A filter designer's filter design aid: filtorX

Chris Ouslis, Martin Snelgrove, Adel S. Sedra

Dept. of Electrical Engineering
University of Toronto
Toronto, Ontario, M5S lA4
Tel: 416-978-4185
Fax: 416-978-7423
e-mail: ouslis@csri.toronto.edu

Introduction

As any filter designer is well aware, few tools exist for modem filter design, and integrated filter design tools are even more scarce. These difficulties lead designers to dedicating an undue amount of time to creating their own individual programs, and to ***massaging*** output data from from one design program to accommodate the input format of another. A drawback of most available filter design packages is the fixed nature of their design definitions: design programs expect their input to be specified in some given manner. Hence, designers must redefine their design problem so that it conforms to the constraints of the program. This scenario should be reversed: design tools should be reconfigurable to support the designer's given constraints.

To address these problems, we introduce filtorX, an interactive filter design package. Specifically, filtorX is an interpreted, or interactive, design aid which is based on the basic mathematicaI objects of filter design: complex numbers, and rational functions. In addition to these basic data objects, the necessary objects for a complete design are also included. Since filtorX is a programming language for filter design, it is fully customisable and hence extensible.

Designing the language

The goal of the project is to create a filter design tool suitable for filter designers with a range of design expertise. For most users, the necessary functionality provided, as library functions or built-in functions, would be sufficient to achieve the design objectives. Advanced designers, however, should not be restricted to the functionality as provided, but should be allowed to build upon existing techniques or alter them according to new design ideas or techniques not originally provided by the program. The only satisfactory solution to meet these requirements is to create a programming language and provide the fundamental design functions for the user.

To create a successful and useful language, a number of criteria were imposed: the language had to be

- intuitive to minimise the learning required

- convenient so that designers wouldn't have to get involved in details and could be freed to design and explore new design techniques
- extensible to provide the freedom to apply new ideas and not limit the application of unexpected techniques.

filtorX is inherently based on complex numbers; all mathematical functions operate on complex numbers, and all values are stored as complex numbers. Rational functions are stored as numerator and denominator roots (to reduce numerical inaccuracies) and have an associated leading coefficient. The rational functions represent gain, or transmission, functions; hence, the numerator roots represent loss poles while the denominator roots represent the natural-modes. The rational function can be expressed as

$$t(s) = \mathrm{leadCoeff}\frac{\prod_i (s-p_i)}{\prod (s-e_i)}$$

Filter optimisation

The basic task of filter design is to derive a suitable rational function which meets the given specifications. This has become a difficult and critical task requiring powerful numerical optimisers to deal with many simultaneous constraints. The optimiser used in filtorX uses a least-pth error criterion to manipulate the roots. The least-pth algorithm can be thought of as a generalisation of a least squares optimisation to any power as [1]:

$$E = \sum_i \left[\int_{C_i} \left| a_i(s) - o_i(s) \right|^{p_i} dl \right]^{1/p_i}$$

where a_i is the criterion of interest, o_i the specified ideal, and dl the element of distance along the contour C_i. The method used is a modified steepest-descent gradient approach [2]. The summation includes constraints from various regions, and of various criteria into the total error. Over each region, an ideal is specified for the object to be optimised; this comes in the form of a function defined in the language. The regions over which the rational function is evaluated are contours of the complex-plane. The contours take the form of a line along the imaginary-axis for continuous-time rational functions, or as arcs of the unit circle for discrete-time rational functions. The criteria that can be specified for each ideal include either the magnitude or group-delay responses of the rational function of interest. Specifying optimisation in this manner provides the designer with a great deal of latitude in creating a filter design. The more traditional approach of minimax optimisation, which can be defined as least-put optimisation as p approaches infinity, can be approximated with large values of p (computationallyp=50 is a reasonable maximum). An optimisation is also available using Remez' second method [3], but this is available only for the magnitude response of a rational function. A minimax type of specification is provided for this purpose, although it is limited in comparison to the ideals provided for least-pth optimisation. In dealing with group-delay response, a procedure is provided which simulates minimax optimisation by converting minimax specifications into least-pth ideals, then performs the optimisation for the user. An automatic simulation for the magnitude response is not yet complete.

Design examples

Some design examples will best serve to convey the syntax of the language, its capabilities and versatility. This is at the expense of rigor, since useful commands in the language will inevitably be neglected.

Inverse-sine, pre-distorted passband, lowpass filter

One design finding widespread application is that of lowpass reconstruction filters, as found in codecs. Their unusual characteristic is an inverse-sine, pre-distorted passband to compensate for the high-frequency roll-off caused by sample-and-hold circuits. Since this filter is often implemented by a switched-capacitor circuit, we will work in the z-domain. First passband and stopband contours will be defined:

```
contour pb ( arc 0j0, 1j0, 0j1 )
contour sb ( arc 0j0, exp( j * PI * 0.55 ), -1j0 )
```

The first contour defines the passband as an arc of centre 0, extending from unity on the real-axis to unity on the imaginary-axis. The stopband contour is denoted as *sb* and extends from an angle of 0.55π to π, or negative unity on the real-axis. As can be seen from the stopband contour example, any point on the complex-plane can be generated from a complex argument supplied to the exponential function **exp** and a scaling of the value through multiplication.

It is necessary to **define** ***functions*** to provide ideal values for the filter during optimisation. In the stopband, the ideal value would be a transmission of zero, while typically, in the passband the ideal would be one. For the purposes of this example, an ideal which is an inverse-sine response is desired in the passband. The following syntax defines the necessary functions:

```
function zero() {return(0)}
function sinc() {
        if (mag($1) < 1e-9) {
                return 1
        } else return sin(PI*$1)/(PI*$1)
}
function digLP() {
        # returns the scaled inverse-sinc value
        return( 1 / (sinc( angle( $1 ) / ( 2 * PI ))))
    }
```

Both functions and procedures are available in the language, and are differentiated by the property that procedures may not return a value and that functions must. Arguments to functions are accessed by a number convention which associates the location of the argument in the argument list, where the first argument is denoted by '$1'. A number of useful built-in functions exist including the magnitude function **mag, the** sine function, and a function which returns the angle (in radians) associated with a complex number, **angle.** Finally, the ideals may be specified as:

```
    ideal pbi pb L 2 magnitude digLP
    ideal sbi sb L 2 magnitude zero
```

The ideals have an associated object name, ***pbi*** and ***sbi*** respectively, and are defined over a given contour. The error measure can be defined as either an absolute error, the L indicated here, or as a relative error, or D measure. The relative error determines an average error value over the given contour then subtracts this value from the error to obtain an error centred around zero. This is used for group-delay purposes, since a designer is interested in the deviation of the error from some average value, not the absolute value of the group-delay. The next property specified is the error criterion; this can be either magnitude or delay (although only magnitude is appropriate for z-domain optimisations). The final specification is the name of the associated ideal function for the given contour.

No specific attenuation values have been provided, simply ideals (which are zero and one for piece-wise constant pass and stopbands). To obtain various passband and stopband errors(i.e. various degrees of attenuation), relative weights must be provided to the optimiser. These can be in the form of constants or functions. Weighting values assign a degree of importance to each ideal during the optimisation process, hence their values affect the resulting rational-function and therefore the passband ripple and minimum stopband attenuation. Assigning these weighting values requires some iteration from the designer and is the only non-intuitive phase of the optimisation process. The complete optimisation definition has required only thirteen lines of simple commands in the filtorX language. An example of an optimised eighth-order lowpass filter resulting from the example code is plotted in Figure 1.

Bandpass delay equalisation

Delay equalisers are essential components in many communications systems. The following is an example of a delay equaliser design for a sixth-order bandpass filter. The following is a minimax specification for the magnitude and group-delay:

```
spec eqSpec ( stopband( 0, 5, 40 )( 5, 10, 30 );
 passband( 20, 30, 0.5 );stopband( 40, 45, 50 ),( 45, Infinity, 40 ))
spec eqSpec ( gdelay( 20, 23, 0.045 )( 23, 27, 0.025 )( 27, 30, 0.045 ))
```

The triplets specify two break-points as frequency values, along with the final value which represents the attenuation in deciBels for the magnitude specification, and the allowed deviation in seconds for the **gdelay,** or group-delay, portion of the specification. The bandpass filter was obtained from a sixth-order seed using the Remez optimiser and is plotted in Figure 3. An allpass network will be cascaded with the bandpass filter to achieve the desired group-delay characteristics. The desired ideal is to achieve a flat group-delay variation, hence the ideal function for the equaliser is the negative of the group-delay of the filter, resulting in a net variation of zero.

```
function delayFunc() return ( gdelay( filter, $1 ) / -2.0 )
```

The **gdelay** function when supplied with a rational function and an imaginary argument returns the group-delay of the given rational function at that point. It should be noted that the group-delay value is halved due to a necessary trick required to optimise equalisers; only the natural-mode roots of a rational function are optimised to ensure imaginary-axis symmetry, hence the group-delay generated is half of that for an actual equaliser. The necessary contours and ideals are thus defined as (note the D used in the ideals to define the relative error criterion):

```
contour eqContour1 (line 0j20 , Oj23 )
contour eqContour2 (line Oj23 , Oj27 )
contour eqContour3 (line Oj27 , 0j30 )
ideal eqIdeal1 eqContour1 D 10 delay delayFunc
ideal eqIdea12 eqContour2 D 10 delay delayFunc
ideal eqIdea13 eqContour3 D 10 delay delayFunc
```

As was mentioned earlier, it is possible to simulate minimax specifications using least-p techniques by using large values of p (note the D 10 in the ideals above) and assigning weighting values which are the inverse of the specified deviation as:

```
eqWt1 = 22.22222222
eqwt2 = 40
eqwt3 =22.22222222
```

The defined specifications and the resulting group-delay response of the cascaded bandpass filter and equaliser are plotted in Figure 4.

Acknowledgement

This work was supported by the Information Technology Research Centre, a designated Centre-of-Excellence funded by the Province of Ontario.

REFERENCES

[1] Richard Schreier, ***Transfer Function Design,*** M.A.Sc Thesis, University of Toronto 1985.

[2] J.E. Dennis Jr. and R.B. Schnabel, ***Numerical Methods for Unconstrained Optimisation and Non-linear Equations,*** Prentice Hall, Englewood Cliffs, New Jersey 1983.

[3] A.S. Sedra and P.O. Bracket, ***Filter Theory and Design: Active and Passive,*** Matrix Publishers, Portland, Oregon 1978.

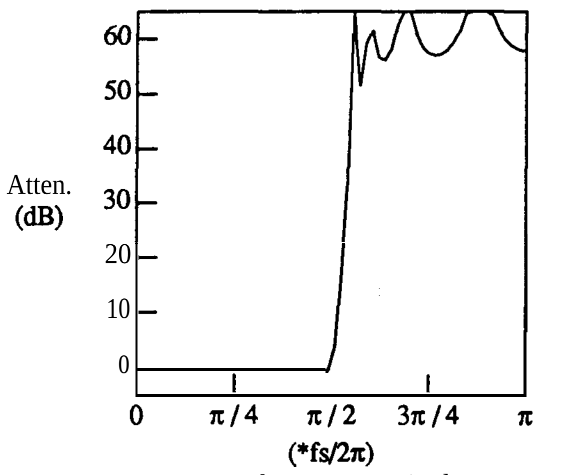

Figure 1. The lowpass magnitude response of an 8th-order SC filter with a sinc compensated passband

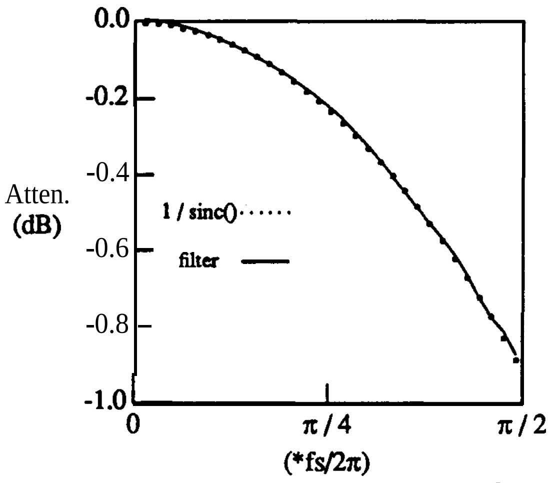

Figure 2. The sinc compensated passband of Fig. 1.

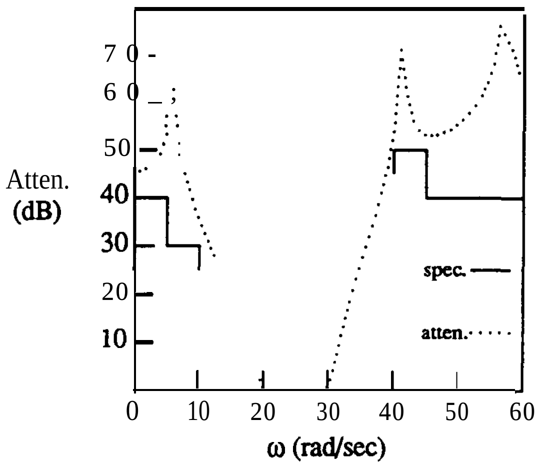

Figure 3. The bandpass specification with an equiripple optimised 6th-order filter response.

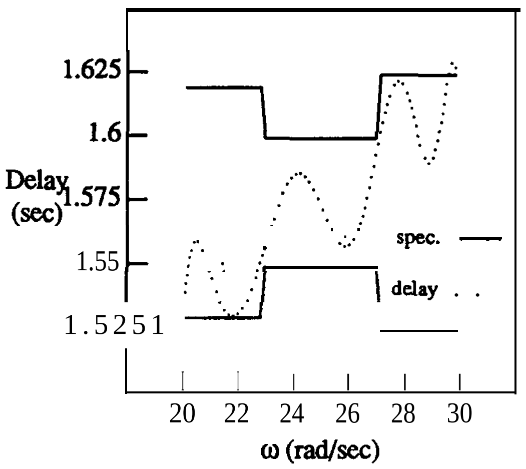

Figure 4. The equaliser specification and delay response of the bandpass, D 10 optimised 6th~order equaliser cascade.