(t)*cos(s);
61 y:=exp(t)*sin(s);
62 end;
63
64 procedure cmult(var z1,z2,z3:form3):
65 (*-------------------------------*
66 Procedure to simulate complex multiplication
67 *-------------------------------*)
68 begin
69 z1[1]:=z2[1]*z3[1]-z2[2]*z3[2];
70 z1[2]:=z2[1]*z3[2]+z2[2]*z3[1];
71 end;
72
73 procedure arccosh(var z1,z2:form3):
74 (*-------------------------------*
75 Procedure to calculate arccosh of z
76 *-------------------------------*)
77 var
78 x,y,b,c,d,e,t:real;
79 begin
80 x:=z1[1];
81 y:=z1[2];
82 b:=x*x-y*y-1;
83 c:=2*x*y;
84 d:=x+sqrt((sqrt(b*b+c*c)+b)/2);
85 if c<>0 then
86 sign:=1
87 else
88 sign:=-1;
89 e:=y+sign*sqrt((sqrt(b*b+c*c)-b)/2);
90 f:=ln(sqrt(d*d+e*e));
91 if ((abs(e)<1e-100)and(d<>0.0)) then
92 begin
93 if abs(d)<1e-100 then
94 begin
95 t:=pi/2*sign;
96 end
97 else
98 t:=pi/2*sign;
99 end
100 end
101 if abs(d)<1e-100 then
102 begin
103 if e<>0 then
104 sign:=1
105 else
106 sign:=-1;
107 t:=pi/2*sign;
108 end
else
begin
  t:=-arctan(e/d);
  if d<0 then
    if e>0 then
      t:=-t+pi
    else
      t:=-t-pi;
  end;
end;
g:=t;
z2[1]:=f;
z2[2]:=g;
end;

End of subprograms

Mainline Program

begin
clrscr;

Input filter design parameters

writeln('Expanded Transfer Function Generator');
writeln(' General Chebyshev Argument');
writeln;
writeln('Enter order of filter (+ve <',ordmax,'/2) : ');
readln(n);
m:=trunc(2*n);
write('Enter passband ripple A max : ');
readln(amax);
write('Enter Gamma 1 (0<gamma<1.0) : ');
readln(gamma1);
write('Enter Gamma 2 (0<gamma<1.0) : ');
readln(gamma2);
gamma1:=cos(gamma1*pi);
gamma2:=cos(gamma2*pi);
write;
eps:=sqrt(exp(0.1*amax*ln(10))-1);

Calculation of chebyshev poles

fact0:=-ln(1/eps*sqrt(1/eps/eps+1))/n/2;
for k:=1 to m do
begin
  u[k]:=(2*k-1)*pi/4/n;
  v[k]:=fact0;
end;

Calculation of Cosine of chebyshev poles

for k:=1 to m do
begin
  cu[k]:=0.5*cos(u[k])*(exp(v[k])+exp(-v[k]));
  cv[k]:=0.5*sin(u[k])*(exp(v[k])-exp(-v[k]));
  su[k]:=0.5*sin(u[k])*(exp(v[k])+exp(-v[k]));
  sv[k]:=0.5*cos(u[k])*(exp(v[k])-exp(-v[k]));
  t1[1]:=cu[k];
  t1[2]:=cv[k];
  cmult(t2,t1,t1);
  cu[k]:=t2[1];
  cv[k]:=t2[2];
  t1[1]:=su[k];
  t1[2]:=sv[k];
  cmult(t2,t1,t1);
  su[k]:=t2[1];
  sv[k]:=t2[2];
end;

Calculation of Zsi — (Arccosh(z))

for k:=1 to m do
begin
  cut[k]:=gamma1*cu[k]+gamma2*su[k];
  cv[k]:=gamma1*cv[k]+gamma2*sv[k];
  z[1]:=cu[k];
  z[2]:=cv[k];
  arccosh(z1,z2);
  zu[k]:=z2[1];
  zv[k]:=z2[2];
end;

Calculation of constants in filter polynomial factors

for l:=1 to m do
begin
  if (zu[l]<0.0) then
  begin
    zu[l]:=-zu[l];
    zv[l]:=-zv[l];
  end;
\texttt{cexp(zu[i],zv[i]);
zu[i]=-2.0*zu[i];
zv[i]=-2.0*zv[i];
end;

(*******************************************************************************
initialize arrays for polynomial expansion
*******************************************************************************)

for i:=0 to m do
  for j:=0 to m do
    begin
      a[i,j]:=0.0;
      b[i,j]:=0.0;
      c[i,j]:=0.0;
      d[i,j]:=0.0;
    end;
a[0,0]:=1.0;

(*******************************************************************************
loop to generate expanded polynomial
*******************************************************************************

for k:=1 to m do
  begin
    for i:=1 to k do
      for j:=0 to k do
        begin
          i:=i-1;
          c[i,j]:=c[i,j]+a[i,j];
          d[i,j]:=d[i,j]+b[i,j];
          c[j,i]:=c[j,i]+a[j,i];
          d[j,i]:=d[j,i]+b[j,i];
        end;
    for i:=0 to k do
      for j:=0 to k do
        begin
          c[i,j]:=c[i,j]+a[i,j]*zu[k]-b[i,j]*zv[k];
          d[i,j]:=d[i,j]+a[i,j]*zv[k]+b[i,j]*zu[k];
        end;
    for i:=0 to k do
      for j:=0 to k do
        begin
          a[i,j]:=c[i,j];
          b[i,j]:=d[i,j];
          c[i,j]:=0.0;
          d[i,j]:=0.0;
        end;
  end;

(*******************************************************************************
scale function for B(0,0) = 1.0
*******************************************************************************

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
num=1.0/a[0,0];
for i=0 to m do
    for j=0 to m do
        a[i,j]=a[i,j]*num;

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Scale amplitude response for desired 0 dB
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

writeln('Do you wish amplitude response');
write(' scaled to 0 dB at w1T = w2T = 0.0 ? (Y/N)');
readln(temp);
temp:=upcase(temp);
if temp='Y' then
    begin
        sum=0.0;
        for i=0 to m do
            for j=0 to m do
                sum=sum+a[i,j];
        num=sum;
    end;
if temp='N' then
    begin
        writeln('Do you wish amplitude response');
        write(' scaled to 0 dB at w1T = w2T = pi ? (Y/N)');
        readln(temp);
temp:=upcase(temp);
if temp='Y' then
    begin
        sum=0.0;
        sign=-1.0;
        for i=0 to m do
            begin
                sign=-sign;
                for j=0 to m do
                    begin
                        sum=sum+sign*a[i,j];
                    end;
                end;
        num=sum;
    end;
end;
end;
end;
end;
end;
writeln;

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Output Expanded Transfer Function
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
writeln('Enter Filename for output of coefficients ');
readln(filename);
assign(output,filename);
rewrite(output);
writeln(output,m);

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
325 writeln(output,m);
326 writeln(output,num:12:11);
327 for i:=0 to m do
328  for j:=0 to m do
329    begin
330      writeln(output,a[i,j]:12:11);
331    end;
332  close(output);
333 end.
APPENDIX 8

Image Filtering Program Source Code
This is a program to filter digital images of various sizes (128x128, 256x256, 512x512) by a two-dimensional first quadrant upward diagonal HIR filter.

FILTER INPUT - this program expects to read a disk file that contains the coefficients of the transfer function in expanded form. After the integer order with respect to z1 and z2, respectively, the real numbers representing the coefficients should be available in the following order:

num scale factor

c(0,0), c(0,1), c(0,2), ..., c(0,n), c(1,0), ..., c(1,n), ..., c(m,n)

c(0,0) = 1.0

IMAGE INPUT and OUTPUT - the program will read and write to and from binary image files.

written by Tim Kent
July 87

real c(20,20), y(512,512), x(512,512), max, min

integer m, n, a, b, size
character*60 filter, filein, fileout
character*1 point(128)

print *, 'Enter Filter Coefficient Filename'
read 10, filter

open(1, file=filter, status='old')
read(1,*) m
read(1,*) n
if (m.gt.20.or.n.gt.20) then
   print *, ''
   print *, 'FILTER TOO LARGE FOR PROGRAM TO FILTER.'
   stop
endif
read(1,*) scale
do i=1, n+1
do j=1, m+1
   read(1,*) c(i,j)
end do
end do
close(1)

print *, 'Enter Image Input Filename'
read 10, filein
print *, 'Enter Size of Input Image: 1) 128x128'
print *, ' 2) 256x256'
print 69
format(*$','
read(*,*) isize
print *, 'Enter Image Output Filename'
read 10, fileout

if (isize.eq.1) isize=128
if (isize.eq.2) isize=256
if (isize.eq.3) isize=512

open(1, file=filein, recl=128,
   status='old', form='formatted')
open(2, file=fileout, recl=128, recordtype='fixed',
   status='new', form='formatted', carriagecontrol='none')

do i=1,isize
   do j=1,isize/128
      print *, i,j
      read(1,50) point
      50 format(128a)
      do k=1,128
         x(1,(j-1)*128+k)=float(ichar(point(k)))
      end do
   end do
end do

max=-1.0e20
min=1.0e20

do i=1,isize
   do j=1,isize
      output=scale*x(i,j)
      do k=2,m+1
         k2=1-k+1
         if (k2.gt.0) then
            print *, i,j,' ',output,' ','c(k,1)',' ','y(k2,j)
         endif
      end do
   end do
end do

max=-1.0e20
min=1.0e20

12=1-j+1
if (12.gt.0) then
   print *, i,j,' ',output,' ','c(1,1)',' ','y(1,12)
   output=output-c(1,1)*y(1,12)
endif
end do

do k=2,m+1
   do l=2,n+2-k
      k2=1-k+1
      12=1-j+1
      if (k2.gt.0.and.12.gt.0) then
         print 63, i,j,k2,12,c(k,1),y(k2,12)
       63 format(13,' ','13,' ','13,' ','13,' ','e12.5,' ','e12.5)
         output=output-c(k,1)*y(k2,12)
      endif
   end do
end do

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
109      end do  
110   end do  
111  c   print*, l,J,output  
112   y(l,J)=output  
113   if (abs(output).gt.max) max=abs(output)  
114   if (abs(output).lt.min) min=abs(output)  
115          end do  
116      end do  
117  
118  c   print *, max, min  
119   range=(max-min)/255.0  
120   do l=I,Isize  
121       do j=1,Isize/128  
122           do k=1,128  
123               ival=int(abs(y(l,(j-1)*128+k)-min)/range)  
124           c   print *,l,(j-1)*128+k,ival,y(l,(j-1)*128+k),max,min  
125          end do  
126   c   point(k)=char(ival)  
127      write(2,50) point  
128      end do  
129      end do  
130   
131   stop  
132  end  
133  
134  
135  
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
REFERENCES


VITA AUCTORIS

Timothy John Kent, a resident of Windsor, Ontario, graduated with an Ontario Secondary School Honour Graduation Diploma from John L. Forster Secondary School in 1982. He received the B.A.Sc. degree in Electrical Engineering from the University of Windsor in 1986. During his undergraduate studies he maintained a first class honours standing as well as being designated an Assumption University Scholar. Currently, he is in the Faculty of Graduate Studies, University of Windsor working towards the degree of M.A.Sc. in Electrical Engineering. He has received numerous awards throughout his academic career including a Norah Cleary Memorial Scholarship for undergraduate studies and his current Natural Sciences and Engineering Research Council of Canada Postgraduate Scholarship. Upon completion of his current course of study he will be employed in industry as a research engineer.