FIR Filter Design ReadMe

James K Beard, Ph.D.

The FIR Filter Design program can be found at http://jameskbeard.com/jameskbeard/Files.html#FIRDSN

Table of Contents

Table of Figures

List of Tables

The FIR Filter Design Program

1 Summary

1.1 The ReadMe File

This ReadMe file provides a background and instructions for use of the programs used to provide the capability to design finite impulse response (FIR) or convolution digital filters for uniformly sampled data. The how-to sections include instructions on use of each of the program's capabilities and a thumbnail on how to use a FIR filter in a digital quadrature demodulator that can be implemented as a multiplexer-filter. The PDF format is used instead of plain text to allow the use of live links, figures, and equations.

1.2 What This Program Adds to Classical Capabilities

The FIR filter engine published by Parks, McClellan, and Rabiner in the IEEE and a landmark book on digital signal processing (see Section 2 for references) provides a method for designing FIR filters, and, with the references, the information needed to use the engine effectively. I took upon myself as a personal project to address speed and ease of use for real-world application in the mid 1980's, in particular allowing input of passband and stopband ripple in decibels in place of a linear weighting between them, and a new automated design process that, instead of a trial-and-error process, leaves one filter design parameter free, finding the free filter parameter through the secant method. Restructuring the program for Fortran 95+ and improving the user interface became a new project in the 2000's, and exploiting the legacy engine's capability to design filters with arbitrarily shaped passbands

was another. This version now also provides a GUI for the user, and all outputs needed to apply the FIR filter are available. A summary of these new capabilities:

- You control the program though GUI pop-up dialogues; no command-line interface,
- You input passband ripple and stopband attenuation requirements in dB,
- For low-pass or high-pass:
    - You can let the filter find the number of weights needed to meet requirements,
    - You can let the filter find passband width, stopband attenuation, or transition bandwidth if the number of weights is fixed,
    - Decimation and interpolation filters are explicitly supported,
- You can round the filter weights to a specified binary word length, compare the filter frequency response using the rounded weights to that using the un-rounded weights, and produce an output file that provides the rounded weights in decimal and hexadecimal with the scaling factors needed to preserve scaling in implementation,
- You get report-ready frequency response plots in both dB and linear scaling as well as in a CSV file for plotting dB scale frequency response in your own program or spreadsheet,
- The program uses double precision so that large filters can be designed with accurate filter weights, and
- The legacy multiple-band user input is still available as an option, as are differentiators and Hilbert Transformers.

1.3 How to Use the Program

Most FIR filters are low-pass filters with known requirements. For this type of design, you probably don't need to read this ReadMe file; just go ahead and use the program, and read the pop-up dialogues carefully the first few times you use it.

If you have questions or run into unexpected issues, or need to design high-pass filters, multi-band filters, differentiators, Hilbert transformers, or user-defined passband shape filters (equalizers), or you are designing your first digital quadrature demodulator, you should read the relevant parts of this ReadMe file before you begin.

Sophisticated applications of FIR filters include decimation, interpolation, and digital quadrature demodulators implemented as a decimation filter combined with a multiplexer. These are treated in how-to sections under Section 3.

1.4 Types of Filters Designed

The program is capable of designing

- Low-pass, and high-pass,
- Multiband bandpass filers,
- Differentiators,
- Hilbert transform filters, and
- User-specified frequency response filters.

We include an example of a user-specified audio equalizer for a speaker system or hearing aid. The low-pass and high-pass capability is extended to designs specifically parameterized for decimation and interpolation filters, where, on downsampling, the stopbands alias into the passband, and the "don't-care" or transition regions are positioned symmetrically about the folding frequencies. All user inputs are via GUI pop-up dialogs.

Linear phase FIR filters cannot produce some types of designs such as high-pass filters with an even number of weights, and the reasons for this are not immediately obvious. Section 3 below is a "How-To" guide that deals with these restrictions; section 4.2 details the theory behind the restrictions.

Use of the FIR filter design program for interpolation and decimation is straightforward, and is explained in Section 3.3. Other applications are a digital quadrature demodulator, in which a real signal is sampled at I.F. and the operation of multiplying by a complex exponential and low-pass filtering the result to provide a complex signal at baseband, and design of general frequency response FIR filters. A hearing aid equalizer is provided as an example of a general frequency response filter along with the program files, and is explained in Section 3.

1.5 Program Outputs

1.5.1 Displays

The classical FIR filter ASCII output is displayed along with a "go-no-go" prompt. The user can observe the number of weights, the passband and stopband attenuation, and other design details before allowing the filter weights and frequency responses to be computed, or the design can be disapproved and the program will loop back for a new design.

The frequency response of the final filter, with and without truncation of the weights to a user-specified word length, is displayed as a full-screen plot as the run ends.

1.5.2 Files and Displays

Output files are all in the format yyyymmdd_hhmm_<title>.<ext> and include

- A text log file with a file name beginning with the date and time in the format that includes the FIR filter design output in the classical format used in the original FORTRAN programs,
- The filter frequency response in decibels versus frequency in a CSV file formatted for quick plots using a spreadsheet, and
- The filter weights normalized and rounded to a user-specified number of bits in decimal and hexadecimal notation in a CSV file.
- The legacy text output file provides a look at the filter parameters as background for a "go-ahead" prompt to complete the design.
- All frequency response plots with dB and linear scales are stored as PNG files.

The log file is echoed in a text screen that is the main window for the application.

1.6 History of the Technology and of This Program

The original FIR theory and technology development was done in partial fulfillment of Ph.D. requirements by J. H. McClellan and published in the IEEE Transactions on Audio and Electroacoustics in 1972; the papers reference the classical work of Remez on Chebychev's problem of optimal polynomial approximation and its practical solutions. The program itself was re-published in 1973 with the authors McClellan, Parks, and Rabiner. An expanded theoretical treatment and a slightly updated FORTRAN program forms a major part of the digital signal processing reference book *Theory and Application of Digital Signal Processing* by Rabiner and Gold, published in 1975. See Section 2 for the references.

The program that encapsulates the FIR design engine in a requirements-oriented design capability was originally programmed in TRS-80 RATFOR by James K Beard in 1986, including user inputs of ripple and stopband in decibels and a recursive search that allows specifying low-pass or high-pass filter performance and finding the necessary number of weights or other parameter left free. A re-write in Fortran 95 that included structuring the legacy engine modules in Fortran 95 was one update in 2001. The program was updated with a Fortran 2008 compiler using the Absoft Fortran compiler's Absoft-specific pop-up dialog and plotting capabilities in 2013, with updates in 2014 (decimation filters and arbitrary frequency response) and 2015 (user-specified frequency response equalizer design, updated user interface, and added ReadMe file).

1.7 Revision History

The revision history is given in Table 1. Versions prior to adding the GUI are not listed.

Table 1. Revision History

Release	Changes	Date
Revision 0.90, Release Candidate 1	First GUI and user-defined equalizer capability for old program	2013
Revision 0.91, Release Candidate 2	Added frequency response plots	2014
Revision 0.92, Release Candidate 3	Added ReadMe, enhanced input parsing and error messages for subtle input errors, bug fixes	2015

2 Principal References

2.1 Books

Lawrence R. Rabiner and Bernard Gold, *Theory and Application of Digital Signal Processing*, Prentice-Hall (1975); ISBN 0-13-914101-4, ISBN-13: 978-0139141010; Paperback: Prentice-Hall of India (1992) ISBN-10: 0-87-692501-8, ISBN-13: 978-0876925010; ASIN: B001G413JG pages 194-204 (FORTRAN program), the entire portion of the book to that point develops and explains the theory of FIR filters that underlies the implementation.

James K Beard, *The FFT in the 21st Century*, Springer (2003); hard covers ISBN-10 1402076754, ISBN-13 978-1402076756; soft covers ISBN-10 1441954104, ISBN-13 978-1441954107; Kindle ASIN

B000QEN5G6. Also available from Springer as an e-Book. Chapter 3 is an exhaustive treatment of spectral windowing for digital data streams, including Dolph-Chebychev, Taylor, and Bayliss.

2.2 Papers

2.2.1 The Computational Engine, its Limitations, and its Parameters

T.W. Parks and J.H. McClellan, "A Program for the design of Linear Phase Finite Impulse Response Digital Filters," IEEE Trans. on audio and Electroacoustics, vol. AU-21, No. 3, 195-199, August 1972.

J. H. McClellan, T. W. Parks, and L. R. Rabiner, *A Computer Program for Designing Optimum FIR Linear Phase Digital Filters*, IEEE Transactions on Audio and Electroacoustics, vol. AU-21, No. 6, pp 506-526, December 1973.

O. Herrmannn, L. R. Rabiner, an D. S. K. Chan, *Practical Design Rules for Optimum Finite Impulse Response Lowpass Digital Filters*, Bell System Technical Journal, vol. 52, No. 6, pp 769-799, July-August 1973.

2.2.2 The Remez Inequality and Approximation Algorithm

Evgeny Yakovlevich Remez, *General Computational Methods of Chebychev Approximation*, Atomic Energy Translation 4491, Kiev, 1957; apparently a translation of the two Compt. Rend. papers.

Evgeny Yakovlevich Remez, “Sur les méthodes pour réaliser la meilleure approximation des fonctions d'après le principe de Tchebychef,” Kiev, Académie des sciences mathématiques, 1935. In-4°, 162 p., fig.

Evgeny Yakovlevich Remez, "Sur la détermination des polynômes d'approximation de degré donnée", Comm. Soc. Math. Kharkov 10, 41 (1934).

Evgeny Yakovlevich Remez , "Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation”, Compt. Rend. Acad. Sc. 198, 2063 (1934);

Evgeny Yakovlevich Remez , "Sur le calcul effectiv des polynômes d'approximation des Tschebyscheff", Compt. Rend. Acade. Sc. 199, 337 (1934).

See also the Wikipedia articles:

- https://en.wikipedia.org/wiki/Remez_inequality
- https://en.wikipedia.org/wiki/Remez_algorithm

2.2.3 Decimation Filters and Digital Quadrature Demodulation

James K Beard, *Optimization of Discrete Signal Processors using Array Processor and CCD Technology*, IEEE ICASSP, April 1979.

2.2.4 Spectral Windows for Use with an FFT

James K Beard, *Planar Array Design and Performance*, presented at the Philadelphia IEEE, June 26, 2014, slides available at http://jameskbeard.com/jameskbeard/Papers.html#SelectedPapers. An error in the literature on Bayliss windows, useful for pulse-splitting time-of-arrival estimation, is corrected and high-

performance Bayliss windows in one and two dimensions are demonstrated. See my 2003 book, Chapter 3, for an exhaustive treatment of this and other spectral widows in one dimension (digital data streams).

2.3 Current Versions of the Program

The current release of this file and a ZIP archive containing the program and this ReadMe file is online at

http://jameskbeard.com/jameskbeard/Files.html#FIRDSN

3 Using the Program

3.1 Installing and Starting the Program

The program is currently a 32-bit Windows executable compatible with Windows 10 and previous Windows 32-bit and 64-bit systems. The files in Table 2 are included in a ZIP archive and should all be put in the same folder. A shortcut can be generated that can be put anywhere, such as the desktop, a user interface, or a folder. If the startup folder is specified in the shortcut, the DLLs must be in the startup folder. The Equalizer.xlsx and Equalizer.csv files are there to support an example and can be anywhere you want to run the program. The output files will be generated in the default folder where the program starts; you can set the start folder in a shortcut. The program will allow you to browse your computer for the equalizer data input file.

Table 2. FIR Filter Design Files

File name	Purpose	License
FIR_Design.exe	Principal executable	James K Beard
libgomp.dll	Multi-Core Support	LGPL (v2 only)
Qtcore4.dll	Windows and Dialogs	LGPL (v2.1 only)
Qtgui4.dll	Windows and Dialogs	LGPL (v2.1 only)
Equalizer.xslx	Example – Equalizer Design	James K Beard
Equalizer.csv	Example – Equalizer CSV file	James K Beard

The license "LGPL" referred to in Table 2 is the Lesser GNU Public License, which is given with its versions at

https://www.gnu.org/copyleft/lesser.html

The license "James K Beard" means that the program is copyright 1986, 2001, 2013-2015 by James K Beard, all rights reserved. Your use of the program is predicated on your acceptance of the conditions in the first pop-up dialogue, shown in Figure 1. The program opens with a maximized text window and this dialogue. The user must check the box marked "Agree" and clock on "OK" to proceed; leaving the box unchecked or clicking "Cancel" will exit the program, leaving the text box up, but with no capability except display, save as text file, and exit. Checking the "Agree" box and "OK" allows the program to proceed and an extra menu item "FIR Design" to the text window, as shown in Figure 2 below. Once the run has been completed, or terminated at your option, you can re-start the program by selecting "New FIR Filter Design" from that new menu item.

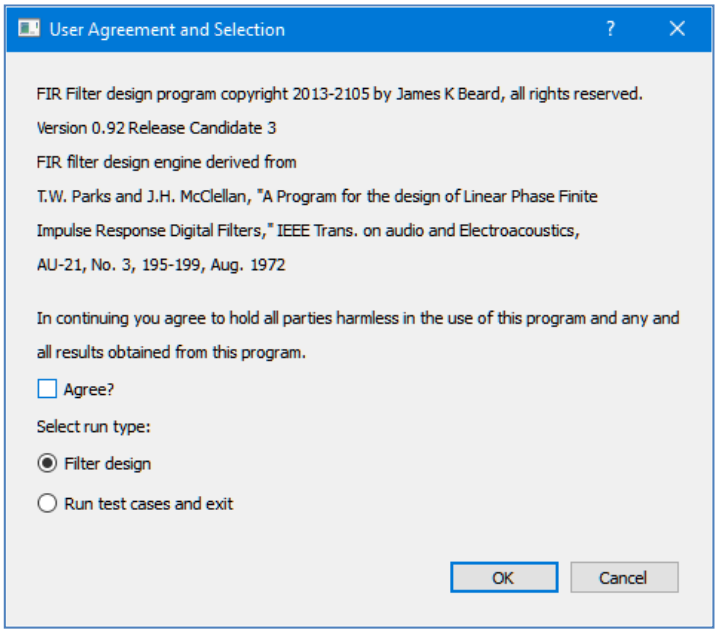


Figure 1. Opening Dialogue; Appears Only Once Per Run

The “Run test cases and exit” reproduces the examples from the legacy publications.

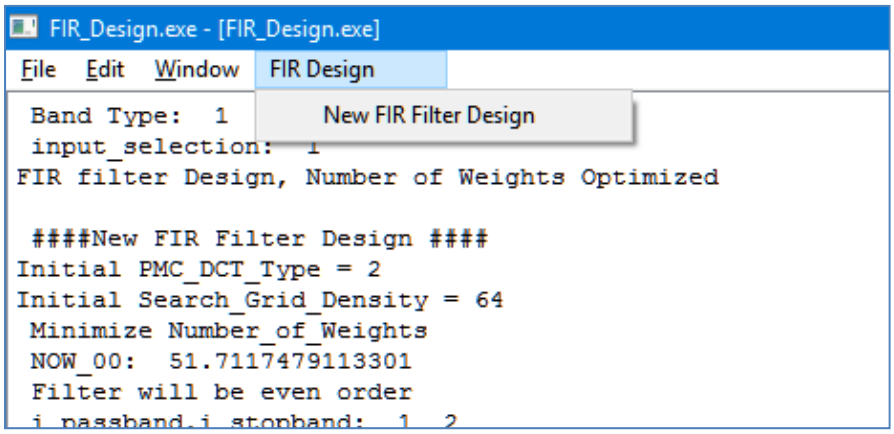


Figure 2. New Run Menu Item; Use for New Design Once Program is Started

3.2 Simplest use: Low Pass or High Pass Filter

Once the opening dialogue is passed, the filter type selection dialogue of Figure 3 appears. The default radio button is a low-pass design. Click “OK” to proceed to specify that you want a FIR low pass filter. The high pass filter design process is similar to that of the low-pass design process, other than that an even number of weights is not possible.

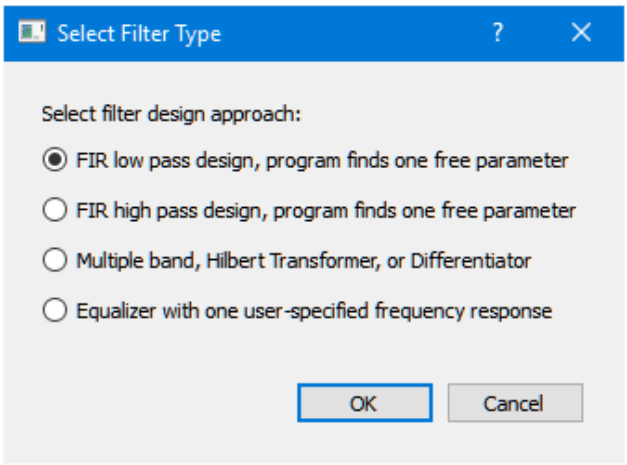


Figure 3. Filter Type Selection Dialogue

If a FIR low-pass or high-pass design is selected, then the program presents the dialog of Figure 4.

This dialog provides control of the most important feature of this program: the user determines which of four filter specifications is to be left free for the program to determine. The program uses the approximations published by Hermann, Rabiner and Chan (and elucidated at some length in the book by Rabiner and Gold) as a starting point, then uses actual filter designs in a secant method algorithm to determine the free parameter that provides a FIR filter design that meets or exceeds requirements.

A FIR filter is characterized in the Z plane as all zeros, with the zeros on the unit circle in the stopband and near the minima of the ripples in the passband. Passband zeros are in pairs, one inside the unit circle and the other outside the unit circle. As all filter design parameters save one are held constant and the free parameter is varied slowly, the zeros migrate. As the free parameter changes, the zeros migrate between the passband and the stopband. If the free parameter is the number of weights, the number of zeros is equal to one less than the number of weights. The achieved parameters exhibit a step change as the number of zeros in the passband changes, so an exact match to the fixed design

parameters may not exist. The algorithm detects when this happens and selects the value of the free parameter that slightly exceeds requirements.

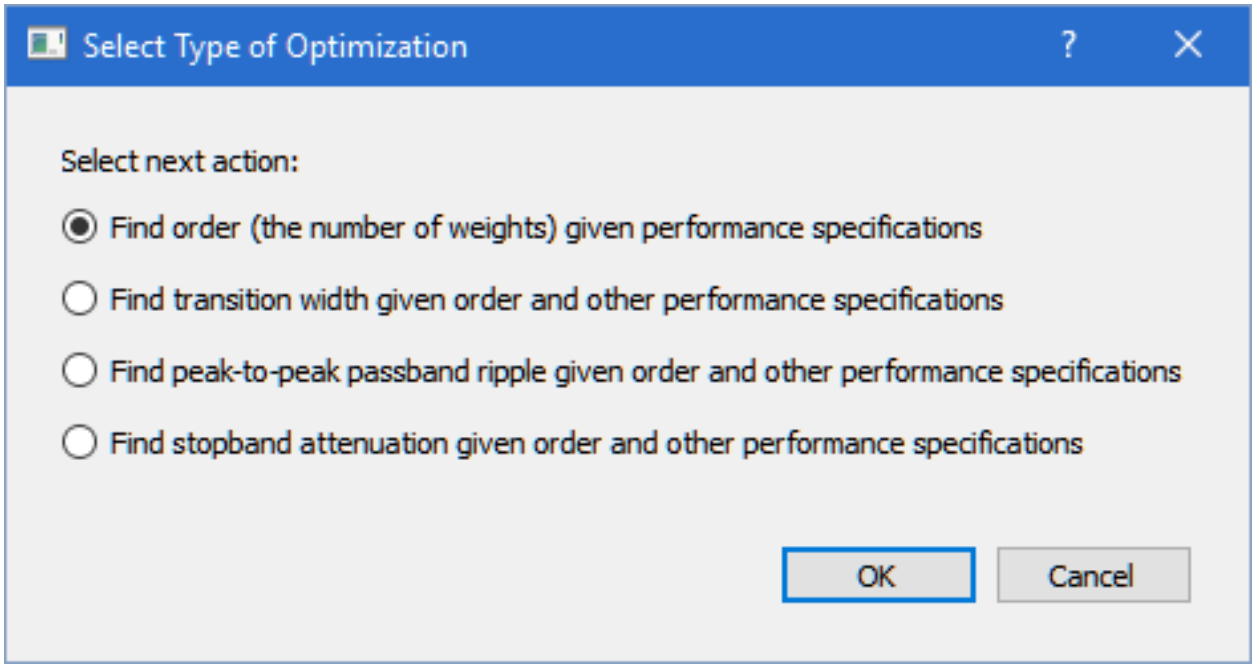


Figure 4. Dialogue for Selection of Free Parameter

At this point, the program needs to know whether you are designing a decimation filter or a general low pass filter, and you will get the dialogue shown in Figure 5. The default is a normal FIR filter; decimation and interpolation filters are discussed in Section 3.3 below as a separate design process; click "OK" to proceed.

This brings us to the parameters of the filter that you define. These are presented to you in the next dialogue, shown in Figure 6 below.

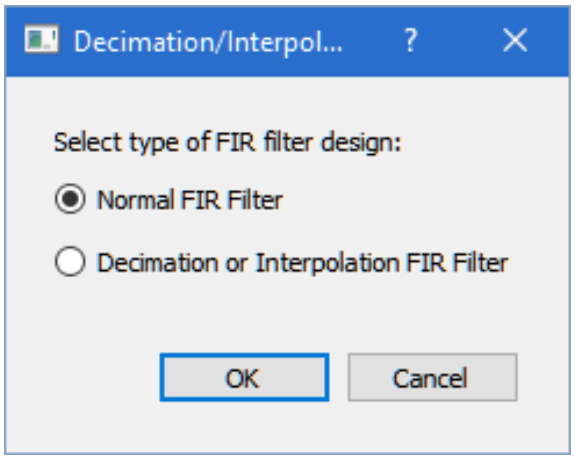


Figure 5. Normal or Decimation Selection Dialogue

Note that the dialogue begins with three radio buttons, which allow the order of the filter (the number of weights) to be unconstrained, or to be forced to be odd or even. This is because the implementation may define whether the number of weights must be odd or even. If the latency of the filter must be an integral number of sample times, then the order must be odd. If the available convolution filter hardware allows an even number of weights, then of course the order of the filter must be even. A high-pass filter must be odd order to prevent forcing a discontinuity in the frequency response, as explained in Section 4.2. If a high pass filter is being designed, additional text in the prompts of the dialogue shown in Figure 6 will warn against allowing an even number of weights.

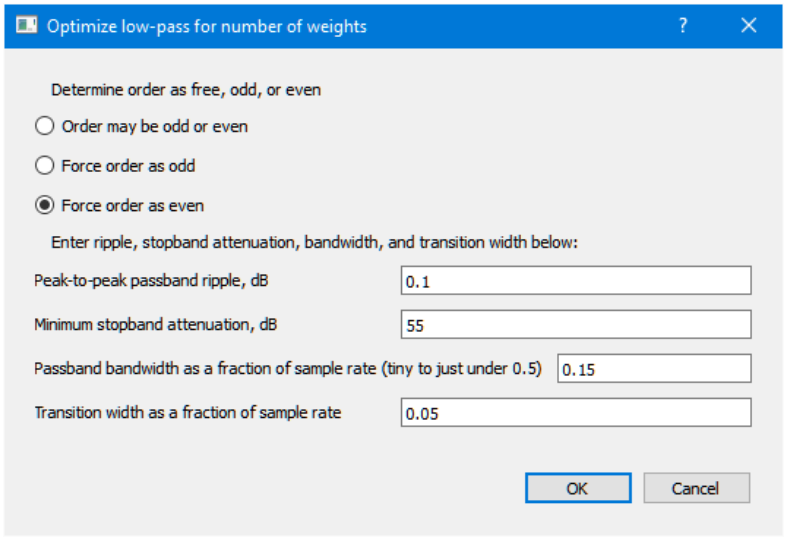


Figure 6. Specification Dialogue for Low Pass Filters

The user specifies the peak-to-peak passband ripple in decibels, the stopband attenuation in dB, the passband width, and the width of the "don't care" region between the top of the passband and the bottom of the stopband. The numbers shown in Figure 6 are the defaults. Enter your own values and press OK. Note that the sum of the passband bandwidth and the transition width must be less than 0.5 to allow for a nonzero bandwidth for the stop band.

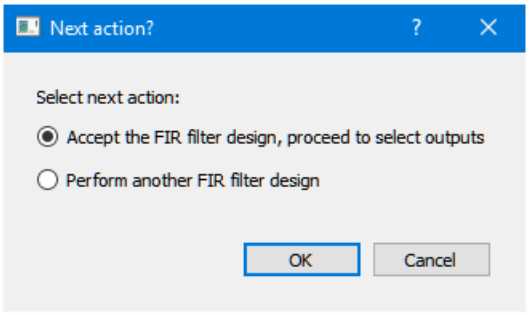


Figure 7. Prompt to Approve a Design

The filter design will then be done, and then there will be a "last chance" prompt as shown in Figure 7. The text box will show the classical filter design output that is used in the book and papers that include Parks and McClellan that are cited in Section 2; this output for the inputs from Figure 6 are shown in Figure 8. The success of the design can be seen immediately from the small values in the "Deviation" rows, which are in bold type in Figure 8. The "Extremal frequencies" are the frequencies of the ripple peaks.

```
**********************************************************************

               Finite Impulse Response (FIR)
             Linear Phase Digital Filter Design
                 Remez Exchange Algorithm

                      Bandpass Filter

                    Filter Length = 56

                  ***** Impulse Response *****
             h( 1) = -1.07805312E-03 = h( 56)
             h( 2) =  3.10888884E-05 = h( 55)
             h( 3) =  1.79203528E-03 = h( 54)
             h( 4) =  2.91983502E-03 = h( 53)
             h( 5) =  1.40601166E-03 = h( 52)
             h( 6) = -2.19413286E-03 = h( 51)
             h( 7) = -4.20884128E-03 = h( 50)
             h( 8) = -1.40772505E-03 = h( 49)
             h( 9) =  4.45293275E-03 = h( 48)
             h(10) =  6.92127273E-03 = h( 47)
             h(11) =  1.50487931E-03 = h( 46)
             h(12) = -7.74449424E-03 = h( 45)
             h(13) = -1.04088602E-02 = h( 44)
             h(14) = -8.57398061E-04 = h( 43)
             h(15) =  1.29587131E-02 = h( 42)
             h(16) =  1.50663323E-02 = h( 41)
             h(17) = -1.05206717E-03 = h( 40)
             h(18) = -2.12204414E-02 = h( 39)
             h(19) = -2.14786858E-02 = h( 38)
             h(20) =  5.43619473E-03 = h( 37)
             h(21) =  3.54510111E-02 = h( 36)
             h(22) =  3.16197150E-02 = h( 35)
             h(23) = -1.57193368E-02 = h( 34)
             h(24) = -6.63289672E-02 = h( 33)
             h(25) = -5.48293316E-02 = h( 32)
             h(26) =  5.11443210E-02 = h( 31)
             h(27) =  0.21070088     = h( 30)
             h(28) =  0.32951125     = h( 29)

                        Band  1       Band  2
  Lower Band Edge    0.00000000    0.20000000
  Upper Band Edge    0.15000000    0.50000000
  Desired value      1.00000000    0.00000000
  Weighting          1.00000000    3.23706113
  Deviation          0.00477627    0.00147549
  Deviation, dB      0.08297287  -56.62124819

  Extremal Frequencies--Maxima of the Error Curve
     0.0000000   0.0206473   0.0407366   0.0608259   0.0797991
     0.0987723   0.1160714   0.1322545   0.1450893   0.1500000
     0.2000000   0.2039063   0.2145089   0.2284598   0.2440848
     0.2602679   0.2775670   0.2948661   0.3121652   0.3300223
     0.3478795   0.3657366   0.3835937   0.4014509   0.4193080
     0.4371652   0.4550223   0.4728795   0.4907366

**********************************************************************
```

Figure 8. Classical McClellan and Parks FIR Filter Design Output

The dialogue shown in Figure 9 will appear, providing you with an option to round the filter weights to a specified number of bits. Enter zero to skip rounding the filter weights. The integer format is ones-complement, the standard small computer format, with a sign bit, so a meaningful output is to an integer with two or more bits.

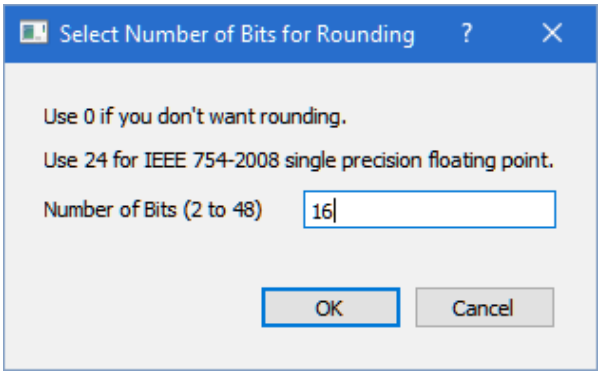


Figure 9. Rounding to Word Length Dialogue

Click OK and the program will proceed to write the output files and compute plots of the frequency response. The plot in the foreground is similar to that of Figure 10. The other plots are discussed in Section 3.7.

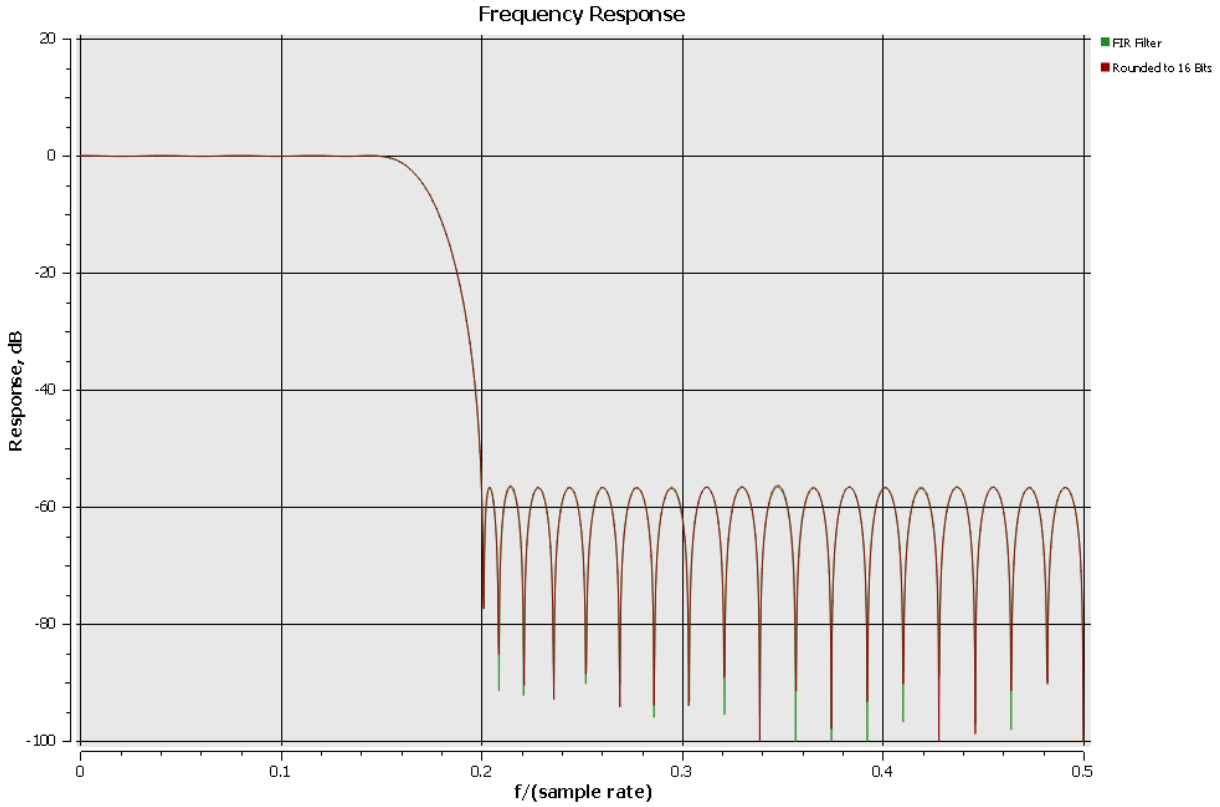


Figure 10. Low Pass FIR Filter Frequency Response

The plot will be maximized. De-maximize or delete the plot window; that will de-maximize the text window and all six plot windows. The program saves all plots as PNG files.

The frequency response plot using the computed filter weights is shown in green, and over it a plot using the filter weights rounded to the specified number of bits. When the plots overlay almost exactly,

as shown in Figure 10, the specified accuracy is sufficient for the filter with rounded weiths to meet specified performance requirements.

You may close the plot windows and use the added text menu item to perform another design. There are two points to observe:

- If you close the application window, you must run the program again and the dialogue of Figure 1 will be repeated.
- Do not close the text window, as a new one will not be generated when you make another run. This can make response to the prompt from the pop-up at Figure 7 guesswork. If you accidentally close the text window, close the application and restart the program.
- The text window and text output file will be appended as you continue to use the program until you close the application. Note that the log is repeated in the output file yyyymmdd_<run type>.txt so that you do not need to save the text from the text window.
- The six PNG files that show the frequency responses will be overwritten once you click "OK" on the prompt shown in Figure 9. Use, move, or rename them if you don't want them overwritten.

3.3 Decimation and Interpolation Filters

When the "Decimation or Interpolation filter" radio button is selected in the dialogue of Figure 5, the filter is specified to optimize it or decimation or interpolation.

In decimation, for a decimation ratio of K, an integer greater than 1, the filter is implemented as a convolution between the input data stream and the filter weights computed at the input sample rate, but only every K^{th} output sample is computed for the output data stream.

In interpolation, use a decimation filter and insert K-1 zero samples between each input data sample and filter the resulting data stream.

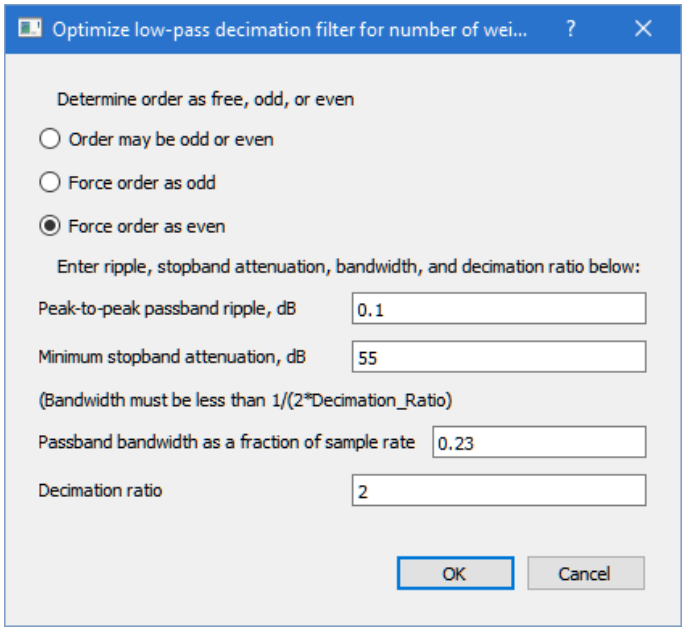


Figure 11. Decimation and Interpolation Specification Dialogue

Decimation aliases data, noise and interference at frequencies higher than the decimated sample rate into the passband. In decimation filtering, the filter stopband attenuation limits this bleed-through.

In interpolation, the filter stopband prevents the signal spectrum from being repeated an extra K-1 times. Please refer to Figure 12 for considerations in specifying a FIR filter for decimation or interpolation.

The frequency $f_S/2K$ becomes the Nyquist frequency for the lower sample rate, so the passband width must be less than $f_S/2K$. The beginning of the stopband, optimally, aliases to the top edge of the passband, and this branch of the design process designs the filter to do exactly that.

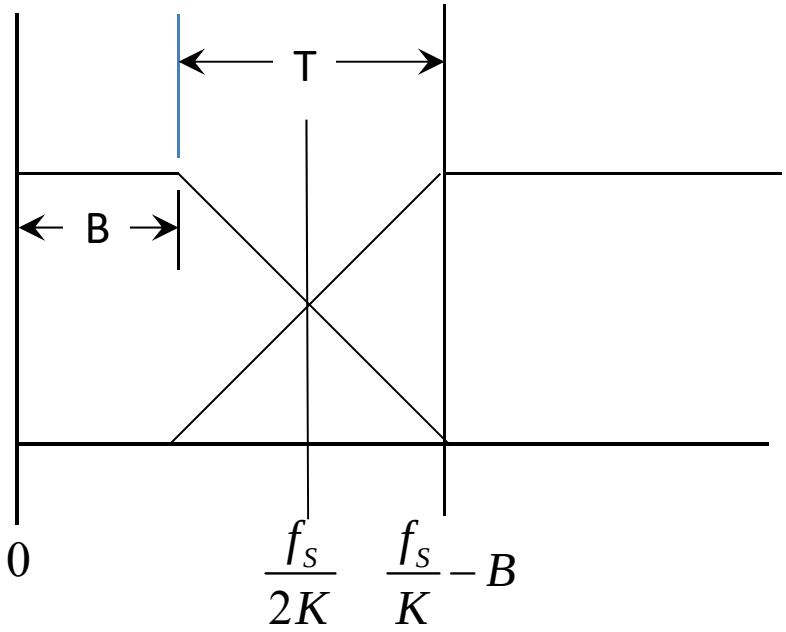


Figure 12. FIR Filter Considerations in Decimating by a Factor of K

The specification dialogue for interpolation and decimation filters is shown in Figure 11. Comparing this dialogue with that shown in Figure 6, note that instead of the transition width, the decimation ratio is specified. The transition width is determined from the inputs in the dialogue of Figure 11 and is

$$\frac{\langle \text{Transition width } T \rangle}{\langle \text{Sample rate } f_S \rangle} = \frac{1}{\langle \text{Decimation ratio } K \rangle} - 2 \cdot \frac{\langle \text{Passband bandwidth } B \rangle}{\langle \text{Sample rate } f_S \rangle} \; . \qquad (3.1)$$

The rest of the design process proceeds as with the low-pass or high-pass filter. If you prefer to specify your interpolation or decimation filter normally instead of using the dialogue of Figure 11, click the "Normal FIR Filter" radio button when the prompt shown in Figure 5 appears.

A decimation filter may be high-pass. This may be desirable when there is a DC (zero frequency) component in the data but the desired signal is in a band that does not include zero frequency, and the sampling and aliasing are selected to alias the signal band center to Nyquist (e.g., half the sample rate) instead of baseband, thus rejecting the DC component by the stopband attenuation of the filter. See digital quadrature demodulation, Section 3.6 for cases where the signal is not at baseband.

A high-pass filter may not have an even number of weights because the frequency response of a bandpass filter will always be zero at half the sample rate, which is the top of the band – a contradiction with the requirement that the filter be a high-pass. See Section 4.2 for details on why this is so.

3.4 Multiple Bandpass and the Classical Inputs

The classical inputs for the Parks, McClellan, and Rabiner programs provided as source code in the references of Section 2 are available by clicking on the ratio button labeled “Multiple Band, Hilbert Transformer, or Differentiator” on the beginning dialogue given in Figure 3 and clicking OK. This brings up the selection dialogue given in Figure 13.

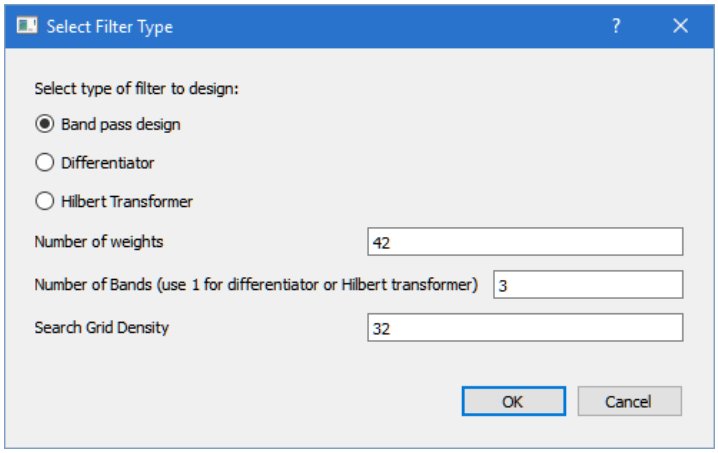


Figure 13. Dialogue for Classical Inputs

The entire paradigm of how the run proceeds from here regresses to that of the legacy program. Although GUI dialogues do support the run, the inputs and outputs of the legacy program are present. Select the radio button for the type of FIR filter that you want to design.

3.4.1 Multiple Band Bandpass Filters

The passband ripple and stopband attenuation are not inputs in this branch of program execution, but instead weights on the Chebychev approximation errors for each band are the user inputs. To achieve specified passband ripple and stopband attenuation, either use a dummy run of a standard lowpass design with your ripple and stopband attenuation requirements and select the stopband weighting from the legacy output as shown in Figure 8, or use equation (4.16) in Section 4.3.

Select the number of weights, number of bands, and search grid density in the dialogue shown in figure Figure 13. The bands will specified in the following dialogues. Note that the default Remez search grid density is 32; the legacy program uses 16 but experience has shown that 32 provides excellent accuracy for larger filters with no noticeable increase in computer loading. The automated designs used in the other execution branches use 32. Note that to reproduce the

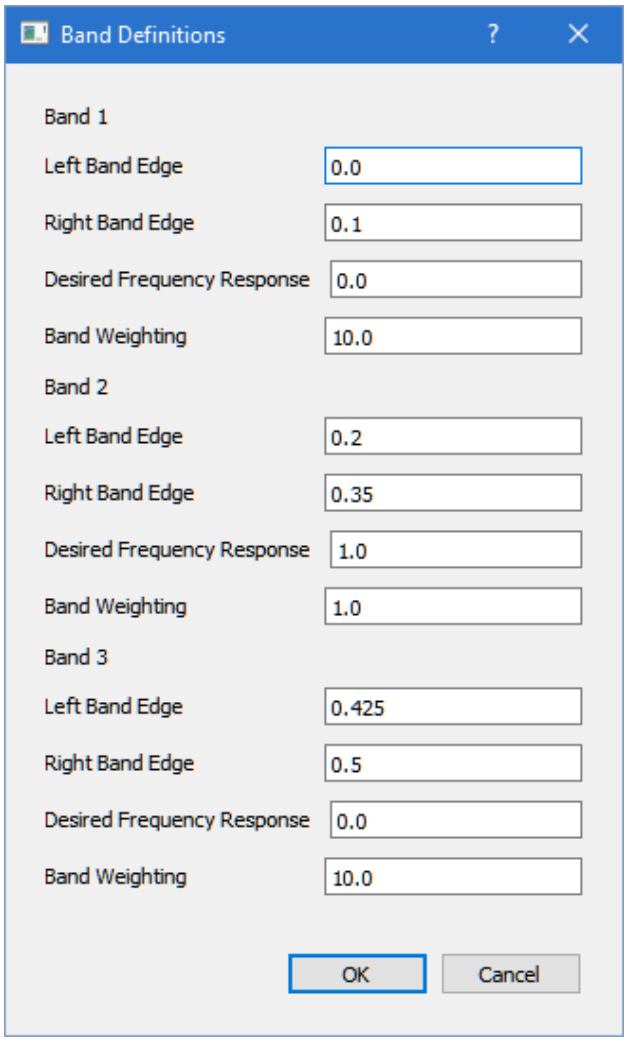


Figure 14. Band Input Dialogue

examples of the legacy publications exactly, a grid density of 16 must be used.

The next dialogue is shown in Figure 14. The band edges, desired frequency response, and band weighting are required for each band. A maximum of five bands will be presented per dialogue. If you have more than five bands, the dialogue will be repeated as necessary until all the band inputs are obtained.

At this point the filter is designed and the acceptance dialogue of Figure 7 is presented, and the legacy ASCII filter design output as shown in the example of Figure 8 is shown in the text window, and the program proceeds as with the other execution branches.

3.4.2 Differentiator and Hilbert Transformer

The derivative of a digital data stream is not a well-defined concept, but a frequency domain description of the derivative can be formulated as the Fourier transform of the signal times frequency. This rationale is evident on examination of the definition of the inverse Fourier transform,

$$\left.\begin{aligned} f(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)\cdot\exp(+j\cdot\omega\cdot t)\cdot d\omega \\ \frac{df(t)}{dt} &= \frac{j}{2\pi}\cdot\int_{-\infty}^{\infty} \omega\cdot F(\omega)\cdot\exp(+j\cdot\omega\cdot)\cdot dw \end{aligned}\right\} \tag{3.2}$$

which is valid when both integrals exist, as they will with differentiable time functions of finite amplitude and duration.

The Hilbert transform is used in harmonic analysis. One way of understanding the Hilbert transform is that if two real signals $f(t)$ and $g(t)$ make up a complex analytic signal by

$$fc(t) = f(t) + j\cdot g(t) \tag{3.3}$$

then the real signals $f(t)$ and $g(t)$ are Hilbert transforms of each other. In addition, the Fourier transform of $fc(t)$ is zero for negative frequency. This means that an FFT that accepts complex input can accept the sampled signal as its real data and its Hilbert transform as its imaginary data and produce a discrete Fourier transform of the signal, which can eliminate adding zero data or extra steps in the Fourier transform algorithm to deal with real data.

The current Wikipedia article has more information on the history, mathematics, applications, and research areas of Hilbert transforms with references for further reading:

https://en.wikipedia.org/wiki/Hilbert_transform

Be sure and make the number of bands 1 in the dialogue of Figure 13. Note that a both a differentiator and a Hilbert transformer are high-pass filters that are odd in the time and frequency domains, and designs go better when an even number of weights is used. See Figure 25 for an example of a

differentiator with an odd number of weights (frequency response plotted on a linear scale), and compare with Figure 26; the Chebychev fit of the frequency response will be more accurate if large steps over a small frequency range are not required.

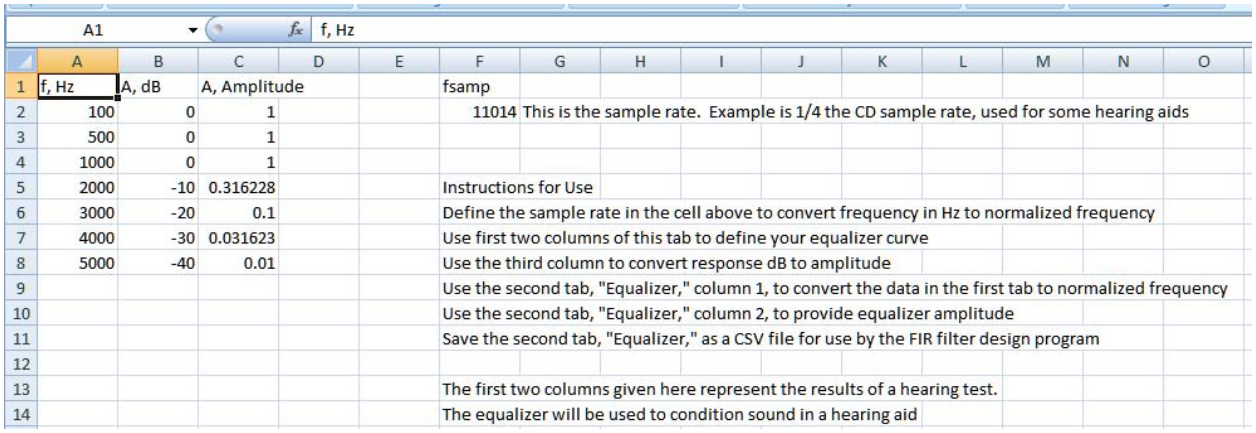
The Hilbert transformer inherently has a large step at zero frequency, as shown in Figure 27, so don't specify the low end of the band as zero.

The design proceeds as with the muliti-band bandpass filter. The "Desired response" is a scale factor and should be left at 1.0. The band weighting has no effect when there is only one band, but it must be positive for the computational engine to work properly.

3.5 User-Defined Equalizer

An equalizer is a filter that has a user-defined frequency response. An example is included with this program for a hearing aid equalizer that compensates for hearing loss that increases with frequency.

The first step is to define your desired frequency response. Please refer to the example file Equalizer.xslx, first tab, as shown in Figure 15. Input your data in the format in the table in the first two columns, frequency in the first column, column A, and desired response in dB in the second column, column B. Note that the frequency intervals are whatever you like; the example assumes that you have data from a hearing test machine provided by a qualified audiologist or speaker system frequency response measurement. The third column, column C, uses a spreadsheet equation to convert the dB figures in column B to amplitude.



A1 f, Hz

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	f, Hz	A, dB	A, Amplitude			fsamp									
2	100	0	1			11014	This is the sample rate. Example is 1/4 the CD sample rate, used for some hearing aids								
3	500	0	1												
4	1000	0	1												
5	2000	-10	0.316228			Instructions for Use									
6	3000	-20	0.1			Define the sample rate in the cell above to convert frequency in Hz to normalized frequency									
7	4000	-30	0.031623			Use first two columns of this tab to define your equalizer curve									
8	5000	-40	0.01			Use the third column to convert response dB to amplitude									
9						Use the second tab, "Equalizer," column 1, to convert the data in the first tab to normalized frequency									
10						Use the second tab, "Equalizer," column 2, to provide equalizer amplitude									
11						Save the second tab, "Equalizer," as a CSV file for use by the FIR filter design program									
12															
13						The first two columns given here represent the results of a hearing test.									
14						The equalizer will be used to condition sound in a hearing aid									

Figure 15. Equalizer.xlsx Tab 1

The sample rate of the digital processor to be used is also entered on the first tab, in cell F2 of the first tab in Equalizer.xlsx. The units of frequency must be the same as you used in column A. In Equalizer.xlsx, all frequencies are in Hz.

The second tab of Equalizer.xlsx, shown in Figure 17, is computed from the data on the first tab. Column A has a title, "f" in the first cell while the rest of Column A is corresponding cells in column A of tab 1 divided by the sample rate in cell F2 of tab 1, providing a list of frequencies as a fraction of the sample rate.

Column B of tab 2 has a title “a” for amplitude in the first cell, and the each value is the reciprocal of the amplitude computed in column C of the first tab.

Once you have the table in tab 2 written correctly, export that tab as a comma-separated value (CSV) file. The file Equalizer.csv is also included with this program so that you need only run the FIR design program to go through the example.

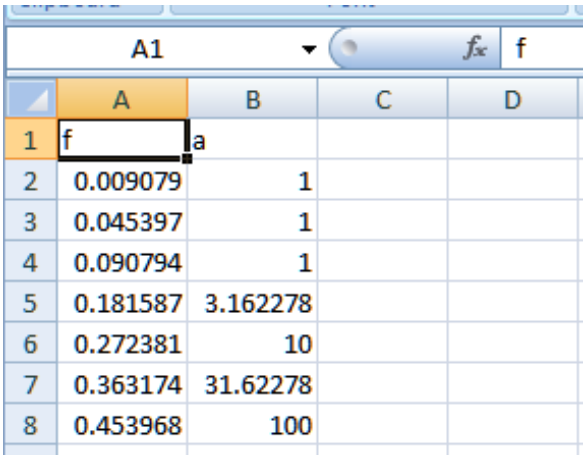


Figure 17. Equalizer.xlsx Tab 2 of, Also Equalizer.csv

Once you select the “Equalizer with one user-specified frequency response” radio button in the dialogue of Figure 3, the dialogue of Figure 16 appears. You can click the shaded ellipsis (“…”) to browse your computer for an input file. The program expects this to be a CSV or other text file with two columns, with one-row headings; it will ignore the first row. The first column must be frequencies as a fraction of the sample rate and all must be zero to 0.5. The second column must be the desired equalizer amplitude response of the FIR filter as amplitude (not decibels); the maximum amplitude does not need to be normalized. All amplitudes must be nonnegative.

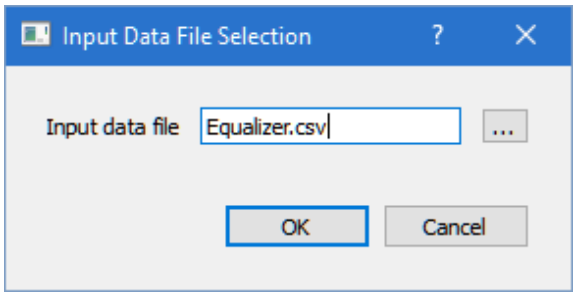


Figure 16. Equalizer Input File Selection Dialogue

The program then reads the input data, normalizes it, and uses a bicubic spline function to interpolate it into a smooth curve for use with the FIR filter design engine. The end condition at zero frequency is zero slope; the end condition at the highest specified frequency is “free,” e.g. the second derivative is zero.

The interpolated frequency response on a dB plot is presented, full-screen, with a prompt on whether to proceed. If there is a bad data point or the data has gaps or irregularities that force the interpolated frequency response to have undesired artifacts like overshoots or bulges, you can abort the design and adjust the inputs and try again. The frequency response for the example of Equalizer.xlsx is shown in Figure 18.

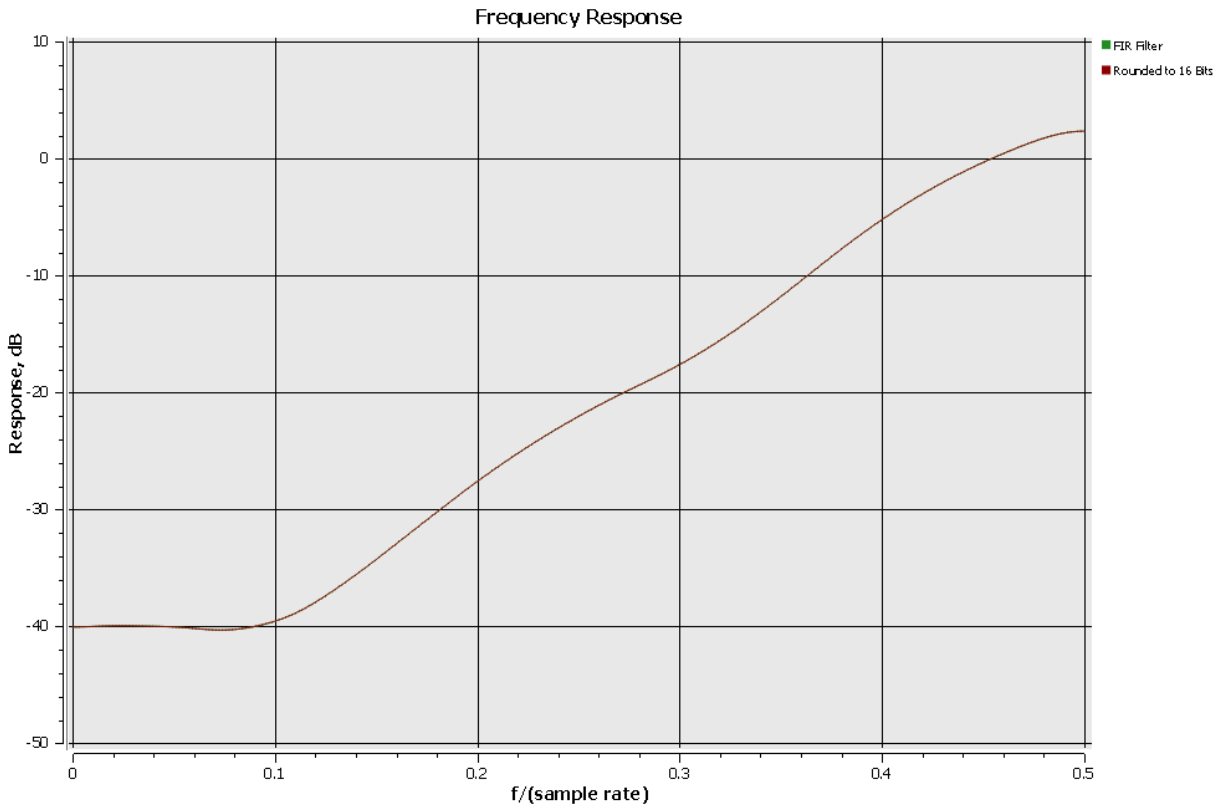


Figure 18. Equalizer Example Frequency Response

3.6 FIR Filter in a Digital Quadrature Demodulator

The problem solved by a digital quadrature demodulator is when a signal of bandwidth B is prepared for sampling at I.F. by an analog filter that has acceptable stopband attenuation beyond a total bandwidth of W, but for practical analog filters $W \gg B$, and high performance is required for signal accuracy, e.g., the application requires

- Linear phase,
- Amplitude accuracy ensured by accurate control of low passband ripple,
- High stopband attenuation, and
- Complex data rate for processing that does not exceed the Nyquist requirement for complex signals of B by an excessive degree.

Latency is half the filter length. The filter length for most designs is about $2.5/T$ where T is the transition bandwidth a a fraction of the sample rate, so the latency is approximately

$$\langle \text{Latency, ms} \rangle \approx \frac{1.3}{\langle \text{Transition bandwidth, KHz} \rangle} \ . \tag{3.4}$$

The combination of high performance, high accuracy, linear phase, and latency on the order of that given in (3.4) are achieved using FIR filters.

In this Section, when discussion complex signals, the signal bandwidth B is aliased to baseband as a complex signal, so that the FIR decimation filter passband is half this bandwidth. We use $B_{COMPLEX}$ below when appropriate to minimize confusion with the Sections on real signal filters.

Note that this approach may show better accuracy and performance than the complex signal obtained by using a Hilbert transformer to generate an imaginary part for a real data stream, because with a quadrature demodulator the real and imaginary parts are outputs from the same filter.

A quadrature demodulator is a dual mixer that downconverts a signal to baseband with an oscillator with two outputs, 90 degrees apart in phase, to produce a complex data stream. This means that the sequence of values from the beat oscillator is $\{\ldots,1,-j,-1,-j,\ldots\}$ so that the action of the mixer in Figure 19 is actually a multiplexer into two channels, one for the real data and one for the complex data. The clean-up filter provides a complex output a either the input sample rate or, most often, the input sample rate divided by an integer, the decimation ratio.

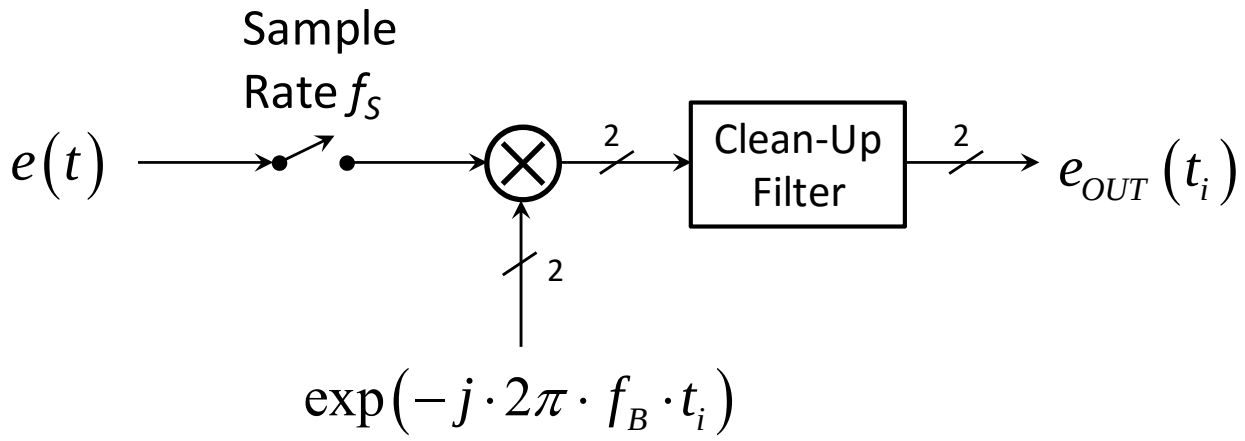


Figure 19. Digital Quadrature Demodulator Block Diagram

The design equations for a signal of bandwidth B with band center frequency f_0 are as follows. We wish to sample the real data so that the center frequency aliases to either positive or negative half-Nyquist so that the quadrature demodulator shifts exactly $\frac{\pi}{2}$ or 90 degrees between samples to put the complex signal at baseband or Nyquist, and the clean-up filter can be a simple decimation filter. The quadrature demodulator and clean-up filter are combined to form a multiplexer.

The sample rate to alias the band center frequency to half Nyquist, midway between 0 and Nyquist, or $\pm f_S/4$ must follow the rule

$$f_S = \frac{4 \cdot f_0}{4 \cdot k \pm 1} \tag{3.5}$$

where the upper sign aliases the signal band into positive frequencies and the lower sign aliases the signal band into negative frequencies; see Figure 20.

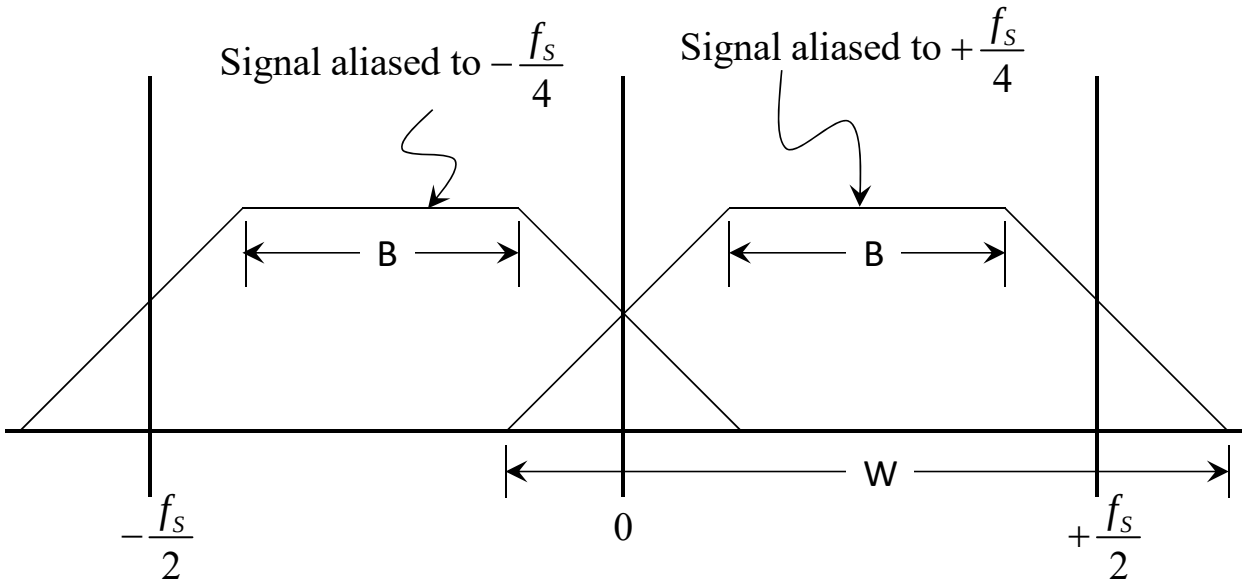


Figure 20. Aliasing Diagram For Undersampled Real Data

In Figure 20 the signal bandwidth is B, and W is the bandwidth of the pre-sampling clean-up filter to the required attenuation for the application. The Nyquist sampling theory limit for avoiding aliasing unwanted signals into the data band is

$$f_S \geq B + W \tag{3.6}$$

with equality only when the band center is plus or minus half-Nyquist. Note that $B + W \gg 2B$, the Nyquist sample rate requirement for real signals.

The quadrature demodulator shifts by 90 degrees each sample and thus puts the signal at either baseband or Nyquist, and presents the real signal with multiplication by the sequence $\{\ldots, 1, -j, -1, -1, \ldots\}$ so that it can be implemented as a multiplexer, allowing the clean-up filter to be designed as a decimation filter – and in fact the output data need only be computed at the decimated rate. Note that when $W \gg B$ as is often the case in practice, equation (3.6) shows that the output data stream may be decimated by a factor D where

$$D < \frac{B + W}{B}. \tag{3.7}$$

The frequency-shifted signal before clean-up and decimation is shown in Figure 21.

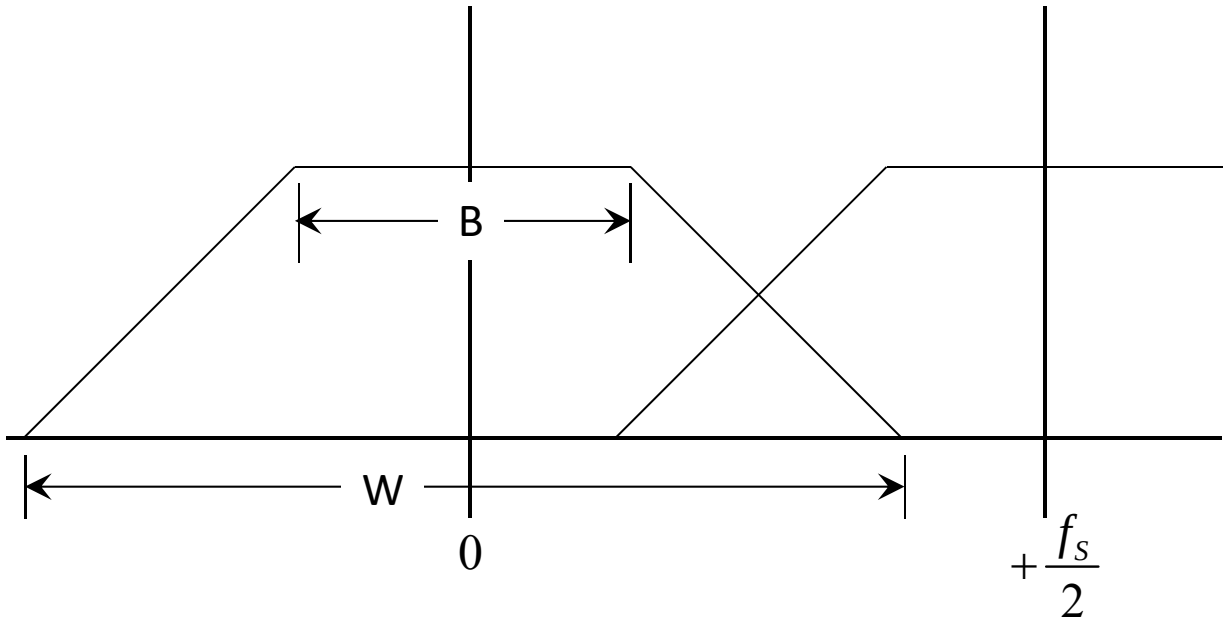


Figure 21. Frequency-Shifted Signal Before Clean-Up

Note that in Figure 21, the signal bandwidth B is centered about zero, with half the signal band as positive frequencies and half as negative frequencies, which are not ambiguous with a complex signal. Figure 21 shows the signal band shifted to baseband; shifting to Nyquist is similar. If there is a DC component in the analog signal, it will be shifted to plus or minus half Nyquist, which will be in the stopband of the decimation filter when $D > 2$.

The block diagram of the atual implementation of a digital quadrature demodulator is shown in Figure 22. The real signal is antialias filtered and, conceptually, a sampler-multiplexer feeds two FIFO registers that provide data to two FIR filters derived from a real decimation filter as explained above. The FIR filter output rate is the decimation rate, and the output is complex data. Actual implementation may use multiple parallel FIR filters or other architectures.

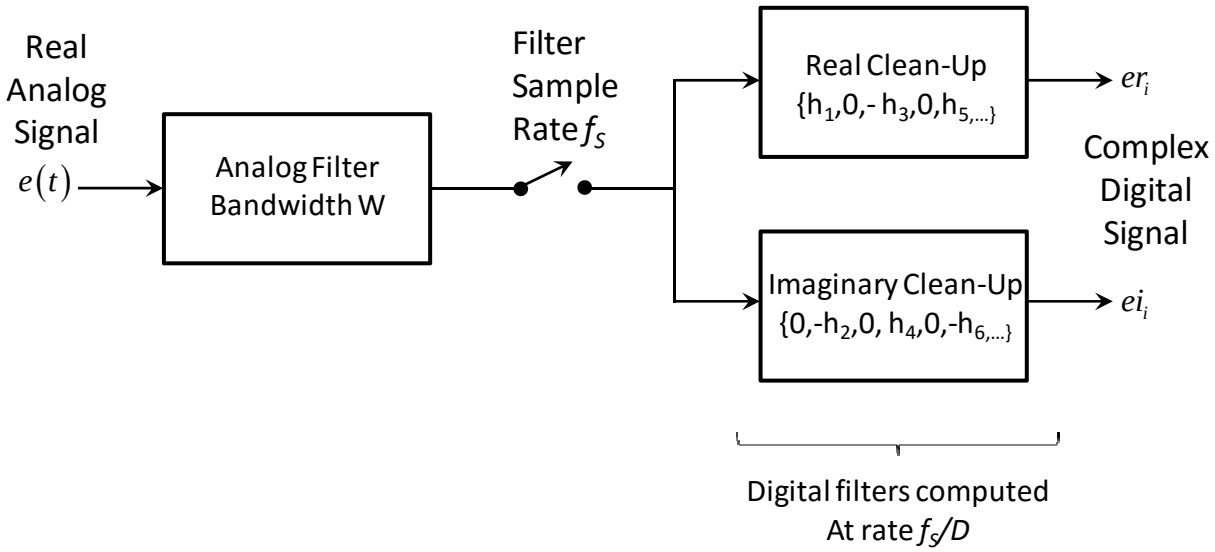


Figure 22. Implementation Configuration of Digital Quadrature Demodulator

The specifications on the decimation filter for the dialogue of Figure 11 are given in Table 3.

Table 3. Decimation Filter Requirements for Digital Quadrature Demodulator

Parameter	Requirement
Peak-to-Peak passband ripple, dB	Application requirement for accuracy; try 0.1 dB
Minimum Stopband Attenuation, dB	Application requirement for accuracy; try 45 dB or more
Passband bandwidth	$\geq \frac{B_{COMPLEX}}{2}$ (equality preferred)
Decimation Ratio	$D \leq \frac{f_S}{B_{COMPLEX}}$

3.7 Output Files

Each full program run will result in the seven files of Table 4. The comma-separated variable (CSV) format is selected for data output to provide machine readability by other programs and spreadsheets.

Table 4. Output Files

File Name	Purpose
yyyymmdd_hhmm_Filter_Weights.csv	Double Precision filter weights
yyyymmdd_hhmm_Frequency_Response.csv	Frequency response table
yyyymmdd_hhmm_Hex_Weights.csv	Weights rounded to specified word length, scaling, and other data needed for implementation
yyyymmdd_hhmm_<Function>_Log.txt	Text run log; primary use is access to classical ASCII design output, also for debugging. The <Function> is the user option of filter design type and will be one of **Optimization**, **Classical_FIR_Design**, or **Equalizer_FIR_Design**.
plot_1X.png	Principal frequency response in dB, frequency 0 to half sample rate
plot_2X.png	Expanded frequency response in dB, frequency minus half sample rate to plus half sample rate
plot_3X.png	Zoomed back, three cycles, in dB, frequency minus 1.5 times sample rate to plus 1.5 times sample rate
plot_1XL.png	Principal linear frequency response, frequency 0 to half sample rate
plot_2XL.png	Expanded linear frequency response, frequency minus half sample rate to plus half sample rate
plot_3XL.png	Zoomed back, three cycles, linear frequency response, frequency -1.5 times sample rate to +1.5 times sample rate

The Hex Weights output file includes the filter weights before and after rounding, the scale factor that makes the largest weight the largest possible number for the specified number of bits, and the scaled filter weights in decimal integers and in hexadecimal format, e.g. everything you need to implement the filter and scale its output. This file is written only when the user elects to find filter weights that are rounded to integers.

When, in the first one-time-per-run dialogue of Figure 1, the “Run test cases and exit” radio button is selected, the string in the log file name be “NEWFIR_Test,” the plot PNG files will not be generated, and the truncation of weights to a binary integer is not done. The filter designs used as examples in Rabiner and Gold, the book referred to in Section 2.1, are computed one at a time and their classical text outputs are provided in the text window. This is primarily a debug or verification function.

A verbose log, including a classical FIR filter design output as shown in Figure 8 for each filter design, including an abbreviated form for the weights truncated to a specified binary word length, is in the text window. The text window can be saved using the “Save” or “Save As” options in the File menu. The file

will be plain text, with the default file name “FIR_Design.exe output” with no extent; you can specify the file name including an extent of “.txt”. This file will be substantially the same as yyyymmdd_hhmm_<Function>_Log.txt and need not be saved.

The file names of the CSV files and the log file begin with the date and time so that they will remain until deleted by the user, but the plots in the PNG file names do not change so the files will be overwritten each time a run finishes, unless the files are renamed or moved.

The frequency response in dB from 0 to half the sample rate is useful in lowpass, highpass, bandpass, and equalizer filters. The zero-centered frequency responses are useful in understanding any issues that arise with the use of an even or odd number of weights; see Section 4.2. The linear frequency response plots are useful in understanding differentiators, Hilbert transformers, and some equalizers, and for user information in the event that issues arise from a design with an odd number of weights; again see Section 4.2.

4 Technical Details

Linear phase convolution digital filters have an impulse response that is a sequence of the weights at the sample rate. The weights are symmetrical about a center point to achieve the linear phase property. This means that the frequency response can be posed as a real function, possibly with a constant phase shift of 90 degrees, referenced in time about that center point, which conveys the property of linear phase – the time delay of the filter is half its duration for all frequencies. The filter impulse responses are even for bandpass filters, and odd about the center point for differentiators and Hilbert transformers. Expressing the frequency response as a Fourier series referenced in time about the center point and accounting for these four configurations means that there are four types of linear-phase FIR filters.

4.1 The Four Types of FIR Filters

The frequency response of a FIR filter is expressed algebraically as

$$\begin{aligned} H_0(f) &= \sum_{n=0}^{N-1} h(n) \cdot \exp\left(-j \cdot 2\pi \cdot n \cdot f \cdot T_{SAMP}\right) \\ &= \exp\left(-j \cdot 2\pi \cdot \frac{N-1}{2} \cdot f \cdot T_{SAMP}\right) \cdot H(f) \end{aligned} \quad . \tag{4.1}$$

The complex term represents a time delay to the center of the filter and $H(f)$ is a real function that gives the frequency response,

$$H(f)=\begin{cases} w_0+2\cdot\sum_{i=1}^{\frac{N-1}{2}} w_i\cdot\cos\left(2\pi\cdot i\cdot f\cdot T_{SAMP}\right), N \text{ odd, h(*) even} \\ 2\cdot\sum_{i=1}^{\frac{N}{2}} w_i\cdot\cos\left(2\pi\cdot\left(i-\frac{1}{2}\right)\cdot f\cdot T_{SAMP}\right), N \text{ even, h(*) even} \\ j\cdot\left(w_0+2\cdot\sum_{i=1}^{\frac{N-1}{2}} w_i\cdot\sin\left(2\pi\cdot i\cdot f\cdot T_{SAMP}\right)\right), N \text{ odd, h(*) odd} \\ j\cdot\left(2\cdot\sum_{i=1}^{\frac{N}{2}} w_i\cdot\sin\left(2\pi\cdot\left(i-\frac{1}{2}\right)\cdot f\cdot T_{SAMP}\right)\right), N \text{ even, h(*) odd} \end{cases} \quad (4.2)$$

The fact that the frequency response is a result of two factors, a time delay and a real (or imaginary) frequency response, means that the transfer function is described as linear phase. In the paragraphs below, we will drop the factor of j for odd transfer functions and omit w_0 for odd transfer functions because w_0 is always zero when the transfer function is odd.

4.1.1 Type 1: Symmetrical Impulse Response, Odd Order

The impulse response is even about the center point so the frequency response is given in terms of a cosine series,

$$H_{TYPE\_1}(f)=w_0+2\cdot\sum_{i=1}^{\frac{N-1}{2}} w_i\cdot\cos\left(2\pi\cdot i\cdot f\cdot T_{SAMP}\right). \quad (4.3)$$

Here, w_0 is the center weight, and the weights w_i are numbered sequentially away from the center. T_{SAMP} is the sampling time, or one over the sample rate.

4.1.2 Type 2: Symmetrical Impulse Response, Even Order

Again, the impulse response is even about the center point so the frequency response is given in terms of a cosine series, but the center point is midway between two data points, so the frequency response is given by

$$H_{TYPE\_2}(f)=2\cdot\sum_{i=1}^{\frac{N}{2}} w_i\cdot\cos\left(\pi\cdot(2i-1)\cdot f\cdot T_{SAMP}\right). \quad (4.4)$$

4.1.3 Type 3: Anti-Symmetrical Impulse Response, Odd Order

The impulse response is odd about the center sample so the frequency response is given in terms of a sine series,

$$H_{TYPE\_3}(f) = 2 \cdot \sum_{i=1}^{\frac{N-1}{2}} w_i \cdot \sin(2\pi \cdot i \cdot f \cdot T_{SAMP}). \quad (4.5)$$

There is no zero term for the center sample because the center weight is always zero.

4.1.4 Type 4: Anti-Symmetrical Impulse Response, Even Order

The impulse response is odd about the center point so the frequency response is given in terms of a sine series, but the center point is midway between two sample points, so the frequency response is given by

$$H_{TYPE\_4}(f) = 2 \cdot \sum_{i=1}^{\frac{N}{2}} w_i \cdot \sin(\pi \cdot (2i-1) \cdot f \cdot T_{SAMP}). \quad (4.6)$$

4.2 Properties and Limitations of the Four Types of Frequency Responses

The principal constraint that provide the properties of linear phase (weights symmetrical or antisymmetrical about the center of the filter) give us the Fourier transform equations for the frequency response that we see above as equations (4.3), (4.4), (4.5) and (4.6). These properties come with the price that precludes certain types of filters in each of the three transfer functions other than Type 1 because the transfer function is zero at either the origin or at Nyquist; the Nyquist frequency is half the sample rate.

The fundamental properties of these frequency responses as given are

- Type 1 and Type 3, the odd orders, have a frequency response that is periodic in frequency with a period of the sample rate.
- Type 2 and Type 4, the even orders, have a frequency response that is periodic in frequency with a period of *twice* the sample rate, and is zero at Nyquist frequency.
- Type 1 and Type 2, the filters that are even about the center point, have a frequency response that is even about zero frequency.
- Type 3 and Type 4, the filters that are odd about the center point, have a frequency response that is odd about zero frequency.

The characteristics of each filter about Nyquist are summarized in Table 5 below. Note that when the number of weights is odd, the filter frequency response has the same gross characteristic, remaining a Type 1 or Type 3, but when the number of weights is even, the gross filter response characteristic about Nyquist is different, a Type 2 becoming a Type 4 and a Type 4 becoming a Type 2. This is because offsetting the frequency response by Nyquist, or half the sample rate, alternates the sign of the filter weights. When the number of weights is odd, the center point of the filter falls on a weight and the gross nature of the frequency response does not change. When the number of weights is even, the center point of the filter falls midway between two weights, so a filter that is even about its center point at zero frequency is odd about its center point at Nyquist, and vice versa.

Table 5. Frequency Responses Shifted by Nyquist Frequency

Type	Frequency response about Nyquist: $H\left(f+\frac{1}{2\cdot T_{SAMP}}\right)$	Remarks
1: Odd Order, Even Filter	$w_0+2\cdot\sum_{i=1}^{\frac{N-1}{2}}(-1)^i\cdot w_i\cdot\cos\left(2\pi\cdot i\cdot f\cdot T_{SAMP}\right)$	Still looks like a Type 1 about Nyquist frequency
2: Even Order, Even Filter	$2\cdot\sum_{i=1}^{\frac{N}{2}}(-1)^i\, w_i\cdot\sin\left(\pi\cdot(2i-1)\cdot f\cdot T_{SAMP}\right)$	Looks like a Type 4 about Nyquist frequency; odd about Nyquist
3: Odd Order, Odd Filter	$2\cdot\sum_{i=1}^{\frac{N-1}{2}}(-1)^i\cdot w_i\cdot\sin\left(2\pi\cdot i\cdot f\cdot T_{SAMP}\right)$	Still looks like a Type 3 about Nyquist frequency; odd about Nyquist
4: Even Order, Odd Filter	$-2\cdot\sum_{i=1}^{\frac{N}{2}}(-1)^i\cdot w_i\cdot\cos\left(\pi\cdot(2i-1)\cdot f\cdot T_{SAMP}\right)$	Looks like a Type 2 about Nyquist frequency; even about Nyquist

Note that even order filters switch apparent types when the frequency responses are centered at Nyquist instead of zero. These characteristics may force even or odd order for some filter requirements. For example, a high-pass filter with even impulse response must have an odd number of weights because the frequency response is always zero at Nyquist, which conflicts with the high-pass requirement.

If the filter weights are even about the center point, as they are with lowpass, highpass and bandpass filters or equalizers, the frequency response may be nonzero at Nyquist, which is the highest unambiguous frequency and is half the sample rate.

Below we have examples of frequency responses for the four filter types:

- The frequency response of a Type 1, a high-pass with an odd number of weights is shown in Figure 23,
- The frequency response of a Type 2, a low-pass with an even number of weights, is shown in Figure 24,
- The frequency response of a Type 3, a differentiator with an odd number of weights, is shown in Figure 25,
- The frequency response of a Type 4, a differential with an even number of weights is shown in Figure 26, and, as a case with a peculiar issue,
- The frequency response of another Type 4, a Hilbert transformer with an even number of weights is shown in Figure 27.

An interesting special case is the Hilbert transformer shown in Figure 27, which would ideally have a discontinuity at zero frequency, so it's low band edge cannot be zero.

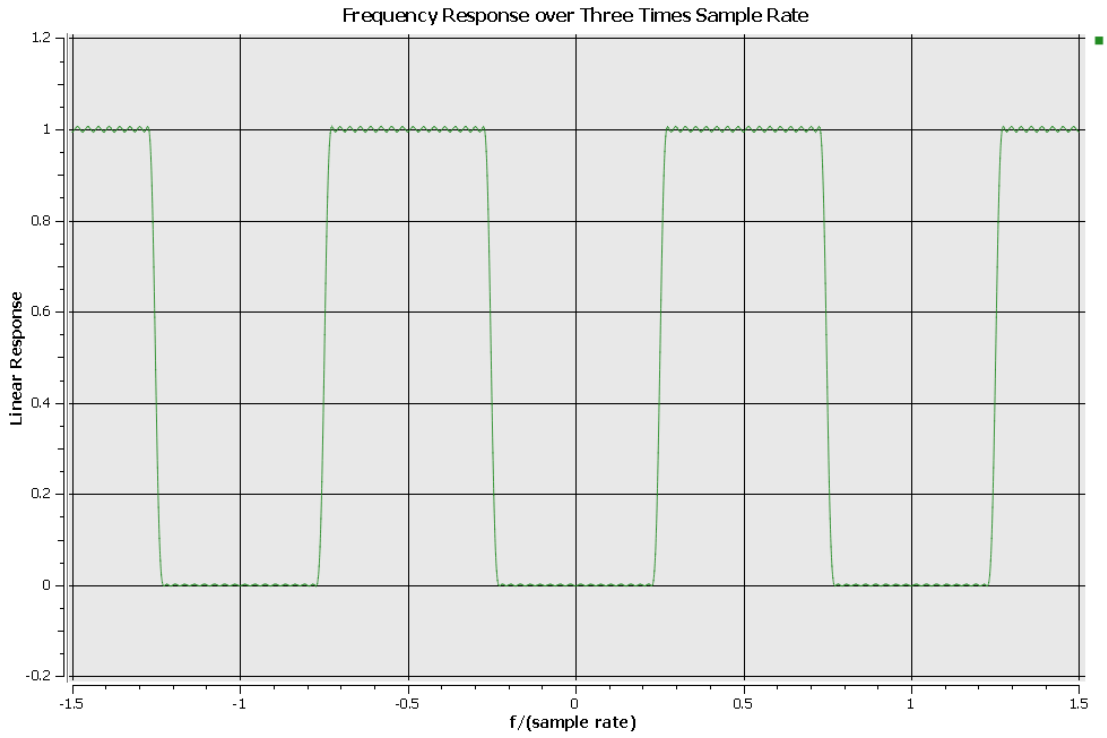


Figure 23. Type 1: High Pass, Odd Number of Weights

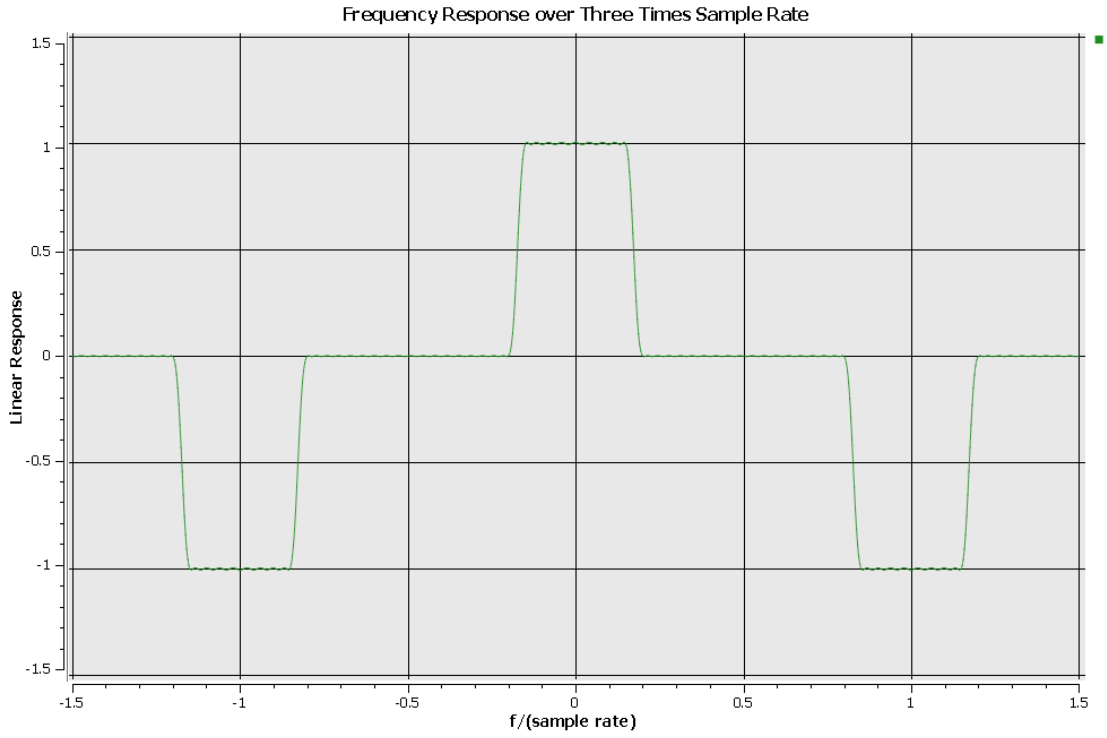


Figure 24. Type 2: High Pass with Even Number of Weights

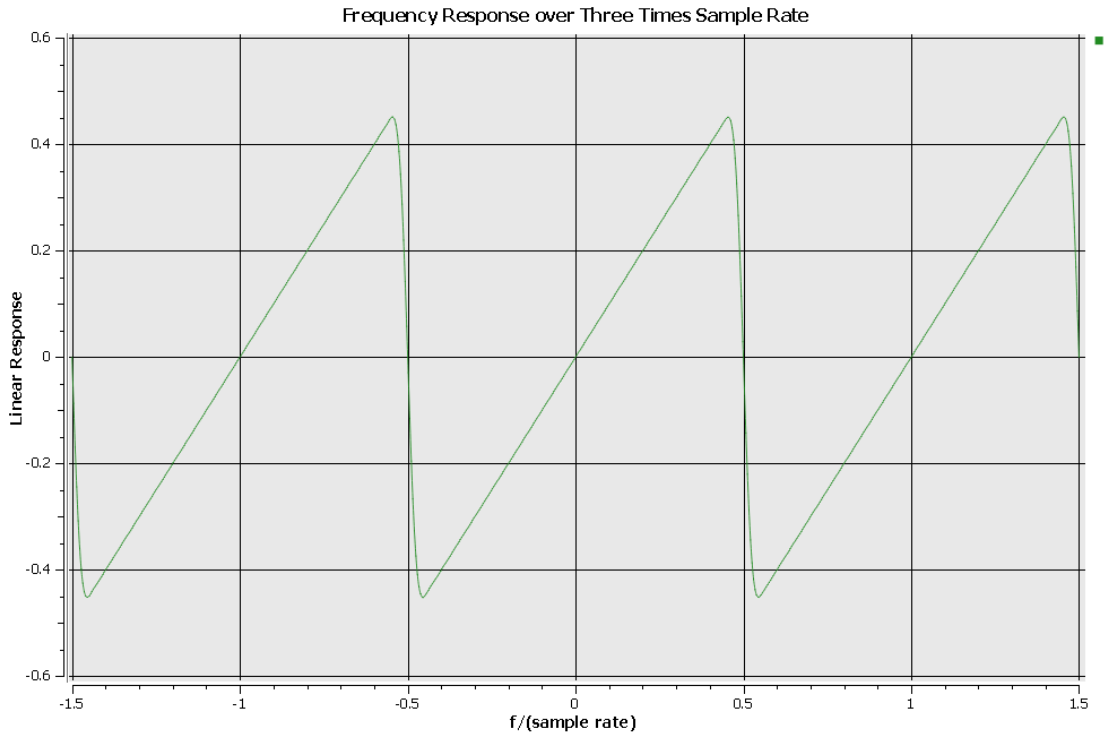


Figure 25. Type 3: Differentiator with Odd Number of Weights (Not Recommended)

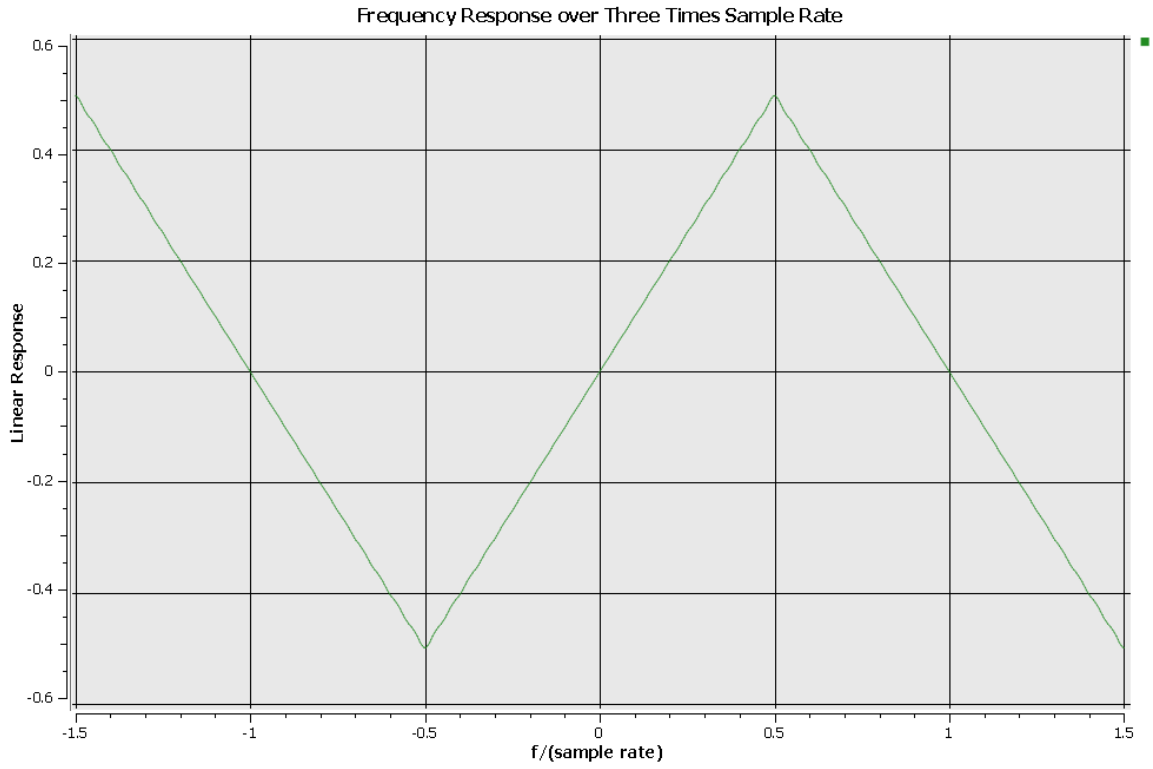


Figure 26. Type 4: Differentiator with Even Number of Weights

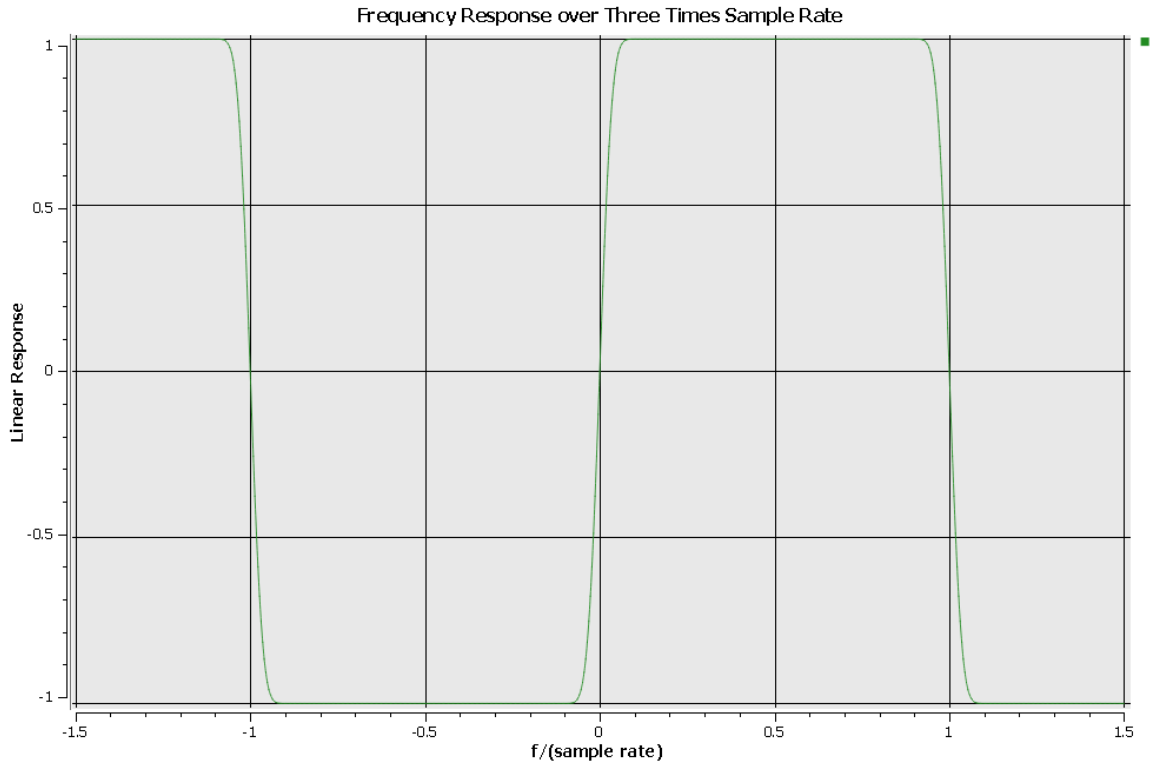


Figure 27. Type 4: Hilbert Transformer with Even Number of Weights, Transition at Zero Frequency

The issues with each type and the remedy or work-around for each situation is summarized in Table 6 below.

Table 6. Limitations and Solutions for Each Filter Type

Filter	Issue	Solution
Type 1	None	Not Applicable
Type 2	Zero at Nyquist	Use odd number of weights (Type 1) for high-pass filters. For bandpass, limit top frequency of passband to just short of Nyquist (NOTE 1).
Type 3	Zero at both zero frequency and Nyquist	Limit upper end of differentiator and Hilbert transformer (NOTE 1). Use even number of weights (Type 4) in differentiators and Hilbert transformers. (NOTE 2)
Type 4	Zero at zero frequency	If appropriate, limit top frequency of differentiator to just under Nyquist to reduce the required number of weights to meet requirements. (NOTE 2)

NOTE 1: The width of the gap between the top of the passband and Nyquist may drive the necessary number of weights to meet requirements.

NOTE 2: In Hilbert transformers, always limit the lower limit of the band because the desired frequency response of 1.0 has a sign change at zero frequency. A "don't-care" region about zero frequency is necessary for the design to succeed.

4.3 Ripple, Filter Deviation, and Error Weighting in the Stopband

The classical FIR filter design engine performs its calculation in the frequency domain, designing a polynomial to achieve a Chebychev fit to the ideal or specified frequency response in the passbands and stopbands. The polynomial is weighted according to a specified weighting function for each passband and stopband, and the weighted deviations of the polynomial from the ideal frequency response are all equal when the design is completed. The achieved ripple is inversely proportional to the weighting applied.

In our program, the passband is always weighted by 1.0. In lowpass, highpass, and bandpass filters, the stopband is weighted to achieve the specified stopband attenuation *and* the specified passband ripple *when the number of weights is exactly sufficient to achieve specifications*; note that since the number of weights is always an integer this is usually not possible exactly, so the next larger allowed number of weights is used in the achieved designs. Thus the ripple and stopband achieved in the achieved designs meets or slightly exceeds specifications.

In design procedures that fix the number of weights and varies another parameter, near-exact conformance to the specifications is usually achieved. Performance as a function of any given parameter is not always well-behaved or even monotonic, so small deviations that emerge during the design process are handled in the program so that the result presented to the user always *meets or slightly exceeds* specifications.

Internally the program designs a function that is equiripple everywhere bit that meets specifications for varying ripple requirements by applying linear weights to the error in different bands. IN our program, the passband ripple is always 1.0 and the stopband weighting is computed from the specifications and applied internally by the program. A derivation for the equations for this weighting follows.

Nearly all practical applications have requirements for passband deviation specified as peak-to-peak ripple specification. If the linear deviation is denoted by δ_{PASS} and the frequency response is denoted by $f_{PASSBAND}$, the ripple is

$$\left.\begin{aligned} R_{dB} &= 20 \cdot \log 10\left(R_{LINEAR}\right) \\ R_{LINEAR} &= \frac{f_{PASSBAND} + \delta_{PASS}}{f_{PASSBAND} - \delta_{PASS}} \end{aligned}\right\} . \tag{4.7}$$

Note that the low-pass, high-pass, differentiator, and Hilbert transformer,

$$f_{PASSBAND} = 1 \tag{4.8}$$

but the equalizer and multi-band filters have user-defined passbands that may be any number from -1 to +1.

Also, note that equation (4.7) is valid only when $f_{PASSBAND} > \delta_{PASS}$ and is meaningful only if

$$f_{PASSBAND} \gg \delta_{PASS} . \tag{4.9}$$

Since the differentiator, Hilbert transformer, and equalizer have no stopband, the generality of the passband applies only to the classical inputs used when there is more than one passband or stopband.

The "average" filter response in the passband, in the log domain – e.g. on a plot in decibels, is the geometric mean of the ripple peaks above and below the "average" filter response,

$$f_{GM} = \sqrt{\left(f_{PASSBAND} + \delta_{PASS} \right) \cdot \left(f_{PASSBAND} - \delta_{PASS} \right)} . \tag{4.10}$$

The required stopband attenuation is attenuation with respect to f_{GM} so that the stopband attenuation on a decibels plot will be at the required level when compared to the mean between the ripple peaks in decibels,

$$\left.\begin{aligned} S_{LINEAR} &= \frac{f_{GM}}{\delta_{STOP}} \\ S_{dB} &= \left| 20 \cdot \log 10 \left(\frac{\delta_{STOP}}{f_{GM}} \right) \right| \end{aligned}\right\} . \tag{4.11}$$

Note that the program computes S_{dB} as positive; the program takes the absolute value of user input here to avoid confusion. Internally, S_{LINEAR} is a large positive number, the reciprocal of the sidelobe suppression level relative to the geometric mean of the passband ripple peaks.

Since the Chebychev ripple in each band is inversely proportional to the error weighting in that band, the stopband weighting is

$$W_{STOP} = \frac{\delta_{PASS}}{\delta_{STOP}} . \tag{4.12}$$

Thus we have two nonlinear equations in two unknowns, driven by two specified inputs. The program converts the input requirements in dB to linear,

$$\left.\begin{aligned} R_{LINEAR} &= 10^{\frac{R_{dB}}{20}} \\ S_{LINEAR} &= 10^{-\frac{S_{dB}}{20}} \end{aligned}\right\} \tag{4.13}$$

and computes the linear domain deltas from equations (4.7), (4.10) and (4.11) and uses equation (4.12) to compute the stopband weighting.

We compute the linear domain deltas as follows. Solving equation (4.7) for δ_{PASS} gives us

$$\delta_{PASS} = \frac{R_{LINEAR} - 1}{R_{LINEAR} + 1} \cdot f_{PASSBAND} \tag{4.14}$$

while from equations (4.10) and (4.11) we have

$$\begin{aligned} \delta_{STOP} &= \frac{f_{GM}}{S_{LINEAR}} \\ &= \frac{\sqrt{\left(f_{PASSBAND} + \delta_{PASS}\right) \cdot \left(f_{PASSBAND} - \delta_{PASS}\right)}}{S_{LINEAR}} \end{aligned} . \tag{4.15}$$

We compute δ_{PASS} using equation (4.14) and δ_{STOP} using equation (4.15), then we compute the weighting W using (4.12).

For reference, the result from rolling up equations (4.12) through (4.15) for a passband objective function of 1.0 is

$$W_{STOP}\left[f_{PASSBAND} = 1\right] = \frac{S_{LINEAR} \cdot \left(R_{LINEAR} - 1\right)}{2\sqrt{R_{LINEAR}}} . \tag{4.16}$$

Note that for very low passband ripple,

$$W_{STOP}\left[f_{PASSBAND} = 1\right] \approx \frac{\ln(10)}{40} \cdot R_{dB} \cdot S_{LINEAR}, \quad R_{dB} \ll 1 . \tag{4.17}$$

When the legacy inputs are used as in Section 3.4.1 and you want to use weightings that give you specific passband ripple and stopband attenuation combinations, you can use equation (4.13) to get the linear ratios and equation (4.16) to get the stopband weighting. Or, you can get the program to do the computation using a dummy run: use a design for a low-pass using the desired ripple and stopband requirements to get the stopband weighting from the legacy output as given in Figure 8, and use that for the stopbands. Use the reciprocal of the passband objective function for the weight in each passband,

$$W_{PASS} = \frac{1}{f_{PASSBAND}} . \tag{4.18}$$

5 The Source Code

The source code is in Fortran 2003, e.g. the added features of later versions of Fortran (complex functions, quad precision, parallel threads and cluster support, etc.) are not needed here. The GUI

portion is compiler-specific to Absoft Fortran compilers (http://absoft.com) which are available for Windows, Mac and Linux environments. My intention is to eventually release the source code under a BSD or GPL license. Issues that must be resolved prior to release include, but are not limited to, the following:

- The program is, at this point, personal working code, with commented-out dead ends and previous versions of some algorithms, some duplicate code, etc. – definitely is in a not-ready-for-prime-time state.
- The FIR design engine was adapted from the original Parks, McClellan and Rabiner program as published in an IEEE journal and in the book by Rabiner & Gold published by Prentice-Hall. As such, the authors, the IEEE, or Prentice-Hall may have some rights to the code for the engine.
- The user interface, using pop-up dialogues, X-Y plots, and a text window, are in compiler-dependent code. Although the user interface is all in a specific, single module, and the earliest versions used the command line and GNU PlPlot, and an embryonic command-line interface does exist in the source code that may be included using conditional compilation, the command-line user interface has not been maintained past the first revision. To retain current functionality, an entirely new command-line user interface module is required.

When these and any other issues that present themselves are resolved, I will release the source code on my web site or through GNU or Sourceforge.

My current planning is to retain the Absoft AWE user interface and not support a command-line interface myself. Dr. John McClellan is aware of this program and my intent to eventually release the source code, and I have sent prior versions of the source code to him, and I do not expect any objections from him or the other authors of the IEEE papers and Prentice-Hall book. The age of the IEEE and Prentice-Hall publications and the degree of revision necessary for structuring this program will probably make satisfaction of IEEE and Prentice-Hall interests a matter of proper attribution. This leaves a few e-mails and code clean-up with a clean compile using GNU gfortran (except for the Absoft-specific GUI module calls) as the primary considerations.