Direct Approximation of Complex IIR Filters

Ken Martin
martin@granitesemi.comu

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, M5S 3G4, Canada

Granite SemiCom Inc.

Abstract: Complex filters are described by transfer functions that have complex coefficients; in addition, the filters may have complex input and/or output signals, as well. The transfer functions are rational functions of the Laplace variable, s, for continuous-time filters, or of the discrete-time variable $z = e^{j\omega t}$, for sampled filters. Complex filters are not restricted to have conjugate-symmetric transfer functions, in the frequency-domain, as is the case for real filters. This gives them more flexibility when used in communication systems having complex signals, such as the IF signals of wireless communication systems. This paper describes a set of Matlab[1] procedures that can be used in solving the approximation problem for deriving complex transfer functions directly without the requirement of first designing a real-transfer-function prototype-filter.

1. Introduction

A *complex* signal consists of a pair of *real* signals at an instant in time. If one denotes the complex signal, $X(t)$, as $X(t) = x_r(t) + jx_q(t)$, where $j = \sqrt{-1}$, then a Hilbert Space can be defined using appropriate definitions for addition, multiplication, and an inner-product and norm. In an actual physical system, the signals are both real (but are called the real and imaginary parts) and are found in two distinct signal paths. The multiplier "j" is used to help define operations

1.

between different complex signals, and is to some degree *imaginary* (that is ficticious); i.e., it doesn't actually exist in real systems.

Complex signal processing is becoming more pervasive [3][2]. An example of an important complex signal-processing block is a complex filter. Complex filters use cross-coupling between the real and imaginary signal paths in order to realize asymmetrical filters (in the frequency domain) having transfer functions that do not have the conjugate symmetry of real filters. This implies that their transfer functions have complex coefficients. Complex filters can be realized using the basic operations of addition and multiplication, and the delay operator (z^{-1}) for discrete-time digital filters, or the integration operator (s^{-1}) for continuous-time analog filters.

Complex analog transfer functions can be derived by either frequency-translating real filters [1][12][24], or by employing a direct method for deriving transfer complex functions to match asymmetrical specifications using procedures similar to those used for real filters [1][2][26]. The advantage of direct-approximation methods, without first designing real prototype filters, is the condition of arithmetic symmetry is not imposed on the transfer functions. For example, the upper and lower transition regions could have very different frequency spans; the upper and lower stop-band attenuation need not be the same. Similar choices exist for obtaining suitable transfer functions for discrete-time and digital filters [12]-[14],[15]-[19].

This publication describes approximation routines for realizing complex IIR filters as first described in [1] and [2], and developed primarily by Martin Snelgrove. The main justifications for the current publication are: references [1] and [2] are not readily available[3], do not give many of the details of implementing their algorithms, and the computer programs developed based on them are no longer available. This publication describes the details of Matlab [5] realizations of the procedures of [1] and [2] which are being made publicly available[4]. Although the general

2. Please note, this paper was written in 2004 and has not been updated to reflect any work after that date (currently Q1 2016); we apologise to the many authors who have been responsible for signifcant advances since then.
3. They have been included in the doc directory that is part of this distribuition.
4. Under the GNU General Public License V3 (http://www.gnu.org/licenses).

approach is the same as that of [1] and [2], many of the details are different as the algorithm details used in [1] and [2] could only be speculatively determined.

The use of the Matlab programming language has many advantages: it represents an exact description of the algorithms that is both readily understandable, very concise, and widely available in many different operating systems. A large reason for the conciseness of the Matlab procedures is due to the use of a rich variety of currently available algorithms in the signal-processing and control tool-boxes of Matlab; examples include robust routines for the root-finding of polynomials and sophisticated plotting routines. Many of the routines described here-in are clarified by including selected portions of the Matlab code.

The current set of routines for directly realizing complex continuous-time IIR filters are restricted to a special class of filters having a single band-pass response with unequal upper and lower stopbands. Low-pass filters are handled as a special case of band-pass filters. The routines can be used for both equiripple and monotonic passbands. The passband ripple is an arbitrarily-specified design parameter. The stopbands are also equiripple with an arbitrary number of loss-pole (transmission zeros) at infinity[5]. Although this is a moderately restrictive set of transfer functions (for example, it can not handle multiple passbands[6]), it is adequate for many practical applications, helps illustrate the principals involved while minimizing the amount of code necessary, and hopefully can be the basis of future extentions

The current set of routines use the bilinear-transform for realizing discrete-time or digital filters. First, a continuous-time complex IIR transfer-function is designed to meet pre-distorted specifications, and then the continuous-time transfer-function is transformed to a discrete-time transfer function that exactly meets the specifications at the specified frequencies. This is adequate for most applications. It is speculated that discrete-time filters could be realized without first realizing a continuos-time prototype, but this extension is not obvious, and is left as a research topic for the future. The routines described here-in are not intended for the design of dis-

5. For the developmental routines that have fixed loss-poles, the stopband minima are only roughly equiripple.
6. The extention to the multiple-passband case is an interesting research topic.

crete-time FIR filters, for group-delay equalizers, or for linear-phase filters; these would be useful extentions.

2. Continuous-Time Complex Band-Pass Filter Approximation.

The filter approximation routines described here-in are restricted to a single passband, but with arbitrary upper and lower passband frequencies, with the only restriction being that the lower passband frequency is less than the upper passband frequency; both frequencies can be negative, both can be positive, or the lower passband frequency can be negative with the upper passband frequency positive. A low-pass filter can be approximated as a special case where the lower and upper passband frequencies are equal in magnitude, with the former being negative and the latter positive. The lower stopband frequency must be less than the lower passband frequency, and the upper stopband frequency must be greater than the upper passband frequency. If both upper and lower passband and stopband frequencies are equal in magnitude and of opposite sign, then the resulting transfer functions obtained degenerate into real transfer functions. In this degenerate case, the transfer functions obtained are identical to those obtained using classical approximation algorithms [10].

The complex filter approximation approach described here-in consists of the following steps.

1. Specify upper and lower passband frequencies, upper and lower stopband frequencies, and the passband ripple. In addition, specify the number of loss poles (transmission zeros) at infinity, and initial guesses for finite-frequency loss poles which correspond to $j\omega$ axis transmission zeros. The frequencies of the finite-frequency loss-poles are adjusted to obtain equiripple stopbands during the approximation procedure. Currently non-moveable finite-frequency loss-poles are supported under a developmental set of routines (included in the distribution).
2. Transform the frequencies of the loss-poles using the conformal mapping described by

$$(s')^2 = \frac{s - j\omega_2}{s - j\omega_1} \tag{1}$$

where ω_1 is the lower passband frequency and ω_2 is the upper passband frequency [1][2]. This approach of doing the design using a transformed variable is similar to that used previously for real filters [8], with addition of the complex transform described in [1].

3. Adapt the positions of the finite-frequency loss-poles to obtain equiripple stopband attenuation in the upper and lower stopbands; the upper stopband attenuation is not normally equal to the lower stopband attenuation. The actual stopband attenuation achieved is completely dependent on the number and type of loss-poles and the passband ripple specified. The modified-Remez algorithm [6]-[10], and especially [9], is used for this procedure. The details of this procedure, which is the essence of this publication, are described in the next section.
4. Obtain the passband reflection-zero frequencies by solving exact equations to obtain either an equiripple or monotonic passband.
5. Transform the reflection zeros and the loss-poles from the s' domain back to the s domain
6. Using Feldtkeller's Equation, find the transfer function poles. This involves a root-finding operation.

3. Computer Considerations and Matlab Coding

Since the previous generation of filter approximation routines were first coded, originally using Fortran, and later using C and C++, much has changed. Perhaps the most significant development, is that currently the computing time of a particular algorithm is seldom an important consideration, given the speed of even inexpensive home computers, and the fact that solving the filter approximation problem is an infrequent event, where a small amount of patience is to be expected. This allows one to considerably simplify many algorithms, especially adaptive ones. For example, in the procedures presented here-in, most of the adaptive procedures can be performed for a fixed number of iterations, where the number is chosen large enough to guarantee convergence in the great majority of cases, irrespective of the specifications. This eliminates the need to check for convergence; the total time for filter approximation design is still found to be reasonable in all practical cases tested[7].

Another significant change from the early days of coding filter approximation routines is the Matlab coding environment. This environment provides a wealth of existing routines that can be used which greatly minimizes the amount of original new code that must be written. It is a largely interpreted rather than compiled environment, but this limitation has not been found to be significant given the current speed of computers. Although much of the Matlab environment can handle the complex math involved in the complex filter approximation problem (this has only been the case recently), not all of it can handle complex transfer functions. For, example, many of the plotting routines currently assume real transfer functions and can not be used for complex transfer functions; also, some of the control model operations give warnings or errors when complex coefficients are used. Limitations such as these are being corrected on an ongoing basis. Much use is made in the current algorithms of the Matlab Control Toolbox, and in particular of the System Objects in this toolbox. Despite documentation to the contrary (at the time of the authoring of this publication), the System Objects can handle complex-coefficient transfer functions, although they often give warnings[8]. In addition, the built-in transfer function display routines do not display the zeros correctly for discrete-time transfer functions and users must access the transfer function zeros directly to see the correct final results (e.g. using syntax like `T.z{1}`). One of the major advantages of using Control Toolbox System Objects is that much of the complexity in combining transfer functions, often done using root finding, is hidden; complexities such as multiple roots are automatically looked after using robust algorithms that have been fine-tuned and extensively tested by Mathworks. This paradigm is significant compared to what was previously available when filter approximation routines were first coded. Another related effect of the Matlab environment, is that there is no easy method of making the coding hidden (currently System Objects can not be compiled); partially for this reason, the routines described here-in are being released into the public domain[9].

---

7. Although it was unnecessary, the author could not resist the temptation in some cases to put in convergence checks and stop when these were reached before the total number of iterations was reached.
8. The zpk model gives an error when added to ‘1’, but can be worked around using the tf model and then transforming the result back to a zpk model (see the code).
9. And, hopefully, to ease the difficulty of realizing extentions.

In the routines being distributed, no efforts have been made, to date, to improve numerical accuracies, other than the fundamental algorithms having originally been chosen for their excellent numerical robustness [8]. The routines generally seem accurate for stopband minima attenuations below around 100dB, but do seem to suffer numerical inaccuracies at either step 4 or 5 (above) for stopband attenuation greater than 100dB; we hope to address this in the future. The routines are often capable of handling fairly high-order filters given this constraint.

4. The s' Domain Formulation

The adaptation of the loss-poles (transfer-function transmission-zeros) is done in the s' domain using the conformal mapping given by (1) [1] and [2]. This mapping transforms the transition and stopband imaginary frequencies (in s) to the positive real axis in s'; infinity is mapped to unity, the upper stopband is mapped to between 0 and unity, and the lower-stopband is transformed to between unity and infinity. In addition, and perhaps more importantly, the passband is mapped to the complete imaginary axis. This latter mapping allows one to use simple all-pass properties to realize elliptic equiripple passbands exactly [10].

The filter approximation problem posed is a restricted one where the loss function ($H(s) = 1/(T(s))$, where $T(s)$ is the desired filter transfer function) has the form

$$H(s)H^*(-s) = 1 + \varepsilon^2 K(s)K^*(-s) \tag{2}$$

This formulation is called Feldtkeller's Equation [9]. Note that $H^*(-s)$ denotes conjugating all coefficients and then also substituting $-s$ for the Laplace variable s. We further constrain the problem for the equiripple passband case[10] to have the characteristic function, $K(s)$, expressible as

10. We will first consider the equiripple passband case only, and then discuss the simpler case of a monotonic passband.

$$K(s) = \frac{F(s)}{P(s)} = \frac{(s - jf_1)\ldots(s - jf_{nf})}{(s - jp_1)\ldots(s - jp_{np})s^{ni}} \tag{3}$$

The parameters $f_1 \ldots f_{nf}$ are all real and equal to the passband frequencies where the gain is at a maximum. These are historically called the reflection zeros, therefore, the polynomial $F(s)$ is called the reflection polynomial. The parameters $p_1 \ldots p_{np}$ are also all real and equal to the frequencies where the transmission gain is zero, and are therefore often called transmission zeros. These are also commonly called imaginary-axis loss-poles. The cases of non-imaginary-axis loss-poles is not dealt with in this formulation[11]. The reflection-zero frequencies are all constrained by

$$\omega_1 < f_i < \omega_2 \tag{4}$$

where ω_1 and ω_2 are the lower and upper passband frequencies, respectively. Further, the loss poles are constrained to be either less than ω_1 or greater than ω_2 (they are further constrained to not be in the transition bands). Finally, we shall see that the number of reflection zeros, nf, is given by

$$nf = np + ni \tag{5}$$

where ni is the number of loss-poles at infinity. In this formulation, it is not necessary that the order of the filter is even as is required for real band-pass filters. In addition, the routines can accommodate the case of $ni = 0$.

Applying the mapping defined by (1) to (3) transforms the characteristic equation in the s domain to the s' domain so that the characteristic equation is now given by

11. None imaginary loss-poles could be useful when it is desired to control the phase (or group delay) in addition to the magnitude response without having to add a separate phase equalizer; again a potential area of future research.

$$K(s') = \frac{(s'^2 + f'^2_1)\ldots(s'^2 + f'^2_{nf})}{(s' + p'_1)^2\ldots(s' + p'_{np})(s' - 1)^{ni}} \tag{6}$$

where

$$p'_i = \sqrt{\frac{p_i - \omega_2}{p_i - \omega_1}} \tag{7}$$

and

$$f'_i = \sqrt{\frac{\omega_2 - f_i}{f_i - \omega_1}} \tag{8}$$

Note that the order of the characteristic function has doubled in the s' domain as compared to the s domain.

The filter design procedure now proceeds according to the following procedure: The loss-poles p'_i are adapted to equalize the stopband minima using the modified-Remez algorithm following the formulation described in [9][10] for unequal stopbands. After each adaptation, the reflection zeros f'_i are updated to guarantee an equiripple passband. A few notes regarding the details of the proposed procedure, which may be different than those used in [1] and [2], are worth mentioning: During the application of the modified-Remez algorithm to the placement of the loss-poles, the loss minima of the function $K(s')K^*(-s')$ rather than $1 + K(s')K^*(-s')$ are equalized. During this process, the derivatives of the natural log of $K(s')K^*(-s')$ with respect to the individual loss-poles are used as they simplify calculations. In calculating these derivatives, the dependence of the reflection zeros on loss-poles is ignored. Due to the approximate orthogonality in the s' plane between the stopband and the passband, this does not introduce significant errors around convergence. At each step of applying the modified-Remez method, it is necessary to find the loss minima of $K(s')K^*(-s')$ in the stopband[12]. Rather than using a search method, these minima are

12. Note that minima between fixed poles are ignored.

found at each step by applying a second adaptive process to find s' where the derivatives of $K(s')K^*(-s')$ with respect to s' are zero. During this process, second-order derivatives are necessary; these are found approximately using an updated difference-equation approach rather than using exact analytic expressions. This approximation has not been found to introduce any errors, and considerably simplifies the algorithms. In addition to the minima where the derivatives are zero, minima are assumed at the stopband edges and also at $s' = 1$, which corresponds to $s = \infty$, for the special case of no loss-poles at infinity (i.e. $ni = 0$).

5. Pole-Placement Routine

The adaptive update used for modifying the loss-poles to get equiripple stopband loss is very similar to that of [10] with un-equal stopbands except for the differences described in the previous section, and the fact that the desired stopband loss is immaterial in the procedures described herein, once the filter order, type, and passband ripple have been specified. Defining

$$L(s') = ln(K(s')K^*(-s')) \tag{9}$$

then at each iteration, it is necessary to calculate the derivative of the minima $L(s')$ with respect to the individual loss-poles, p'_i. That is, we need $DL_{i,j}$, where

$$DL_{i,j} = \frac{d}{dp'_i} L(s') \Big|_{s' = \gamma_j} \tag{10}$$

and γ_j is the j'th minima. Given this definition, the loss-poles are updated by solving the following linear equation at each iteration

$$\begin{bmatrix} DL_{1,1} & \cdots & DL_{1,np} & -1 & 0 \\ & \cdots & & & \\ DL_{nu,1} & \cdots & DL_{nu,np} & -1 & 0 \\ DL_{nu+1,1} & \cdots & DL_{nu+1,np} & 0 & -1 \\ & \cdots & & & \\ DL_{nu+nl,1} & \cdots & DL_{nu+nl,np} & 0 & -1 \end{bmatrix} \begin{bmatrix} \Delta p_1 \\ \cdots \\ \Delta p_{np} \\ M_1 \\ M_2 \end{bmatrix} = - \begin{bmatrix} L(\gamma_1) \\ \cdots \\ L(\gamma_{nu}) \\ L(\gamma_{nu+1}) \\ \cdots \\ L(\gamma_{nu+nl}) \end{bmatrix} \tag{11}$$

where np is the number of moveable poles, nu is the number of minima in upper stopband, and nl is the number of minima in lower stopband. Normally $nu + nl = np + 2$ unless there are fixed loss-poles other than those corresponding to infinity (in s). The programming of this iteration step is considerably simplified using the built-in linear-equation-solving capabilities of Matlab. To illustrate how easily this can be done in Matlab, the applicable code is shown in Fig. 1. for

```
% Set up the last two columns of the sensitivity matrix
s2 = zeros(nu + nl,2);
for i=1:nu
   s2(i,:) = [-1 0];
end
for i=1:nl
   s2(i+nu,:) = [0 -1];
end

% Set the initial values for the X vector
X = [p.'; 1; 1]; % p is currently the initial values for the moveable loss-poles

% Adapt the loss-pole positions to equalize the stopband loss minima
for i = 1:30 % repeat enough times to guarantee success (normally anything > 15)
      S = [dlogKK_dp(Kz_,zmin) s2]; % Calculate the sensitivity matrix at the current miima
      Y = -log_rsps(Kz_*Kz_',zmin); % Calculate the loss at the current mimima
      X = S\(Y); % Calculate the changes in the pole frequencies using built-in matrix inversion
      p = p + X(1:np).'; % Calculate the new pole positions
      Kz_ = make_Kz(p,ni,type); % Calculate the new characteristic equation in s'
      zmin = find_minima(Kz_); % Find the new stopband loss minima
      zmin = [z4 zmin z3]; % Augument with the loss at the stopband edges
      if ni == 0 % Augument with the loss at s=infinity (s' = 1) when ni = 0
            zmin = sort([1 zmin]);
      end
end
```

Fig. 1. The Matlab code that realizes the adaptive pole-placement using the modified-Remez algorithm.

the case of no fixed loss-poles.

6. Calculating Log Response Vector and Sensitivity Matrix

At each iteration of the pole-placement routine, it is necessary to calculate the log response at the current minima. This occurs at the line, `Y = -log_rsps(Kz_*Kz_',zmin)`, in Fig. 1. The variable `Kz_` is a Control-Toolbox System Object equal to the characteristic function in the s' domain. The function `Kz_'` (note the $'$ operator) uses built-in Matlab routines to conjugate all internal coefficients and substitute $-s'$ for s' (using operator over-loading). Thus,

$$Kz\_' = K^*(-s') \tag{12}$$

Note the very compact notation. Also, when calculating the minima, the frequencies are passed in as a vector, `zmin`; thus, all responses at the minima are calculated with a single function call. In addition, the syntax `Kz_*Kz_'` denotes combining the equivalent of two cascaded system objects, $K(s')$ and $K^*(-s')$; this implicitly finds the roots of the resulting object using routines that are robust and have special cases for repeated roots. Internal to the routine, the log10 response of $K(s')K^*(-s')$ is calculated using the routine shown in Fig. 2.

```
function h = log_rsps(sys,s)
% This is a program for calculating the log10 response of a zpk system.

[z,p,k] = zpkdata(sys); % Find the zeros and poles
ls = length(s); % Find the number of frequency points
h = zeros(ls,1); % More convenient for loop below

for i = 1:ls % Iterate for each frequency
  zs = s(i) - z{1}; % Calculate the vector of zeros; z is coefficients possibly complex
  ps = s(i) - p{1};% Calculate the vector of poles
  if any(ps==0)
      h(i) = Inf; % Check for division by zero
  elseif ~any(zs==0)
      % RE: Go to log to prevent overflow in products
      h(i) = log2(k) + sum(log2(zs)) - sum(log2(ps)); % Use builtin functions to simplify
  end
end
h = log10(abs(pow2(h))); % convert log2 to log10 % Convert to log10
```

Fig. 2. The Matlab code that calculates the log10 response of $K(s')K^*(-s')$ *at the minima.*

The calculation of the sensitivity of $K(s')K^*(-s')$ with respect to the individual moveable poles, at the minima frequency, is one of the most essential operations in the pole placement rou-

tine. This operation is considerably simplified by calculating the sensitivities of $ln\{K(s')K^*(-s')\}$ [13] and by ignoring the fact that the reflection zero frequencies are implicitly functions of the pole frequencies in order to maintain the proper response in the passband. This simplifying approximation has not been found to degrade approximation accuracy. From (6), we have

$$K(s')K^*(-s') = \frac{(s'^2 + f'^2_1)^2 \ldots (s'^2 + f'^2_{nf})^2}{(p'^2_1 - s'^2)^2 \ldots (p'^2_{np} - s'^2)^2} \tag{13}$$

and

$$ln\{K(s')K^*(-s')\} = 2\{ln(s'^2 + f'^2_1) + \ldots ln(s'^2 + f'^2_{nf}) - ln(p'^2_1 - s'^2) - \ldots - ln(p'^2_{np} - s'^2)\} \tag{14}$$

Note that the special case of poles at infinity in s translate to poles at ± 1 in s' and these do not require special consideration different than for the moveable poles in (13) and (14). Therefore, with the assumption f'_j is independent of p'_i, for all i,j, we have

$$\frac{\partial}{\partial p'_i} ln\{K(s')K^*(-s')\} = \frac{-4p'_i}{(p'^2_i - s'^2)} \tag{15}$$

This equation is the basis of the Matlab routine shown in Fig. 3. for calculating the sensitivity matrix.

7. Calculating the Characteristic-Function Loss-Squared Minima

In addition to calculating the loss-sensitivity matrix, another essential operation for the current approximation methodology is the calculation of the minima frequencies of $ln\{K(s')K^*(-s')\}$. Rather than using a search approach as suggested in [10], the current approach is to adaptively determine the frequencies, between the loss-poles, where the derivatives of $ln\{K(s')K^*(-s')\}$

13. By not converting to log base-10, the pole-placement adaptation is slower but less prone to divergence.

```
function h = dlogKK_dp(sys,s)
% This is a routine for calculating the derivative of the log of the
% magnitude squared response of system Kz_ with respect to the loss poles.
% sys should be an LTI object.
% s is the frequencies of the minima
% The final matrix h returned has one row for each minima frequency and one
% column for each moveable loss-pole

ls = length(s); % Number of frequency minima
[z,p,k] = zpkdata(sys); % Get zeros and poles

% delete any imagainary residues and then convert to positive
p_ = abs(real(p{1}));
p_ = sort(p_); % Sort
np = length(p_);

% delete every second pole as all poles are repeated at least twice
p_ = p_(1:2:np-1);
np_ = np/2;

% get rid of loss poles with multiplicity greater than 1
indx = [abs(p_(1:np_-1) - p_(2:np_)) > 10*eps; 1];
p_ = p_(logical(indx));

% delete poles at 1, as these are unmoveable and correspond to poles at
% inf.
% This could be extended to handle other fixed poles

p_ = p_(p_~=1);
p_ = p_.'; % Convert to row
np_ = length(p_);

pz2 = p_.^2; % Calculate squares of pole frequencies
s = s.'; % Convert to column
h = zeros(ls,np_); % Matrix size

for i = 1:ls % For each frequency
  p2 = pz2 - s(i)^2; % Calculate row vector
  if any(p2==0) % Don't divide by zero
       h(i,p2(p2==0)) = Inf;
  else
       h(i,:) = -4*(p_./p2); % Calcuate row of derivatives of ln(Kz_*Kz_')
  end
end
```

Fig. 3. The Matlab routine that calculates the loss-pole sensitivities of $ln\{K(s')K^*(-s')\}$ *at the minima.*

with respect to s' are zero. This is easy to do since the first-order sensitivities are easy to calculate. The second-order sensitivities are also required in this procedure; however their accuracy is not important. These could be calculated exactly; however, using a difference equation approach to approximate the second-order sensitivities is simpler, and does not degrade over-all approximation accuracy. If we define

$$\frac{\partial}{\partial s'} ln\{K(s')K^*(-s')\} = dLK \tag{16}$$

then the approach taken, based on the Levenberg-Marquardt optimization procedure [11], is to interactively update z_{min}, the minima frequency according to

$$\Delta z_{min} = \frac{-dLK}{\frac{\partial}{\partial s'} dLK\Big|_{s' = z_{min}} + D_{min}} \tag{17}$$

The term D_{min} is included to prevent division by zero and is currently taken equal to 1×10^{-16}. The first order derivatives, dLK, are calculated exactly using the formula

$$\frac{\partial}{\partial s'} ln\{K(s')K^*(-s')\} = 4\left\{\frac{s'}{s'^2 + f'^2_1} + \ldots + \frac{s'}{s'^2 + f'^2_{nf}} + \frac{s'}{p'^2_1 - s'^2} + \ldots + \frac{s'}{p'^2_{np} - s'^2}\right\} \tag{18}$$

The second-order derivatives are updated at each iteration using the formula

$$\frac{\partial}{\partial s'} dLK\Big|_{s' = z_{min}} \cong \frac{dLK(z_{min+1}) - dLK(z_{min})}{z_{min+1} - z_{min}} \tag{19}$$

at each minima frequency of z_{min}, where z_{min+1} is the updated minima frequency. This procedure for finding the minima was found to converge with excellent accuracy in typically 4-8 iterations. The Matlab routine for calculating the characteristic-function squared loss minima is given in Fig. 4.

8. Calculating the Reflections Zeros and the Final Transfer Function

The calculation of the reflection zeros, of the characteristic function, is exact; the procedure is the same as described in [10] for real filters. Assume

$$K(s') = \frac{F(s')}{P(s')} \tag{20}$$

```
function zmin = find_minima(sys)
% This is a simple program for finding the minima of Kz_*Kz_'  in the z plane
% Note that z is equiv. to s', not the z variable used in discrete-time filters
% The minima of Kz_Kz_' are all assumed to be between the loss poles.
% Loss poles at 0 in z correspond to jw2 in s
% Loss poles at inf in z correspond to jw1 in s
% Loss poles at 1 in z correspond to inf in s
% each loss pole is repeated twice (at least) and normally negative

[z,p,k] = zpkdata(sys); % Get the zeros and poles

p_ = abs(real(p{1})); % Delete any imagainary residue and then convert to positive
p_ = sort(p_); % Sort
np = length(p_); % Find length

p_ = p_(1:2:np-1); % Delete every second pole as all poles are repeated
np_ = np/2; % Correct order

% Get rid of loss poles with multiplicity greater than 1
indx = [abs(p_(1:np_-1) - p_(2:np_)) > 10*eps; 1];
p_ = p_(logical(indx));
np_ = length(p_);

winit = sqrt(p_(1:np_-1).*p_(2:np_)); % Calculate initial guesses of minima as geometric averages

for k = 1:np_-1 % Iterate for each minima
   w1 = winit(k);
   w2 = w1 + 1e-5; % Used to calculate estimate of second derivative
   FPrime = 1; % Initialize second derivative
      for i = 1:25 % Iterate enough times to guarantee convergence, 15 is adequate
       F1 = dlogKK_dz(sys,w1); % Find vector of first derivatives at w1
       F2 = dlogKK_dz(sys,w2); % Find vector of first derivatives at w2
       if (w2 - w1)~=0 % Do not divide by zero
          FPrime = (F2 - F1)./(w2 - w1); % Calculate second derivative using difference equat.
       end
       w1 = w2;
       w2 = w2 - F2./(FPrime + 1e-16); % Update minima; do not divide by zero
% This check for convergence and early termination is not necessary.
        % I couldn't keep myself from including. Typical termination is for i = 4 to 8
if max(abs(w1 - w2)) < 2*eps
          % disp('Minima Iteration Terminated')
          % i
          break
       end
      end
   zmin(k) = w2;
end

if np_<=1 % Look after special case of no moveable loss-poles
   zmin = [];
end
```

Fig. 4. The Matlab routine that calculates the minima frequencies between loss-poles of $ln\{K(s')K^*(-s')\}$.

In order to obtain an equiripple passband, one simply takes

$$F(s') = \mathrm{Re}\{P(s')\} = \frac{P(s') + P^*(-s')}{2} \tag{21}$$

This is implemented in Matlab by first transforming $P(s')$ to transfer function form (as opposed to pole zero form) so the cancellation of the odd terms implied by (21) is exact. This step is a potential candidate as the major culprit for numerical inaccuracies when stopband minima are greater than 100dB.

For the case of monotonic passbands, there are $nf = np + ni$ *equal* reflection zeros (np is the number of finite $j\omega$ loss-poles, and ni is the number of loss poles at infinity in the s domain). The magnitude of the frequency of the reflection zeros is equal to the geometric average of all the loss-poles (including those corresponding to infinity in the s domain) in the s' domain. Taking into account the fact that there is a repeated loss-pole in s' corresponding to each loss-pole in s, we have

$$f'_0 = \left(\prod_{i=0}^{2nf} p'_i \right)^{\frac{1}{2nf}} \tag{22}$$

After the reflection zeros are found using either (21) or (22), then the reflection zeros and loss-poles are transformed back to the s domain, and the final loss function, $H(s)$, is found by solving Feldtkeller's Equation (that is equation (2)) and choosing the left-half-plane loss-zeros (which correspond to the left-half-plane poles of the transfer function). The Matlab code for the equiripple case is shown in Fig. 5. The routine for the monotonic passband case is similar, except that (22) is implemented rather than (21).

9. Discrete-Time Filters

Discrete-time and digital complex filter approximation is currently supported using the bilinear [20] transform. The filter specifications of the passband and stopband frequencies (ω_i) are first pre-distorted using

```
function H = elliptic_bp(wp,w1,w2,ni,e_)
% ni: number of loss poles at infinity
% wp: finite loss poles
% w1: lower passband edge
% w2: upper passband edge
% e_: passband ripple = sqrt(1 + e_^2)

np=length(wp); % number of finite loss poles (including zero)

% transform loss poles to s-plane and calculate denominator of H(s)
% which is the numerator of transfer function T(s)
pz = zpk([],[],1);
for i = 1:np % Find transformed finite jw axis poles
   wpp(i) = trnsfrm_wp(wp(i),w1,w2); % wpp(i) is equal to pi^2
   pp(i) = zpk(tf([1 2*sqrt(wpp(i)) wpp(i)],1));
   pz = pz* pp(i);
end
for i = 1:ni % Find transformed poles at infinity
   pi(i) = zpk(tf([1 2 1],1));
   pz = pz*pi(i);
end

% fz is even part of pz
fz = (tf(pz) + tf(pz)')/2; % Convert to transfer function so cancellation is exact
fz = zpk(fz); % Convert back to zero, pole form

% transform fz back to s plane
z = fz.z{1,1};
nz = length(z)/2;
for i = 1:nz
   z2(i) = z(2*i - 1)*z(2*i);
   wf(i) = trnsfrm_yf(z2(i),w1,w2);
end
wf = wf.';

% find H by solving Feldtkeller's equation in the s domain
K = zpk(wf, wp, 1.0);
KK_ = K*K';
k = 1.0/abs(freqresp(K,w1));
HH_ = 1 + e_^2*k^2*KK_;

% choose LHP poles (which are zeros of HH_)
hh_z = HH_.z{1,1};
n = ni + np;
k = 1;
for i = 1:2*n
   if real(hh_z(i)) < 0
      we(k) = hh_z(i);
      k = k+1;
   end
end

% finally form transfer function and normalize gain
H = zpk(we,wp,1);
g = sqrt(1.0 + e_^2)/abs(freqresp(H,w1));
H = g*H;
```

Fig. 5. The Matlab routine that given a set of loss-poles, calculates the reflection zeros in the s' *domain, transforms the characteristic equation back to the* s *domain, and then solves Feldtkeller's equation for the final loss function* $H(s) = 1/T(s)$.

$$\omega_{di} = 2\tan\left(\frac{\omega_i}{2}\right) \tag{23}$$

where $-\pi < \omega_i < \pi$ with π equal to half the sampling frequency in radians. After pre-distortion, a continuous-time complex IIR filter is designed; this filter is then transformed to the discrete-time domain using the bilinear substitution

$$s \rightarrow \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{24}$$

This substitution is realized using the Matlab bilinear() function operating on the poles and zeros; the control-system toolbox function d2c(sys,Ts,'tustin') was found to not work correctly for complex filters at the time this publication was written[14]. For discrete-time filters, it is assumed the sampling frequency is $1\,\mathrm{Hz}$; for other desired sampling frequencies, the filter specification frequencies should first be scaled by dividing them by the sampling frequency.

10. Examples

As an example, consider the design of a continuous-time equiripple passband filter having a passband between 1 rad. and 2 rad., a lower-stop at frequencies less than 0.05 rad., and an upper stopband at frequencies greater than 2.5 rad. Note that the upper transition region is larger than the lower transition region. This particular filter is an example of a positive-pass-filter (PPF) which has the passband completely at positive frequencies only. In this example, two loss-poles at infinity are specified, two moveable loss-poles in the lower stopband are specified, and three loss-poles in the upper stopband are specified. The appropriate Matlab code for designing this filter, and plotting the response is shown in Fig. 6. Also, shown in Fig. 6 is the code to design an identical filter except with a monotonic passband rather an equiripple passband. Plots of the stopband

14. That is in 2004; we have not yet checked if this has changed in 2016.

```
p = [-2 -15 0 3 5]; % initial guess at finite loss poles; note pole at zero
ni=2; % number of loss poles at infinity
w(1) = 1; % lower passband edge
w(2) = 2; % upper passband edge
w(3) = 0.05; % lower stopband edge
w(4) = 2.5; % upper stopband edge
Ap = 0.1; % the passband ripple in dB

% A positive-pass continuous-time filter with an equiripple passband
H = design_ctm_filt(p,ni,w,Ap,'elliptic');
hndl(1) = figure('Position',[100 200 500 600]);
plot_crsps(H,w,'r');

% A positive-pass continuous-time filter with a monotonic passband
H = design_ctm_filt(p,ni,w,Ap,'monotonic');
hndl(2) = figure('Position',[200 200 500 600]);
plot_crsps(H,w,'b');
```

Fig. 6. The Matlab code for designing a continuous-time PPF with an equiripple passband.

and passband magnitude transfer functions are shown in Fig. 7. and Fig. 8. Note how much larger

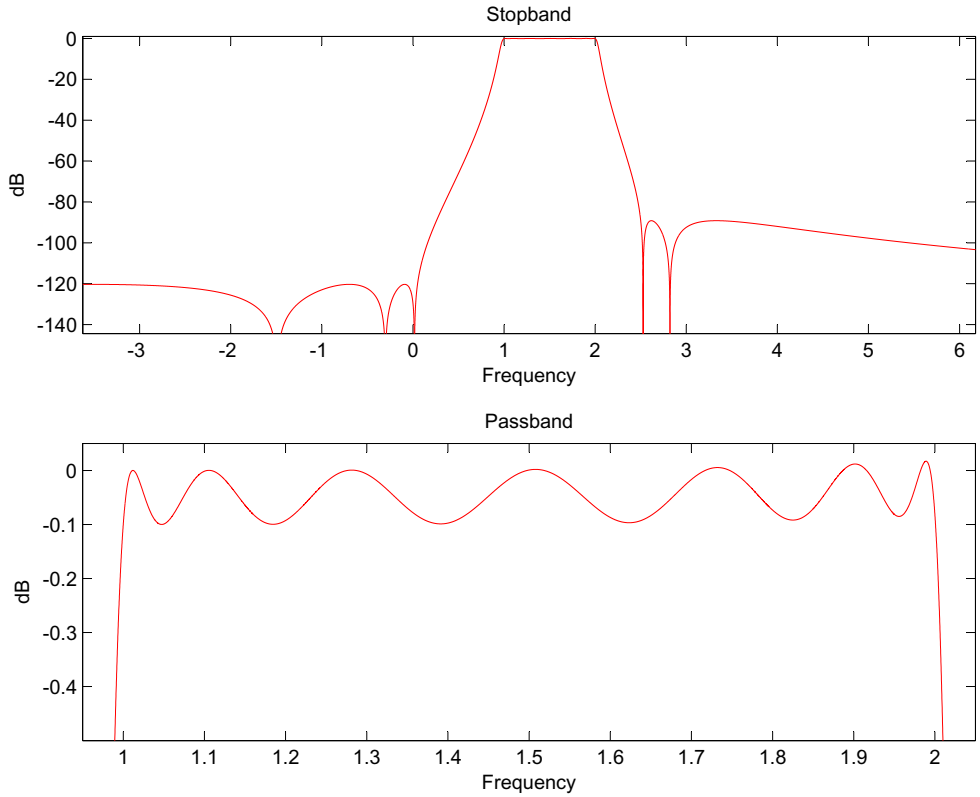

Fig. 7. The stopband and passband responses of a continuous-time PPF having an equiripple passband.

the stopband attenuation is in the equiripple passband case.

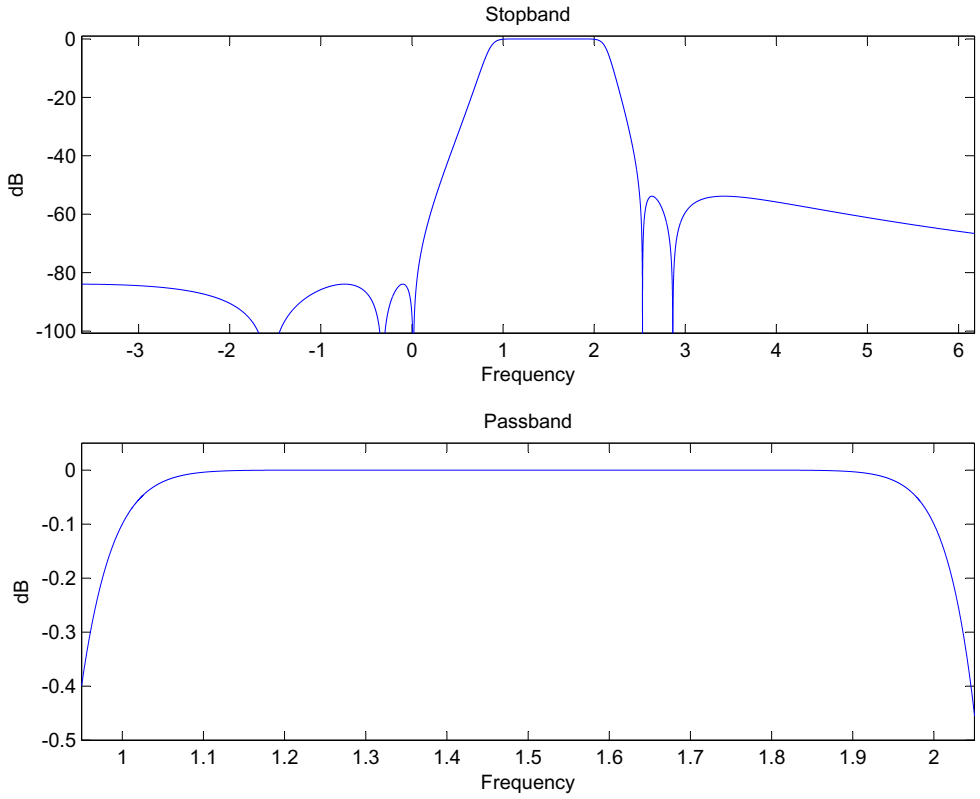

Fig. 8. The stopband and passband responses of a continuous-time PPF having a monotonic passband.

The next two examples are similar, but this time discrete-time filters are designed, and there are two moveable loss-poles in both the lower and the upper stopbands, but again the response is asymmetric. The Matlab code is shown in Fig. 9. The responses for the equiripple and monotonic

```
pd = [ -.25 -.2   .25 .4]; % initial guess at finite loss poles
ni=2; % number of loss poles at infinity
wd(1) = 0.05; % lower passband edge
wd(2) = 0.15; % upper passband edge
wd(3) = 0.02; % lower stopband edge
wd(4) = 0.20; % upper stopband edge

Ap = 0.1; % the passband ripple in dB

H = design_dtm_filt(pd,ni,wd,Ap,'elliptic');
hndl(3) = figure('Position',[300 200 500 600]);
plot_drsps(H,wd,'r');

H = design_dtm_filt(pd,ni,wd,Ap,'monotonic');
hndl(4) = figure('Position',[400 200 500 600]);
plot_drsps(H,wd,'b');
```

Fig. 9. The Matlab code for designing a continuous-time PPF with an equiripple passband.

cases are shown in Fig. 10. and Fig. 11., respectively. Once again, the significantly superior stop-

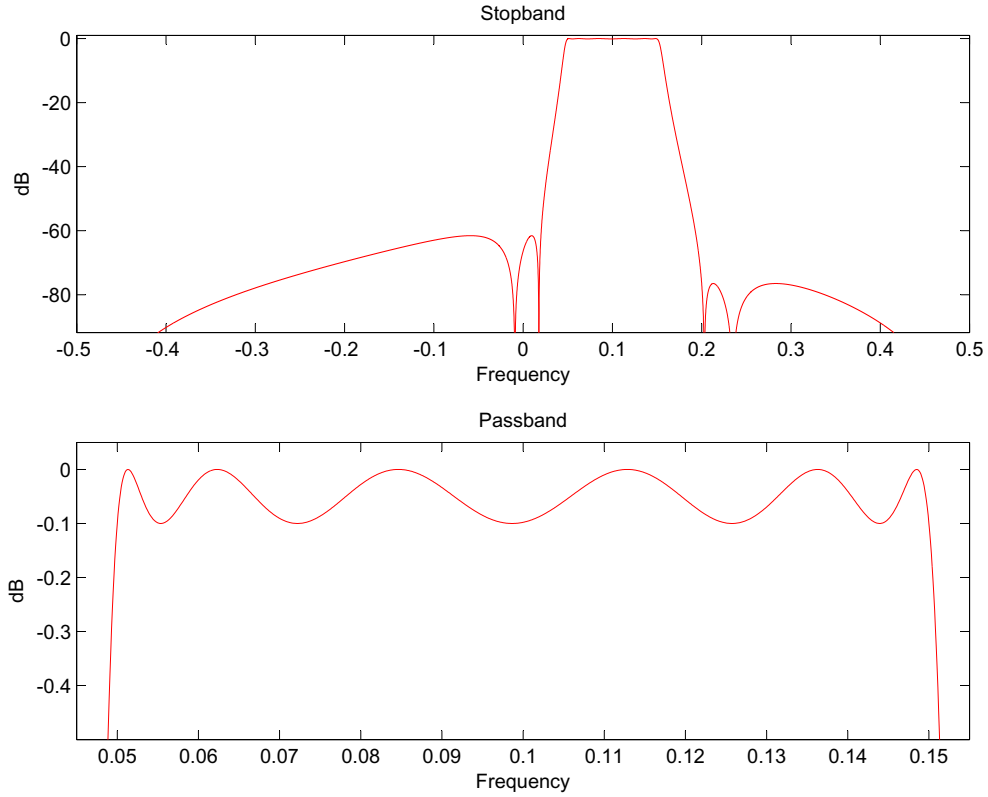

Fig. 10. The stopband and passband responses of a discrete-time PPF having an equiripple passband.

band attenuation of the equiripple case is apparent.

11. Fixed-Frequency Loss-Poles

Extending the filter-approximation routines to handle fixed-frequency loss-poles involves considerable complication; this extension is still under development, and, thus, will only be discussed briefly. An example of where one may like to have fixed-frequency loss-poles is where a loss-pole exactly at d.c. is desired to eliminate d.c. offsets that may be present in the input signal of some systems[15]. The problem with filter approximation with apriori specified loss-pole frequencies is that when setting up the pole-adaptation equations, similar to (11), there are more minima than there are moveable loss-poles, and the equations are over-determined. This means that an exact equal-minima stopband is not possible. The developmental approach is to solve (11)

15. Having loss poles at d.c. is especially useful in wireless systems.

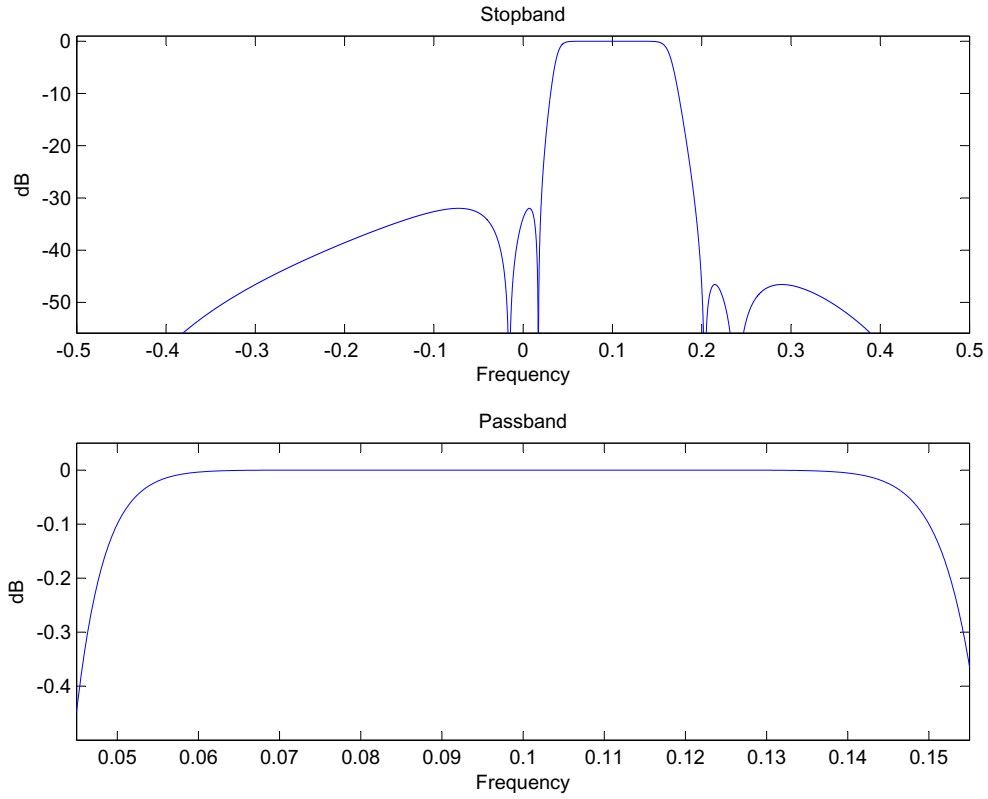

Fig. 11. The stopband and passband responses of a discrete-time PPF having a monotonic passband.

using pseudo-inverses which may only give roughly-equal stopband minima, but still gives exactly equiripple passbands. A practical proposed design approach is as follows:

1. Find a filter approximation that does not have moveable poles which has the desired stopband loss with some margin.
2. Change the specification where the initial poles are close to their final frequencies found in the first-step; but, where the moveable loss-pole found that is closest to the frequency of the fixed-frequency poles is removed, and a fixed-frequency pole is specified. The filter found at this step will have stopband minima roughly equal to the minima found in the first step, the transfer function may now be adequate. Very-often in real applications, as long as the stopband minima are all small-enough, having exactly equal stopband minima is unnecessary.
3. If more accurately equal stopband minima are desired, then the stopband frequency specification at the stopband edge closest to the fixed frequency pole can be adjusted by trial-and-error to give very-closely-equal stopband minima; it was found that 10 minutes of effort during this

step would always result in stopband minima that were equal to better than 0.01 dB, for the cases tried.

Although this extension is developmental, it does seem to work well for many practical applications and therefore is being added to the distribution. The details of the extensions necessary for handling the fixed-frequencies can be found by examining the code, as Matlab code is very readable with a little practice. The matlab code for an example calling the approximation routines with a fixed-frequency loss-pole at d.c. is given Fig. 12. The resulting transfer function is shown in Fig.

```
% This is an example of a discrete-time filter with a fixed frequency loss-pole exactly at
% d.c. The lower-stopband-edge frequency was adjusted by trial-and-error until the stopband
% minima were very closely equal.
p = [ -.4  -0.05  0.018  .35 .45]; % initial guess at moveable finite loss poles
px = [0]; % fixed pole
ni=1; % number of loss poles at infinity
w(1) = 0.1; % lower passband edge
w(2) = 0.2; % upper passband edge
w(3) = 0.024715; % lower stopband edge
w(4) = 0.3; % upper stopband edge

Ap = 0.1; % the passband ripple in dB

fig = figure('Position',[100 200 500 800]); % This places and sizes plot figure
H = design_dtm_filt2(p,px,ni,w,Ap,'monotonic'); %This is a discrete-time design with a fixed loss-pole at dc.
plot_drsps(H,w,'r',[-0.5 0.5 -120 1]); % Plot the response (with specified axis scaling)
```

Fig. 12. The Matlab code for designing a discrete-time filter with a fixed-frequency pole at d.c.

13. In this example, the stopband minima are very-accurately made equal by making the lower-stopband frequency edge equal to 0.024715 Hz. This accuracy is not necessary in any real application.

12. Conclusion

A set of procedures, and corresponding Matlab routines, for designing complex continuos-time and discrete-time filters directly, without first having to design a real prototype filter, have been described. The procedures can design filters with either equiripple or monotonic passbands. The procedures can handle any number of mobile loss-poles, loss-poles at infinity, and (although

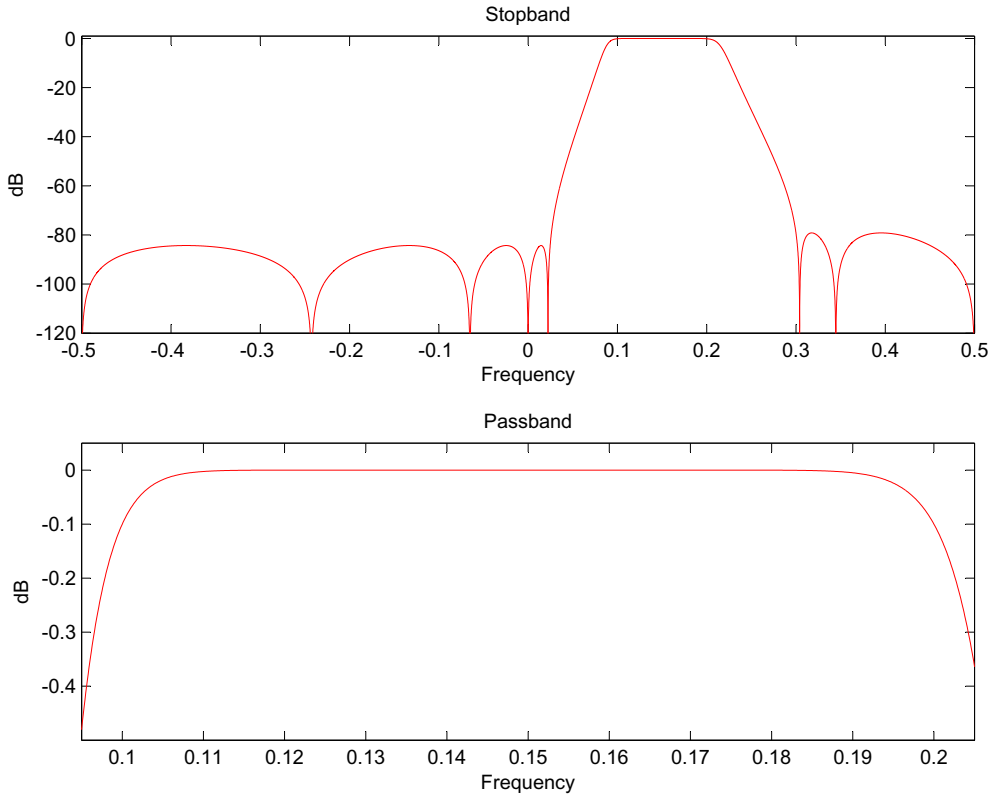

Fig. 13. The stopband and passband responses of a discrete-time PPF having a monotonic passband and a fixed-frequency pole at d.c.

still under development) fixed-frequency loss-poles. A number of example filter designs have also been given.

13. References

[1] W. M. Snelgrove, *Intermediate Function Synthesis*, Ph.D. Thesis, University of Toronto, 1982.

[2] W.M. Snelgrove, A.S. Sedra, G.R. Lang, and P.O. Brackett, "Complex Analog Filters," 1981, ftp://ftp.eecg.toronto.edu/pub/software/martin/snelgrove_cmplx.pdf.

[3] K. Martin, "Complex Signal Processing is Not - Complex," *Plenary Presentation at 2003 European Solid-State Circuits Conference*, Estoril Portugal, Sept. 2003.

[4] K.W. Martin, "Complex signal processing is not complex," IEEE Trans. Circuits and Systems I, vol. 51, pp. 1823–1836, Sept. 2004.

[5] "Matlab, The Language of Technical Computing - Using Matlab," Copyright 1984-2004, The MathWorks Inc.

[6] E. Remez, "Sur le calcul effectif des polynomes de Tchebicheff," Compt. Rend. Acad. Sci. (France), Vol. 199, pp. 337-340, July 1934.

[7] B.R. Smith and G.C. Temes, "An iterative approximation procedure for automatic filter synthesis," *IEEE Trans. on Circuit Theory*, vol.12, pp.107-122, March 1965.

[8] H.J. Orchard and G.C. Temes, "Filter Design using Transformed Variables," *IEEE Trans. Circuit Theory,* CT-15, pp. 385-405, 1968.

[9] Modern Filter Theory and Design, Eds. G.C. Temes and S.K. Mitra, Wiley, 1973. (See especially Chapters 1 - 3.).

[10] A.S. Sedra and P.O. Brackett, *Filter Theory and Design: Active and Passive*, 1978 Matrix Publishers.

[11] P. Gill, W. Murray, and M. Wright, *Practical Optimization,* pg. 136, Academic Press, 1981,

[12] T. H. Crystal and L Ehrman, "The Design and Applications of Digital Filters with Complex Coefficients," *IEEE Trans. Audio and Electroacoustics*, AU-16, No. 3, pp. 315-320, Sept. 1968.
[13] R. Boite and H. Leich, "On Digital Filters with Complex Coefficients," in *Network and Signal Theory* (edited by J.K. Skwirzynski and J.O. Scanlan), pp. 344-351, Peter Peregrinus, London, 1973.
[14] M.T. McCallig, "Design of Digital FIR Filters with Complex Conjugate Pulse Responses," *IEEE Trans. on Circuits and Systems,* CAS-25, pp. 1103-1105, Dec. 1978.
[15] A. Fettweis, "Principles of Complex Wave Digital Filters," *Int. J. Circuit Theory Appl.*, Vol. 9, pp. 119-134, Apr. 1981.
[16] X. Chen and T.W. Parks, "Design of FIR Filters in the Complex Domain,: *IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. 35, No. 2, pp. 144-153, Feb. 1987.
[17] P. Regalia, S. Mitra, and J. Fadavi-Ardekani, "Implementation of Real Coefficient Digital Filters Using Complex Arithmetic," *IEEE Transaction on Circuits and Systems,* Vol. CAS-34, No. 4, pp. 345-353, Apr. 1987.
[18] X. Chen and T. Parks, "Design of IIR Filters in the Complex Domain, "*IEEE Trans. on Acoustics, Speech, and Signal Processing*, Vol. 38, No. 6, pp. 910-920, June 1990.
[19] M. Komodromos, S. Russell, and P.T.P. Tang, "Design of FIR Filters with Complex Desired Frequency Response Using a Generalized Remez Algorithm," *IEEE Transactions on Circuits and Systems-II*, vol. 42, no. 4, pp. 428-435, Apr. 1995.
[20] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, 1975.
[21] W.M. Snelgrove and A.S. Sedra, "State-Space Synthesis of Complex Analog Filters," in *Proc. of the European Conference on Circuit Theory and Design*, pp. 420-424, 1981.
[22] G. R. Lang and P.O. Brackett, "Complex Analogue Filters," in *Proc. of the European Conference on Circuit Theory and Design*, pp. 412-419, 1981.
[23] R. Allen, "Complex Analog Filters Obtained from Shifted Lowpass Prototypes," M.A.Sc. Thesis, University of Toronto, 1985.
[24] A. Sedra, W. Snelgrove, and R. Allen, "Complex Analog Bandpass Filters Designed by Linearly Shifting Real Low-Pass Prototypes," in *Proc. of Int. Symp. on Circuits and Systems,* Vol. III, pp. 1223-1226, 1985.
[25] Q. Liu, "Switched-Capacitor Complex Filters," M.A.Sc. Thesis, University of Toronto, 1985.
[26] C. Cuypers, N. Voo, M. Teplechuk, and J. Sewell, "The General Synthesis of Complex Analogue Filters," 9'th Intl. Conf. on Elect., Circuits, and Systems, Vol. 1, pp. 153-156, 2002.