Title:	***A Helpful Filter Design Approach***
Author:	AM1 LLC
Abstract:	I have wanted to attempt to do a filter design using the iteration method described by Orchard [1] for many years, but never had a situation posed where I could justify digging into the details. That situation finally arose where I needed the normalized LC component values for a range of *unequally terminated* inverse Chebyshev lowpass filters. This technique seems to work so well that I thought it worth documenting. Note: Most of the design details in [1] are specifically aimed at Cauer (elliptic) lowpass filters rather than the more general case. Consequently, most of the paper assumes that the poles and zeros of the characteristic function $K(s)$ are on the $j\omega$-axis which can be a bit confusing unless you are intimately familiar with the Darlington filter design approach. The derivations provided herein address the more general case.
Disclaimer:	*The technical information and opinions provided in this document are believed to be accurate, but no warranty, assurance, or indemnification of any kind is granted nor implied as to the completeness, accuracy, or suitability of this information for any purpose whatsoever. Upon receipt of this information or any derivatives of it, Boeing acknowledges that this information may contain inaccuracies or errors, and AM1 expressly excludes any liability for any such inaccuracies or errors to the fullest extent permitted by law. AM1 assumes no liability whatsoever for any damages or losses resulting from the use or reliance upon the information contained or absent herein.* *Furthermore, AM1 provides no warranty, assurance, or indemnification of any kind in regard to the possible infringement of the provided material upon the intellectual property rights of others. AM1 assumes no liability of any kind concerning such property rights. Upon receipt of this information or any derivatives of it, Boeing acknowledges that this information may possibly infringe upon the intellectual property rights of others, and is itself solely responsible for the proper use of the information and opinions contained herein.*
Date:	13 April 2015
Version:	1.02

1 Introduction

Filter design usually begins as an approximation problem. The designer can choose to stay within established filter norms (e.g., Butterworth, Chebyshev), or venture outside where more general transfer functions are in play and or unusual filter termination resistance values are used. While there are closed-form design equations for the design of unequally terminated Butterworth and Chebyshev filters, these equations are usually not available for the general case. This was the complication I encountered; detailed design information was available for equally terminated inverse Chebyshev filters, but not for the unequally terminated case.

2 Supporting Details

Reference [1] goes more deeply into the classical filter design theory based upon Darlington's method than is really needed for the general case. Perhaps the most valuable tenet provided in [1] is the use of ABCD matrices to formulate the design solution. In the case of a lossless two-port network as shown in Figure 1, the ABCD formulation provides

$$\begin{bmatrix} v_1 \\ i_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} v_2 \\ i_2 \end{bmatrix} \tag{1}$$

The associated power-related transfer function of interest is given by[1]

[1] This is the *reciprocal* of the relationship used in [1] so do not get confused.

$$H(s) = \frac{2v_2}{E}\sqrt{\frac{R_1}{R_2}} \tag{2}$$

and in terms of decibel power-gain,

$$\begin{aligned} G_{dB} &= 10\log_{10}\left[\frac{4R_1}{R_2}\left|\frac{v_2}{E}\right|^2\right] \\ &= 10\log_{10}\left[\frac{4R_1}{R_2}\right] - 10\log_{10}\left[\left|\frac{E}{v_2}\right|^2\right] \end{aligned} \tag{3}$$

In the situation where $R_2 \to \infty$, the first term in (3) is discarded and attention is focused on the voltage-gain term alone. It is convenient to take E = 1 without any loss of generality. In terms of ABCD components,

$$\frac{E}{v_2} = \frac{1}{v_2} = A + \frac{B}{R_2} + R_1 C + \frac{R_1}{R_2} D \tag{4}$$

The calculation method employed herein ultimately makes use of the Newton method and partial derivatives with respect to each network element value e_n are consequently needed. From (3),

$$\begin{aligned} \frac{\partial G_{dB}}{\partial e_n} &= \frac{-10}{\log_e(10)}\frac{\partial}{\partial e_n}\left[\log_e\left(\frac{1}{v_2\overline{v_2}}\right)\right] \\ &= \frac{-10}{\log_e(10)}\frac{\partial}{\partial e_n}\left[\log_e\left(\frac{1}{v_2}\right) + \log_e\left(\frac{1}{\overline{v_2}}\right)\right] \\ &= \frac{-10}{\log_e(10)}\left[v_2\frac{\partial}{\partial e_n}\left(\frac{1}{v_2}\right) + \overline{v_2}\frac{\partial}{\partial e_n}\left(\frac{1}{\overline{v_2}}\right)\right] \\ &= \frac{-20}{\log_e(10)}\mathrm{Re}\left[v_2\frac{\partial}{\partial e_n}\left(\frac{1}{v_2}\right)\right] \end{aligned} \tag{5}$$

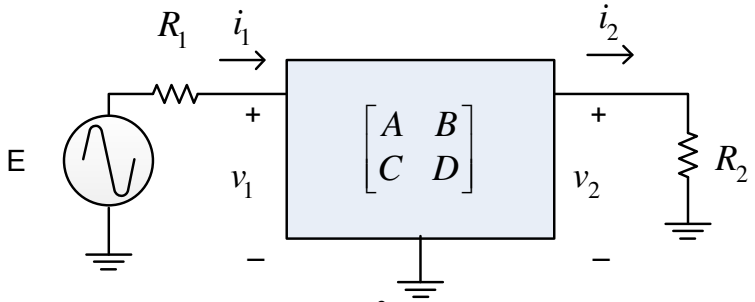


Figure 1 Definition terms for ABCD matrix description[2]

[2] From U22332 Figures.vsd.

The quantity v_2^{-1} is directly available from (4). In order to compute the partial derivatives required in (5), we turn our attention now to Figure 2 and Figure 3. For the shunt-admittance case in Figure 2, the resultant ABCD network is given by

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{Bmatrix} \left[A_1A_2 + B_1\left(YA_2 + C_2\right)\right] & \left[A_1B_2 + B_1\left(YB_2 + D_2\right)\right] \\ \left[C_1A_2 + D_1\left(YA_2 + C_2\right)\right] & \left[C_1B_2 + D_1\left(YB_2 + D_2\right)\right] \end{Bmatrix} \tag{6}$$

from which

$$\frac{\partial}{\partial Y}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} B_1A_2 & B_1B_2 \\ D_1A_2 & D_1B_2 \end{bmatrix} \tag{7}$$

Based upon Figure 1, (2), and (7)

$$\frac{E}{v_2} = \frac{1}{v_2} = \left[R_1C + \frac{R_1}{R_2}D + A + \frac{B}{R_2}\right] \tag{8}$$

which leads to

$$\frac{\partial}{\partial Y}\left(\frac{E}{v_2}\right) = R_1D_1A_2 + \frac{R_1}{R_2}D_1B_2 + B_1A_2 + \frac{1}{R_2}B_1B_2 \tag{9}$$

For the series-impedance case shown in Figure 3, the resultant derivative is

$$\frac{\partial}{\partial Z}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} A_1C_2 & A_1D_2 \\ C_1C_2 & C_1D_2 \end{bmatrix} \tag{10}$$

from which

$$\frac{\partial}{\partial Z}\left(\frac{E}{v_2}\right) = \left[R_1C_1C_2 + \frac{R_1}{R_2}C_1D_2 + A_1C_2 + \frac{1}{R_2}A_1D_2\right] \tag{11}$$

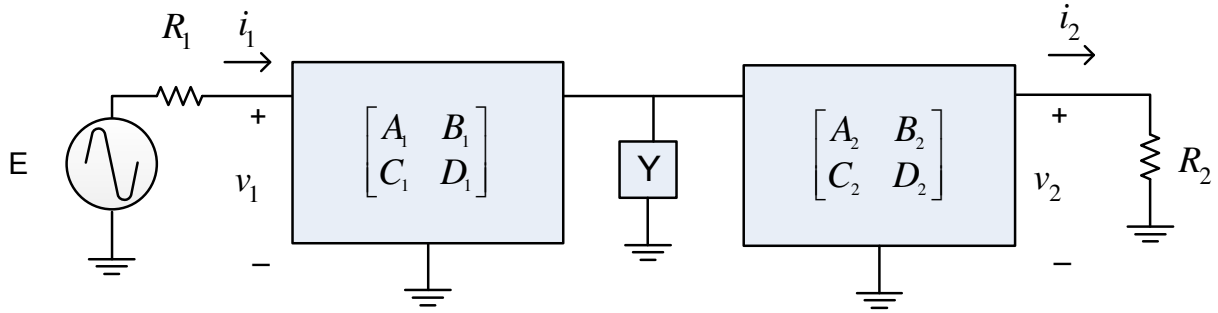


Figure 2 Cascaded network with shunt admittance Y present

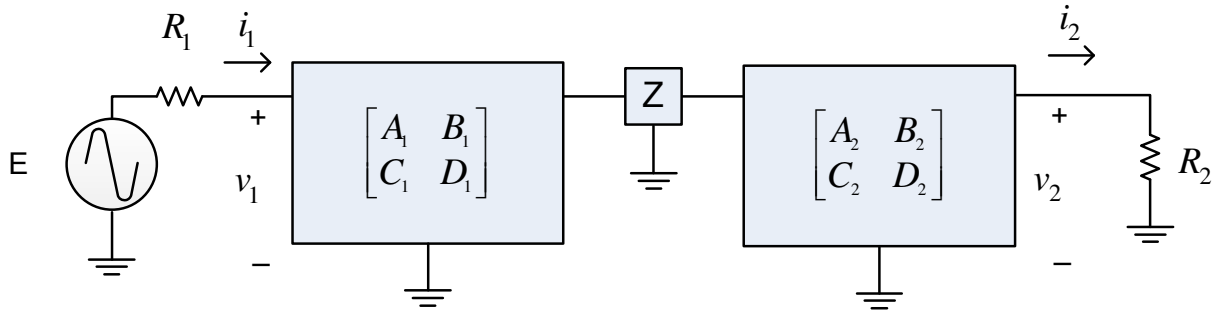


Figure 3 Cascaded network with series impedance Z present

In order to get the partial derivatives with respect to the component values e_k, the chain-rule must be used. In the case where admittance Y is a shunt capacitor C_{shunt}

$$\frac{\partial Y}{\partial C_{shunt}} = j\omega \tag{12}$$

If the shunt admittance is a series-LC trap,

$$Y = \frac{1}{\frac{1}{sC_{trap}} + sL_{trap}} = \frac{j\omega C_{trap}}{1 - \left(\frac{\omega}{\omega_{trap}}\right)^2} \tag{13}$$

which leads to

$$\frac{\partial Y}{\partial C_{trap}} = \frac{j\omega}{1 - \left(\frac{\omega}{\omega_{trap}}\right)^2} \tag{14}$$

If the series impedance is an inductance

$$\frac{\partial Z}{\partial L_{series}} = j\omega \tag{15}$$

In the case where the series impedance is a series LC-trap (parallel L and C),

$$Z = \frac{1}{sC_{trap} + \frac{1}{sL_{trap}}} = \frac{j\omega L_{trap}}{1 - \left(\frac{\omega}{\omega_{trap}}\right)^2} \tag{16}$$

which leads to

$$\frac{\partial Z}{\partial L_{trap}} = \frac{j\omega}{1 - \left(\frac{\omega}{\omega_{trap}}\right)^2} \tag{17}$$

It is assumed here that the trap resonant frequencies are known a priori as part of the transfer function approximation step. Consequently,

$$\omega_{trap} = \frac{1}{\sqrt{L_{trap} C_{trap}}} \tag{18}$$

and this relationship can be used to eliminate one of the unknowns during the iterative calculations for each trap.

Recapping then, the needed partial derivatives are given by (5) where v_2 is available from (8) as

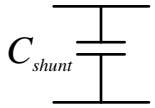
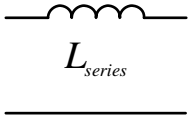
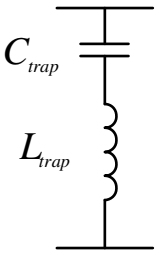
$$v_2 = \frac{1}{R_1 C + \frac{R_1}{R_2} D + A + \frac{B}{R_2}} \tag{19}$$

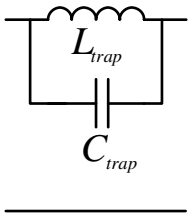
and the partial derivatives with respect to *Y* and *Z* are given by (9) and (11) respectively. The chain rule must then be applied to these in order to translate them into partial derivatives with respect to e_n per (12), (14), (15), or (17) as appropriate.

3 Iterative Calculation

Assume now that a power-gain transfer function goal $G_{goal}(f)$ is known and that a filter circuit topology has been chosen which contains the correct number of poles and zeros to realize this transfer function. The iterative calculation consists of using a sufficient number of (fixed) frequency points to enable a least-squares solution to take place using Newton's Method in an iterative manner.

Table 1 Summary of Lowpass Section Types

Type	Lowpass Section Type	Partial Derivative	ABCD
1		$\frac{\partial Y}{\partial C_{shunt}} = j\omega$ (20)	$\begin{bmatrix} 1 & 0 \\ j\omega C_{shunt} & 1 \end{bmatrix}$ (21)
2		$\frac{\partial Z}{\partial L_{series}} = j\omega$ (22)	$\begin{bmatrix} 1 & j\omega L_{series} \\ 0 & 1 \end{bmatrix}$ (23)
3		$\frac{\partial Y}{\partial C_{trap}} = \frac{j\omega}{1 - \left(\frac{\omega}{\omega_{trap}}\right)^2}$ (24)	$\begin{bmatrix} 1 & 0 \\ \frac{j\omega C_{trap}}{1 - \left(\frac{\omega}{\omega_{trap}}\right)^2} & 1 \end{bmatrix}$ (25)

Type	Lowpass Section Type	Partial Derivative	ABCD
4		$\frac{\partial Z}{\partial L_{trap}} = \frac{j\omega}{1-\left(\frac{\omega}{\omega_{trap}}\right)^2}$ (26)	$\begin{bmatrix} 1 & \frac{j\omega L_{trap}}{1-\left(\frac{\omega}{\omega_{trap}}\right)^2} \\ 0 & 1 \end{bmatrix}$ (27)

Let the set of fixed evaluation frequencies be denoted by f_k and the difference between the goal attenuation values and the ones at iteration-k denoted by

$$G_{err}(f_k) = G_{goal}(f_k) - G_{dB}(f_k) \tag{28}$$

The (non-square) matrix of partial derivatives has the form

$$J = \left[\frac{\partial G_{dB}(f_k)}{\partial U_n} \frac{\partial U_n}{\partial e_n} \right] \tag{29}$$

where the matrix rows correspond to the different evaluation frequencies f_k and the matrix columns correspond to the circuit element values being iterated. The element values after the p^{th} iteration are given by

$$[e_n]_p = [e_n]_{p-1} + \gamma lms\left(J_{p-1}, G_{err,p}\right) \tag{30}$$

where γ is a numerical gain term having a magnitude < 1 and *lms* designates a least-mean-square solution for the matrix and vector involved.

Table 2 Normalized LC Values for 3rd-Order Unloaded Inverse Chebyshev Lowpass[3]

Stopband, dB	C_1	C_2	C_3	L
25	0.494	0.42	2.495	1.784
30	0.732	0.323	3.000	2.322
35	0.992	0.255	3.631	2.941
40	1.287	0.204	4.39	3.671
45	1.628	0.165	5.314	4.536
50	2.028	0.135	6.472	5.561

Table 3 Normalized LC Values for 5th-Order Unloaded Inverse Chebyshev Lowpass[4]

Stopband, dB	C_1	C_2	C_3	C_4	C_5	L_1	L_2
30	0.0021	0.4960	1.3204	0.6302	1.7103	0.6966	1.4351
40	0.1743	0.3385	1.7652	0.4419	2.1750	1.0205	2.0467
45	0.2616	0.2874	2.0303	0.3786	2.4468	1.2020	2.3887
50	0.3533	0.2458	2.3203	0.3271	2.7592	1.4055	2.7647
60	0.5453	0.1852	3.0065	0.2495	3.4924	1.8654	3.6255

[3] Computed using u21373_inverse_chebyshev_synthesis.m followed by U22328 N=3 Inverse Cheby Unloaded.mcd.
[4] Computed using u22336_inverse_chebyshev_iterated_synthesis.m.

4 References

1. H.J. Orchard, “Filter Design by Iterated Analysis,” *IEEE Trans. Circuits and Systems*, Nov. 1985, U20631.

5 Appendix: MATLAB Script for Unloaded Case ($R_2 = \infty$)

The first portion of the script computes the poles and zeros of the inverse Chebyshev lowpass filter of interest. There are a number of calculations done pertaining to characteristic functions, etc.

```
%================== u22336_inverse_chebyshev_iterated_synthesis.m ==================
%
%
%   J.A. Crawford
%   20 March 2014
%
%   Earlier synthesis program appended with iteration-based design of
%   5th order inverse Chebyshev lpf as an unloaded LC network
%   First attempt at using Orchard's iterative design technique and this
%   example shows that it works well.
%
%   A more general script is required if other load impedance values are
%   needed, or if the order needs to be changed.
%
%   Pretty cool, I must say. Anxious to incorporate this method into
%   a general synthesis tool in C#. I've had need for being able to
%   use an arbitrary load resistance on quite a few past occassions.
%
%   Don't get good convergence for stopband attenuations less than about
%   30 dB for some reason. Otherwise, fantastic even up to 110 dB
%   stopband attenuations. Found that the reason is that C1 must be allowed
%   to go negative for these lower stopband attenuations.
%
fil_order= 5;
Astop_dB= 80;
jx= i;

Astop= 10^(0.1*Astop_dB);
epsilon= sqrt(1/(Astop-1));

odd_order= (mod(fil_order,2)==1);       % 1 if odd-order, otherwise 0

%
%   Computes poles and zeros of inverse Chebyshev
%
aa= sinh( (1/fil_order)*asinh(1/epsilon) );
bb= cosh( (1/fil_order)*asinh(1/epsilon));
%
nden= fil_order;
nnum= floor(fil_order/2);
```

```
kk=1:nden;
theta= (2*kk-1)*pi/(2*fil_order);
cheby_poles= -aa*sin(theta) + jx*bb*cos(theta)

kk=1:nnum;
inv_cheby_poles=1./cheby_poles;                    % All poles in the left-half plane
inv_cheby_zeros= jx./cos( (2*kk-1)*pi/(2*fil_order) );
inv_cheby_zeros(nnum+kk)= conj(inv_cheby_zeros(kk));    % All zeros on jw axis

gam= 1;
for jk=1:length(inv_cheby_poles)
   gam= gam * inv_cheby_poles(jk);
end
for jk=1:length(inv_cheby_zeros)
   gam= gam / inv_cheby_zeros(jk);
end

%
%   Sweep filter
%
Npts= 512*4;
fswp= 10.^(-3+5*(1:Npts)/Npts);
Hfil= zeros(1,Npts);
tau= zeros(1,Npts);
for jk=1:Npts
   Hcas= 1;
   taux= 0;
   ss= i*2*pi*fswp(jk);
   for ii=1:length(inv_cheby_zeros)
      Hcas= Hcas * (ss - inv_cheby_zeros(ii));
   end
   for ii=1:length(inv_cheby_poles)
      Hcas= Hcas / (ss - inv_cheby_poles(ii));

      px= -real(inv_cheby_poles(ii) );
      py= imag( inv_cheby_poles(ii) );
      taux= taux + px/(px^2 + (abs(ss)-py)^2);
   end
   Hfil(jk)= 10*log10( abs(gam*Hcas)^2 );
   tau(jk)= taux;
end

figure(1);
clf;
p1= semilogx( fswp, Hfil, 'r' );
set( p1, 'LineWidth', 2 );
grid on
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Gain, dB', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Inverse Chebyshev', 'FontName', 'Arial', 'FontSize', 14 );
axis( [0.001, 100, -80, 10] );

%
%   Look at filter group delay
%
figure(2);
clf;
p1= semilogx( fswp, tau, 'r' );
set( p1, 'LineWidth', 2 );
grid on
```

```
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Group Delay', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Inverse Chebyshev', 'FontName', 'Arial', 'FontSize', 14 );
%axis( [0.01, 100, -80, 0] );
%
%=========================================================================
%
%
%
%   Form H(s) polynomial
%
%
%
H_num= 1;
for ii=1:length(inv_cheby_zeros)
   H_num= conv( [1 -inv_cheby_zeros(ii)], H_num );
end
H_den= 1;
for ii=1:length(inv_cheby_poles)
   H_den= conv( [1 -inv_cheby_poles(ii)], H_den );
end

Hx= gam*polyval( H_num, jx*2*pi*fswp ) ./ polyval( H_den, jx*2*pi*fswp );

figure(3);
clf;
p1= semilogx( fswp, 10*log10( abs(Hx).^2 ), 'r' );
set( p1, 'LineWidth', 2 );
title( 'Check on H(w) Using Poles & Zeros',  'FontName', 'Arial', 'FontSize', 14 );
set( p1, 'LineWidth', 2 );
grid on
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Group Delay', 'FontName', 'Arial', 'FontSize', 12 );
%
%=========================================================================
%=========================================================================
%
%
%   Form Characteristic Function K(s)
%
%
K2= abs(1./Hx).^2 - 1;      % Transducer gain is 1/Hx here
K2_dB= 10*log10( K2 );

figure(4);
clf;
p1= semilogx( fswp, K2_dB, 'r' );
set( p1, 'LineWidth', 2 );
grid on
title( '|K(\omega)|^2',  'FontName', 'Arial', 'FontSize', 14 );
%
%   H(s)=  gam * [H_num] / [H_den]
%
%   |H(s)|^2 = gam*gam * [H_num]*[H_num] / ( [H_den]*[H_den] )
%
%   |T(s)|^2 = |1/H(s)|^2 = 1 + |K(s)|^2
%
%   |T(s)|^2 = [H_den]*[H_den]/(gam^2 * [H_num]*[H_num] )
```

```
%           = [p_num]/[p_den]
%
p_den= gam*gam*conv(H_num, H_num)
%
%   H_num only has even-power polynomial coeffs whereas
%   H_den has both, so must take care of conjugation (i.e., negation)
%   of odd-power polynomial coefficients
%
H_den2= H_den;
for ii=length(H_den)-1:-2:1 H_den2(ii)= -H_den2(ii); end
p_num= conv(H_den, H_den2)
%
%   Check this form for |T(s)|^2
%
if( 0 )
   T2check= polyval(p_num, jx*2*pi*fswp) ./ polyval(p_den, jx*2*pi*fswp);
   hold on
   p1= semilogx( fswp, 10*log10( abs(T2check)), 'ko' );
end
%
%   Form numerator polynomial for |K(s)|^2 = |T(s)|^2 - 1
%
p_wrk= p_num;
lx= length(p_wrk)-length(p_den)+1;
jk= length(p_den);
for ii=length(p_wrk):-1:lx
   p_wrk(ii)= p_wrk(ii) - p_den(jk);
   jk= jk - 1;
end
p_wrk= real(p_wrk)

figure(5);
clf;
K2= polyval( p_wrk, jx*2*pi*fswp ) ./ polyval( p_den, jx*2*pi*fswp );
p1= semilogx( fswp, 10*log10( abs(K2) ), 'k' );
set( p1, 'LineWidth', 2 );
grid on
title( '|K|^2 From Polynomials','FontName', 'Arial', 'FontSize', 14 );
%
%   Factor K^2
%
K2_num_roots= roots( p_wrk );
K2_den_roots= roots( p_den );

figure(6);
clf;
plot( real(K2_num_roots), imag(K2_num_roots), 'ro' );
title( 'Roots of |K|^2 Numerator' );
grid on
%
%   Retain only left-plane roots
%
K_num= 1;
mm=1;
for ii=1:length(K2_num_roots)
   if( real(K2_num_roots(ii)) <= 0.001 )
      K_num= conv( [1 -K2_num_roots(ii)], K_num );
      K_num_lhp_roots(mm)= K2_num_roots(ii);
      mm= mm+1;
   end
end
K_den= H_num
```

```
K_num= K_num

figure(7);
clf;
K= (1/gam)*polyval( K_num, jx*2*pi*fswp ) ./ polyval( K_den, jx*2*pi*fswp );
p1= semilogx( fswp, 10*log10( abs(K).^2 ), 'b-' );
set( p1, 'LineWidth', 2 );
grid on
title( 'Final K(\omega)','FontName', 'Arial', 'FontSize', 14 );
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'dB', 'FontName', 'Arial', 'FontSize', 12 );
axis( [0.0001, max(fswp), -60, 80] );

disp( 'Characteristic Function Numerator:' );
disp( num2str( real(K_num), '%5.4e  ' ) );
disp( 'Denominator:' );
disp( num2str( real(K_den), '%5.4e  ' ) );

K_num_lhp_roots
K2_den_roots
%=========================================================================
%
%   Iteratively design filter
%
%   Focus on 5th order filter here
%
C1= 0.5;
C2= 0.25;
C3= 3.0;
C4= 0.25;
C5= 3.0;
wo1= 1.7013;
wo2= 1.0515;
L1= 1/(C2*wo1*wo1);
L2= 1/(C4*wo2*wo2);

freqs= 0.001*10.^(4*(0:50)/50);
Nfreqs=length(freqs);

abcd1= @(s) [ 1 0; s*C1 1 ];
abcd2= @(s) [ 1 1/(s*C2+1/(s*L1)); 0 1];
abcd3= @(s) [ 1 0; s*C3 1 ];
abcd4= @(s) [ 1 1/(s*C4+1/(s*L2)); 0 1];
abcd5= @(s) [ 1 0; s*C5 1 ];

err= 0;
figure(100);
clf;
PD= zeros(Nfreqs,5);    % Partial derivatives
clear g1;
clear g2;
for iter= 1:40
   for ff=1:Nfreqs
      g1(ff)= gam*polyval( H_num, jx*2*pi*freqs(ff) ) ./ polyval( H_den, jx*2*pi*freqs(ff) );
      g1(ff)= 10*log10( abs(g1(ff))^2 );

      ss= jx*2*pi*freqs(ff);

      abcd= abcd1(ss) * abcd2(ss) * abcd3(ss) * ...
           abcd4(ss) * abcd5(ss);
```

```
    g2(ff)= 1/(abcd(1,1) + abcd(2,1));
    g2(ff)= 10*log10( abs(g2(ff))^2 );

    dg(ff)= g1(ff) - g2(ff);
    if( abs(dg(ff)) > 10 )
       dg(ff)= 10*sign(dg(ff));
    end

    %
    %   Get partial derivatives for this frequency
    %   and for each element value
    %
    %   Cap C1
    %
    ss= jx*2*pi*freqs(ff);

    abcd= abcd1(ss)*abcd2(ss)*abcd3(ss)*abcd4(ss)*abcd5(ss);
    apc= (-20/log(10)) / ( abcd(1,1) + abcd(2,1) );

    M1= [ 1 0; 0 1];
    M2= abcd2(ss)*abcd3(ss)*abcd4(ss)*abcd5(ss);
    PD(ff,1)= ( M1(1,2)*M2(1,1) + M2(1,1)*M1(2,2) )*ss*apc;

    %
    %   First LC trap
    %
    M1= abcd1(ss);
    M2= abcd3(ss)*abcd4(ss)*abcd5(ss);
    PD(ff,2)= ( M1(1,1)*M2(2,1) + M1(2,1)*M2(2,1) )*ss/( 1 - abs(ss/wo1)^2 )*apc;

    %
    %   Cap C3
    %
    M1= abcd1(ss)*abcd2(ss);
    M2= abcd4(ss)*abcd5(ss);
    PD(ff,3)= ( M1(1,2)*M2(1,1) + M2(1,1)*M1(2,2) )*ss*apc;

    %
    %   Second LC trap
    %
    M1= abcd1(ss)*abcd2(ss)*abcd3(ss);
    M2= abcd5(ss);
    PD(ff,4)= ( M1(1,1)*M2(2,1) + M1(2,1)*M2(2,1) )*ss/( 1 - abs(ss/wo2)^2 )*apc;

    %
    %   Cap C5
    %
    M1= abcd1(ss)*abcd2(ss)*abcd3(ss)*abcd4(ss);
    M2= [ 1 0; 0 1 ];
    PD(ff,5)= ( M1(1,2)*M2(1,1) + M2(1,1)*M1(2,2) )*ss*apc;
end
%
%   Update element values
%
de= lscov(real((PD)),dg');

gamma= 0.25;

C1= abs(C1 + gamma*de(1));
L1= abs(L1 + gamma*de(2));
C3= abs(C3 + gamma*de(3));
L2= abs(L2 + gamma*de(4));
```

```
    C5= abs(C5 + gamma*de(5));

    C2= abs(1/(wo1^2*L1));
    C4= abs(1/(wo2^2*L2));

    abcd1= @(s) [ 1 0; s*C1 1 ];
    abcd2= @(s) [ 1 1/(s*C2+1/(s*L1)); 0 1];
    abcd3= @(s) [ 1 0; s*C3 1 ];
    abcd4= @(s) [ 1 1/(s*C4+1/(s*L2)); 0 1];
    abcd5= @(s) [ 1 0; s*C5 1 ];

    semilogx( freqs, g1, 'r' );
    hold on
    semilogx( freqs, g2, 'k--' );
    hold on
end

figure(200);
clf;
for ii=1:Npts
    ss= jx*2*pi*fswp(ii);

    abcd= abcd1(ss) * abcd2(ss) * abcd3(ss) * ...
            abcd4(ss) * abcd5(ss);
    gn(ii)= 10*log10( abs(1/(abcd(1,1) + abcd(2,1)))^2 );


    g1(ii)= gam*polyval( H_num, ss ) ./ polyval( H_den, ss );
       g1(ii)= 10*log10( abs(g1(ii))^2 );
end
axes( 'FontName', 'Arial', 'FontSize', 12 );
p1= semilogx( fswp, gn, 'r' );
set( p1, 'LineWidth', 2 );
hold on
p1= semilogx( fswp, g1, 'k--' );
set( p1, 'LineWidth', 2 );
h= gca;
set( h, 'LineWidth', 2 );
grid on
xlabel( 'Normalized Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Gain, dB', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Original Versus Modified Filter Gain', 'FontName', 'Arial', 'FontSize', 12 );
legend( 'Iterative Design Result', 'Ideal from Poles & Zeros' );

C1
C2
C3
C4
C5
L1
L2
```

6 Appendix: MATLAB Script for Unequally Terminated Case (Arbitrary R_2)

```
%================ u22345_inverse_chebyshev_iterated_synthesis.m ==================
%
%   Same as u22336_inverse_chebyshev_iterated_synthesis.m except that
%   arbitrary load impedance can be specified
%
%   J.A. Crawford
%   23 March 2014
%
%   Earlier synthesis program appended with iteration-based design of
%   5th order inverse Chebyshev lpf as an unloaded LC network
%   First attempt at using Orchard's iterative design technique and this
%   example shows that it works well.
%
%   A more general script is required if other load impedance values are
%   needed, or if the order needs to be changed.
%
%   Pretty cool, I must say. Anxious to incorporate this method into
%   a general synthesis tool in C#. I've had need for being able to
%   use an arbitrary load resistance on quite a few past occassions.
%
%   Don't get good convergence for stopband attenuations less than about
%   30 dB for some reason. Otherwise, fantastic even up to 110 dB
%   stopband attenuations. Found that the reason is that C1 must be allowed
%   to go negative for these lower stopband attenuations.
%
fil_order= 5;
Astop_dB= 60;
jx= i;

Astop= 10^(0.1*Astop_dB);
epsilon= sqrt(1/(Astop-1));

odd_order= (mod(fil_order,2)==1);       % 1 if odd-order, otherwise 0

%
%   Computes poles and zeros of inverse Chebyshev
%
aa= sinh( (1/fil_order)*asinh(1/epsilon) );
bb= cosh( (1/fil_order)*asinh(1/epsilon));
%
nden= fil_order;
nnum= floor(fil_order/2);

kk=1:nden;
theta= (2*kk-1)*pi/(2*fil_order);
cheby_poles= -aa*sin(theta) + jx*bb*cos(theta)

kk=1:nnum;
inv_cheby_poles=1./cheby_poles;                     % All poles in the left-half plane
inv_cheby_zeros= jx./cos( (2*kk-1)*pi/(2*fil_order) );
inv_cheby_zeros(nnum+kk)= conj(inv_cheby_zeros(kk));    % All zeros on jw axis

gam= 1;
for jk=1:length(inv_cheby_poles)
   gam= gam * inv_cheby_poles(jk);
end
for jk=1:length(inv_cheby_zeros)
```

```
    gam= gam / inv_cheby_zeros(jk);
end

%
%   Sweep filter
%
Npts= 512*4;
fswp= 10.^(-3+5*(1:Npts)/Npts);
Hfil= zeros(1,Npts);
tau= zeros(1,Npts);
for jk=1:Npts
    Hcas= 1;
    taux= 0;
    ss= i*2*pi*fswp(jk);
    for ii=1:length(inv_cheby_zeros)
        Hcas= Hcas * (ss - inv_cheby_zeros(ii));
    end
    for ii=1:length(inv_cheby_poles)
        Hcas= Hcas / (ss - inv_cheby_poles(ii));

        px= -real(inv_cheby_poles(ii) );
        py= imag( inv_cheby_poles(ii) );
        taux= taux + px/(px^2 + (abs(ss)-py)^2);
    end
    Hfil(jk)= 10*log10( abs(gam*Hcas)^2 );
    tau(jk)= taux;
end

figure(1);
clf;
p1= semilogx( fswp, Hfil, 'r' );
set( p1, 'LineWidth', 2 );
grid on
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Gain, dB', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Inverse Chebyshev', 'FontName', 'Arial', 'FontSize', 14 );
axis( [0.001, 100, -80, 10] );

%
%   Look at filter group delay
%
figure(2);
clf;
p1= semilogx( fswp, tau, 'r' );
set( p1, 'LineWidth', 2 );
grid on
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Group Delay', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Inverse Chebyshev', 'FontName', 'Arial', 'FontSize', 14 );
%axis( [0.01, 100, -80, 0] );
%
%=========================================================================
%
%
%
%   Form H(s) polynomial
%
%
```

```
%
H_num= 1;
for ii=1:length(inv_cheby_zeros)
   H_num= conv( [1 -inv_cheby_zeros(ii)], H_num );
end
H_den= 1;
for ii=1:length(inv_cheby_poles)
   H_den= conv( [1 -inv_cheby_poles(ii)], H_den );
end

Hx= gam*polyval( H_num, jx*2*pi*fswp ) ./ polyval( H_den, jx*2*pi*fswp );

figure(3);
clf;
p1= semilogx( fswp, 10*log10( abs(Hx).^2 ), 'r' );
set( p1, 'LineWidth', 2 );
title( 'Check on H(w) Using Poles & Zeros',  'FontName', 'Arial', 'FontSize', 14 );
set( p1, 'LineWidth', 2 );
grid on
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Group Delay', 'FontName', 'Arial', 'FontSize', 12 );
%
%=========================================================================
%=========================================================================
%
%
%   Form Characteristic Function K(s)
%
%
K2= abs(1./Hx).^2 - 1;      % Transducer gain is 1/Hx here
K2_dB= 10*log10( K2 );

figure(4);
clf;
p1= semilogx( fswp, K2_dB, 'r' );
set( p1, 'LineWidth', 2 );
grid on
title( '|K(\omega)|^2',  'FontName', 'Arial', 'FontSize', 14 );
%
%   H(s)=  gam * [H_num] / [H_den]
%
%   |H(s)|^2 = gam*gam * [H_num]*[H_num] / ( [H_den]*[H_den] )
%
%   |T(s)|^2 = |1/H(s)|^2 = 1 + |K(s)|^2
%
%   |T(s)|^2 = [H_den]*[H_den]/(gam^2 * [H_num]*[H_num] )
%            = [p_num]/[p_den]
%
p_den= gam*gam*conv(H_num, H_num)
%
%   H_num only has even-power polynomial coeffs whereas
%   H_den has both, so must take care of conjugation (i.e., negation)
%   of odd-power polynomial coefficients
%
H_den2= H_den;
for ii=length(H_den)-1:-2:1 H_den2(ii)= -H_den2(ii); end
p_num= conv(H_den, H_den2)
%
%   Check this form for |T(s)|^2
%
```

```
if( 0 )
   T2check= polyval(p_num, jx*2*pi*fswp) ./ polyval(p_den, jx*2*pi*fswp);
   hold on
   p1= semilogx( fswp, 10*log10( abs(T2check)), 'ko' );
end
%
%   Form numerator polynomial for |K(s)|^2 = |T(s)|^2 - 1
%
p_wrk= p_num;
lx= length(p_wrk)-length(p_den)+1;
jk= length(p_den);
for ii=length(p_wrk):-1:lx
   p_wrk(ii)= p_wrk(ii) - p_den(jk);
   jk= jk - 1;
end
p_wrk= real(p_wrk)

figure(5);
clf;
K2= polyval( p_wrk, jx*2*pi*fswp ) ./ polyval( p_den, jx*2*pi*fswp );
p1= semilogx( fswp, 10*log10( abs(K2) ), 'k' );
set( p1, 'LineWidth', 2 );
grid on
title( '|K|^2 From Polynomials','FontName', 'Arial', 'FontSize', 14 );
%
%   Factor K^2
%
K2_num_roots= roots( p_wrk );
K2_den_roots= roots( p_den );

figure(6);
clf;
plot( real(K2_num_roots), imag(K2_num_roots), 'ro' );
title( 'Roots of |K|^2 Numerator' );
grid on
%
%   Retain only left-plane roots
%
K_num= 1;
mm=1;
for ii=1:length(K2_num_roots)
   if( real(K2_num_roots(ii)) <= 0.001 )
      K_num= conv( [1 -K2_num_roots(ii)], K_num );
      K_num_lhp_roots(mm)= K2_num_roots(ii);
      mm= mm+1;
   end
end
K_den= H_num
K_num= K_num

figure(7);
clf;
K= (1/gam)*polyval( K_num, jx*2*pi*fswp ) ./ polyval( K_den, jx*2*pi*fswp );
p1= semilogx( fswp, 10*log10( abs(K).^2 ), 'b-' );
set( p1, 'LineWidth', 2 );
grid on
title( 'Final K(\omega)','FontName', 'Arial', 'FontSize', 14 );
h= gca;
set( h, 'LineWidth', 2 );
xlabel( 'Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'dB', 'FontName', 'Arial', 'FontSize', 12 );
axis( [0.0001, max(fswp), -60, 80] );
```

```
disp( 'Characteristic Function Numerator:' );
disp( num2str( real(K_num), '%5.4e  ' ) );
disp( 'Denominator:' );
disp( num2str( real(K_den), '%5.4e  ' ) );

K_num_lhp_roots
K2_den_roots
%==========================================================================
%
%   Iteratively design filter
%
%   Focus on 5th order filter here
%
R1= 1;      % Source impedance
R2= 0.5;      % Load impedance

C1= 0.5;
C2= 0.25;
C3= 3.1;
C4= 0.25;
C5= 3.2;
wo1= 1.7013;
wo2= 1.0515;
L1= 1/(C2*wo1*wo1);
L2= 1/(C4*wo2*wo2);

freqs= 0.01*10.^(3*(0:75)/75);
Nfreqs=length(freqs);

abcd1= @(s) [ 1 0; s*C1 1 ];
abcd2= @(s) [ 1 1/(s*C2+1/(s*L1)); 0 1];
abcd3= @(s) [ 1 0; s*C3 1 ];
abcd4= @(s) [ 1 1/(s*C4+1/(s*L2)); 0 1];
abcd5= @(s) [ 1 0; s*C5 1 ];

err= 0;
figure(100);
clf;
PD= zeros(Nfreqs,5);    % Partial derivatives
clear g1;
clear g2;
for iter= 1:60
   for ff=1:Nfreqs
      g1(ff)= gam*polyval( H_num, jx*2*pi*freqs(ff) ) ./ polyval( H_den, jx*2*pi*freqs(ff) ) * R2/(R1+R2);
      g1(ff)= 10*log10( abs(g1(ff))^2 );

      ss= jx*2*pi*freqs(ff);

      abcd= abcd1(ss) * abcd2(ss) * abcd3(ss) * ...
           abcd4(ss) * abcd5(ss);
      g2(ff)= 1 /(abcd(1,1) + abcd(1,2)/R2 + R1*abcd(2,1) +(R1/R2)*abcd(2,2) );
      g2(ff)= 10*log10( abs(g2(ff))^2 );

      dg(ff)= g1(ff) - g2(ff);
      if( abs(dg(ff)) > 10 )
         dg(ff)= 10*sign(dg(ff));
      end

      %
      %   Get partial derivatives for this frequency
      %   and for each element value
```

```
        %
        %   Cap C1
        %
        ss= jx*2*pi*freqs(ff);

        abcd= abcd1(ss)*abcd2(ss)*abcd3(ss)*abcd4(ss)*abcd5(ss);
        apc= (-20/log(10)) / (abcd(1,1) + abcd(1,2)/R2 + R1*abcd(2,1) +(R1/R2)*abcd(2,2) );

        M1= [ 1 0; 0 1];
        M2= abcd2(ss)*abcd3(ss)*abcd4(ss)*abcd5(ss);
          PD(ff,1)= ( R1*M1(2,2)*M2(1,1) + (R1/R2)*M1(2,2)*M2(1,2) + M1(1,2)*M2(1,1) + (1/R2)*M1(1,2)*M2(1,2)
)*ss*apc;

        %
        %   First LC trap
        %
        M1= abcd1(ss);
        M2= abcd3(ss)*abcd4(ss)*abcd5(ss);
          PD(ff,2)= (R1*M1(2,1)*M2(2,1) + (R1/R2)*M1(2,1)*M2(2,2) + M1(1,1)*M2(2,1) + (1/R2)*M1(1,1)*M2(2,2)
)*ss/( 1 - abs(ss/wo1)^2 )*apc;

        %
        %   Cap C3
        %
        M1= abcd1(ss)*abcd2(ss);
        M2= abcd4(ss)*abcd5(ss);
          PD(ff,3)= ( R1*M1(2,2)*M2(1,1) + (R1/R2)*M1(2,2)*M2(1,2) + M1(1,2)*M2(1,1) + (1/R2)*M1(1,2)*M2(1,2)
)*ss*apc;

        %
        %   Second LC trap
        %
        M1= abcd1(ss)*abcd2(ss)*abcd3(ss);
        M2= abcd5(ss);
          PD(ff,4)= (R1*M1(2,1)*M2(2,1) + (R1/R2)*M1(2,1)*M2(2,2) + M1(1,1)*M2(2,1) + (1/R2)*M1(1,1)*M2(2,2)
)*ss/( 1 - abs(ss/wo2)^2 )*apc;

        %
        %   Cap C5
        %
        M1= abcd1(ss)*abcd2(ss)*abcd3(ss)*abcd4(ss);
        M2= [ 1 0; 0 1 ];
          PD(ff,5)= ( R1*M1(2,2)*M2(1,1) + (R1/R2)*M1(2,2)*M2(1,2) + M1(1,2)*M2(1,1) + (1/R2)*M1(1,2)*M2(1,2)
)*ss*apc;
    end
    %
    %   Update element values
    %
    de= lscov(real((PD)),dg');

    gamma= 0.25;

    C1= (C1 + gamma*de(1));
    L1= (L1 + gamma*de(2));
    C3= (C3 + gamma*de(3));
    L2= (L2 + gamma*de(4));
    C5= (C5 + gamma*de(5));

    C2= abs(1/(wo1^2*L1));
    C4= abs(1/(wo2^2*L2));

    abcd1= @(s) [ 1 0; s*C1 1 ];
```

```
    abcd2= @(s) [ 1 1/(s*C2+1/(s*L1)); 0 1];
    abcd3= @(s) [ 1 0; s*C3 1 ];
    abcd4= @(s) [ 1 1/(s*C4+1/(s*L2)); 0 1];
    abcd5= @(s) [ 1 0; s*C5 1 ];

    semilogx( freqs, g1, 'r' );
    hold on
    semilogx( freqs, g2, 'k--' );
    hold on
end

figure(200);
clf;
for ii=1:Npts
    ss= jx*2*pi*fswp(ii);

    abcd= abcd1(ss) * abcd2(ss) * abcd3(ss) * ...
          abcd4(ss) * abcd5(ss);
    gn(ii)= 10*log10( abs(1/(abcd(1,1) + abcd(1,2)/R2 + R1*abcd(2,1) +(R1/R2)*abcd(2,2) ))^2 );


    g1(ii)= gam*polyval( H_num, ss ) ./ polyval( H_den, ss ) *(R2/(R1+R2));
      g1(ii)= 10*log10( abs(g1(ii))^2 );
end
axes( 'FontName', 'Arial', 'FontSize', 12 );
p1= semilogx( fswp, gn, 'r' );
set( p1, 'LineWidth', 2 );
hold on
p1= semilogx( fswp, g1, 'k--' );
set( p1, 'LineWidth', 2 );
h= gca;
set( h, 'LineWidth', 2 );
grid on
xlabel( 'Normalized Frequency, Hz', 'FontName', 'Arial', 'FontSize', 12 );
ylabel( 'Voltage Gain, dB', 'FontName', 'Arial', 'FontSize', 12 );
title( 'Original Versus Modified Filter Gain', 'FontName', 'Arial', 'FontSize', 12 );
legend( 'Iterative Design Result', 'Ideal from Poles & Zeros' );

C1
C2
C3
C4
C5
L1
L2
```