

LOW-PASS FILTER IMPLEMENTATION: PRELAB*

Mark Butala

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License [†]

1 IIR Filter Design Methods

1.1 Overview

Implementing the narrow-band LPF using an IIR filter is probably the most difficult option to design but the most straightforward to implement. One reason for the design difficulty stems from the fact that in order to get such a sharp response poles close to the unit circle are needed. These poles can then drift outside the unit circle and the system can then become unstable when finite precision effects are added. Also, perfectly linear phase (constant group delay) cannot be realized using IIR filtering.

There are several approaches to designing an approximately linear phase IIR filter. For example, an IIR filter could be run on blocks of samples in both the forward and reverse direction and the results of each block added together; using the filter in both directions would cancel the nonlinear phase response of the filter. Also, iterative design methods exist to design filters that simultaneously minimize errors in magnitude and group delay. Yet another approach is to design an IIR filter which approximates the desired magnitude response (e.g., an elliptic filter using the `ellip` command in MATLAB) and then design an IIR all-pass filter which compensates for the nonlinear phase. This last approach is the one we will examine here.

1.2 MATLAB Filter Design Toolbox

The MATLAB Filter Design (FD) Toolbox contains algorithms used for optimal or near optimal design of filters subject to various constraints. You can view a description of the toolbox and the functions it contains at this link¹. The FD Toolbox will be installed on the white Dell machine nearest the door in the ECE 320 lab.

1.3 Using the Filter Design Toolbox

Although it is possible to design a very good LPF magnitude response using an elliptic filter, we do not have the advantage with the `ellip` command of being able to constrain the poles away from the unit circle to prevent instability. Fortunately, the FD Toolbox provides a command called `iirlpnormc` which allows us to keep the poles within a circle of a specified radius. Note that this command implements a least- p 'th

*Version 2.6: Aug 19, 2005 11:05 am GMT-5

[†]<http://creativecommons.org/licenses/by/1.0>

¹<http://www.mathworks.com/access/helpdesk/help/toolbox/filterdesign/filterdesign.shtml>

algorithm. The term least- p 'th signifies that the algorithm attempts to minimize a L_p -norm error. In the case of the magnitude response, the L_p -norm error is given as

$$|H|_p = \frac{1}{2\pi} \int_{-\pi}^{\pi} |H(\omega) - H_d(\omega)|^p W(\omega) d\omega^{\frac{1}{p}} \quad (1)$$

where H is the actual frequency response, H_d is the desired response, and W is some weighting function. Most of the time the weighting function used is one which equals 1 over the passband and stop-band and 0 in the transition band. The role of W in the above equation is similar to that used in FIR filter design (`remez`). The relative weights in the stop-band and pass-band are set by W and control the relative magnitude of the ripples in these bands. Note that minimizing the L_2 -norm is equivalent to minimizing the RMS error in the magnitude. In contrast, the L_∞ -norm is equivalent to minimizing the maximum error over the frequencies of interest (why?). In this lab we are concerned with minimizing the L_∞ -norm. Of course, we cannot use infinity in any of our computations so using a large number (e.g. 128) must suffice.

Once our magnitude response has been selected, we need to perform group delay equalization to yield approximately constant group delay. This can be done using the `iirgrpdelay` command in the FD Toolbox. Again, note that a least- p th algorithm is used and that we can constrain the radius of the poles. The resulting all-pass filter can be connected in series with the nonlinear phase low-pass filter created with `iirlpnormc` to complete the entire system.

The FD Toolbox can also aid in analyzing quantization effects. We suggest using `FDATool`, a convenient GUI for interacting with the FD Toolbox, to carry out the analysis. See the internet documentation for more information on the functions available. Of course, you may choose to do this by scaling and rounding as you have done in previous labs. Note that even though MATLAB uses high-precision arithmetic you may find that for long IIR filters MATLAB has difficulty rendering frequency responses, etc. Thus, you may find it useful to design a filter that has half the passband ripple, half the stop-band attenuation, etc. and implement it twice in your code to meet the specification.

Note that `FDATool` and the filter design toolbox (`qfilt` function) can be used to analyze quantization effects on various filter structures, as well as on the FFT. The quantization parameters can be chosen and optimized in `FDATool`. Also, `FDATool` (with or without the filter design toolbox) can compute correct scaling to avoid overflow.

1.4 Implementation Structures

There are several ways to implement an IIR filter. One of these, a cascade of second-order systems, we have already seen. An alternative is placing these second-order sections in parallel. Another common implementation is a lattice structure (see any standard DSP textbook), which tends to be more resistant to finite word-length effects and may be more computationally efficient. To examine your choices, as a starting point you should examine the MATLAB functions listed as "Linear System Transformations" when you type `help signal` at the command prompt (does not require the FD Toolbox).

One of the difficult aspects of an IIR lattice is that although the lattice coefficients are in the interval (-1,1), the internals of the lattice can grow to be prohibitively large. To compensate for this, an m-file (`latcfilt.m`) has been created that performs normalization after each lattice section to prevent overflow. If you are interested in exploring a lattice implementation you may want to copy this m-file to your own directory and modify it to suit your needs. Note that there are comments within the file to indicate where you might add checks for overflow conditions.

We suggest the use of `FDATool` and `dfilt` for structure transformations. The function `dfilt` works also without the Filter Design Toolbox. It is also useful for evaluating cascade or parallel connections of sub-filters. The MATLAB command `fvtool` can be used to quickly evaluate frequency response of various filter structures.

Many extremely efficient structures for IIR filter implementations exist. Two of special note are the following:

- All-pass filter implementations with N multiplies for an order N filter (instead of $2N$ multiplies for a cascade realization). These are structurally all-pass, meaning that they remain all-pass even when their coefficients are quantized. (S. Mitra, Digital Signal Processing, a Computer Based Approach, 2nd Ed., pp. 378-382).
- Parallel all-pass Realization of IIR transfer functions with N multiplies for an N 'th order filter. (S. Mitra, Digital Signal Processing, a Computer Based Approach, 2nd Ed., pp. 401-405, and Sec. 9.9, pp. 629-633).

1.5 Questions

1. Generate an elliptic LPF using the command: `[b,a]=ellip(4,.5,10,.1)`; Using MATLAB commands, generate a cascaded second-order system implementation and a lattice implementation (don't worry about normalization if you don't want) of this system and compare their advantages and disadvantages - especially as they relate to implementation on the C5400.
2. How close to the unit circle are the poles of the system from question 1? Does this concern you? Explore how much the poles moved in the 2 implementations of part 1.
3. Use the `grpdelay` command to view the group delay for the filter in the passband. Is it approximately linear?
4. Why does minimizing the L_1 -norm equate to minimizing the L_∞ -norm maximum error over a given frequency range?

1.6 Simulation

Simulate the IIR system in MATLAB. Compute the response of the system with appropriate test inputs. Make sure to include side-effects due to finite precision in your simulation.

2 Multi-rate/Multi-stage

2.1 Reading Exercise

Read through the following resources:

- "Optimum FIR Digital Filter Implementations for Decimation, Interpolation, and Narrow-Band Filtering," by Crochiere and Rabiner. This is colloquial paper on the topic of multi-stage filter implementation. The paper is available here².
- Course notes on multi-stage filter implementation by Prof. Mark Fowler³ from Binghamton University. The notes are available here⁴.

2.2 Design Exercises

Given the filter specification given in the filter specification⁵, answer the following questions:

- What is the maximum decimation factor that can be used?
- What is the average number of MACs per input sample that are required for a single stage implementation?
- What are the appropriate decimation and interpolation factors for a a two stage implementation?

²<http://www.ews.uiuc.edu/~ece320/crochiererabiner.pdf>

³<http://www.ee.binghamton.edu/fowler/>

⁴<http://www.ews.uiuc.edu/~ece320/multistage.pdf>

⁵"Low-Pass Filter Implementation: Filter Specification" <<http://cnx.org/content/m11056/latest/>>

- What are the appropriate pass-band and stop-band frequencies and maximum ripple for the overall filter at each stage? Your answer will demonstrate that the use of multiple filter stages along with multi-rate signal processing can achieve a overall filter of lower order than just a single stage filter.
- Estimate the filter order for each stage. We recommend using the MATLAB command `remezord`. This algorithm frequently underestimates the filter order needed, but gives you a good starting point. Verify that the filter specifications are met, i.e. pass-band and stop-band ripple and pass-band and stop-band band edge locations. Do this by passing the arguments returned by `remezord` to the MATLAB command `remez`. Observe the frequency response of the system described by the filter coefficients returned by `remez` using the MATLAB command `freqz`. If the specifications are not met, increase the order of the filter until the specifications are met.
- Determine the average number of MACs per sample for the two-stage implementation. Which is more efficient, the single stage approach or the multistage approach?

2.3 Matlab Simulation

Using your results from the previous part, simulate the two-stage multi-rate filter in MATLAB. Plot the frequency response of each stage's filter using `freqz` and determine the overall frequency response of your multi-rate system to verify that it meets the specifications. Since there is not a command for directly finding the frequency response plot of a multi-rate system in MATLAB, you will have to be a bit creative.

2.4 Additional Questions (optional, but for your benefit)

- Does it make a difference in which order the two decimations are done in a two-stage implementation?
- Could / would you add additional stages? Why or why not?
- Are quantization effects more or less pronounced in the multi-stage implementation compared to a direct implementation? Why or why not?

3 Fourier-Based Filtering Methods

It is possible to perform linear convolution quickly using the FFT. This idea allows for the efficient implementation of a FIR filter when the number of filter coefficients and the length of the input sequences are large.

3.1 Questions

- Read Lecture 49 of the ECE 310 Course Notes on "Block Convolution." This lecture provides an excellent overview of two methods for efficiently performing convolution using the FFT: "Overlap and Add" and "Overlap and Save." For a more in depth treatment of these methods, refer to Discrete-Time Signal Processing by Alan Oppenheim and Ronald Schaffer.
- Simulate both an (1) overlap and add and an (2) overlap and save filtering implementation in MATLAB. Your simulations should work for any choice of an FIR filter. The filter length M and block length L should be variable parameters.
- Verify that your simulated systems are working properly by comparing their performance with a direct FIR implementation. Test using several FIR filter designs and appropriate test inputs.
- Derive expressions for the amount of computation (in terms of multiply accumulates) required per input sample for both the overlap and add and overlap and save implementations. Plot the computation per sample as a function of the input block length (for a particular filter size M) for both schemes. Is there a value of M for which the Direct FIR is always more efficient? Derive an expression for the optimal block size L in terms of the filter length M for both implementations.

- In the DSP implementation, the input sequence is purely real. The values of the imaginary components are all set to zero. We can speed up the implementations further by exploiting the symmetry properties of the Fourier transform. These properties are stated as follows:

$$\text{DFT}(\Re x(n)) = \text{Even}(X(\omega)) \quad (2)$$

$$\text{DFT}(j\Im x(n)) = \text{Odd}(X(\omega)) \quad (3)$$

Using these properties, determine how to get two FFT's for the price of one. Implement this scheme in MATLAB, and verify that the operation is correct.

- Design a FIR filter that meets the filter specification given in the filter specification⁶. Lecture 38 of the ECE 310 notes on "Parks-McClellan" might be a good reference here. Design an efficient implementation of this filter using the methods you explored above. The MATLAB commands `remezord` and `remez` may be of great help. Simulate this implementation in MATLAB, programming in such a way that you can easily convert your MATLAB simulation to assembly. Find the number of computations per input for your method.
- What are the benefits and trade-offs of using the Fourier-based method in terms of accuracy of the filter specification, finite precision errors, and computational expense? Compare with the IIR and multi-rate filter implementations.

Be prepared to show all the necessary plots and MATLAB simulations as well as answers to all of the questions posed above to your T.A. as your prelab.

3.2 Implementation Issues

Due to the limitations of the core file, it is not possible to take in more than 64 input samples from the A/D converter at a time (unless the core file is rewritten to accomplish this task). Therefore, when implementing a Fourier-based filter, you should use the C skeleton from Lab 4 to perform the FFT on a large block of samples. All of your filtering operations (i.e., the multiplications of DFT samples, the additions of the overlap, the discarding of samples) and function calls must be performed in assembly. You will be graded on the number of cycles per input sample based on the portion of code in your assembly routine.

You should use the `fft.asm` routine provided in Lab 4 to perform the forward and reverse FFT's. You should study this file to determine how it works. If you need to change the length of the FFT, you will first need to change the relevant parameters in your assembly file (i.e., `N`, `K_FFT_SIZE`, `K_LOGN`, and other variables). You will also need to change the following parameters in the FFT file:

```
K_TWID_TBL_SIZE
K_TWID_IDX_3
```

`K_TWID_TBL_SIZE` is the size of the twiddle tables (how long should these be for a given FFT length?) and `K_TWID_IDX_3` is the amount by which the program increments through the twiddle table during at the third stage of the FFT. What is this increment for a given `N`? Is `fft.asm` a decimation in time or decimation in frequency algorithm?

You will also need a modified twiddle table when you change the length of the FFT to use `fft.asm` as written. For a length 1024 FFT, the twiddle tables are length 512 each. `TWIDDLE1` is a table of sine values from zero to π , and `TWIDDLE2` is a table of cosine values from $\frac{\pi}{2}$ to $\frac{3\pi}{2}$. The support for the cosine and sine is different because `fft.asm` code uses the fact that $\sin(-\theta) = -(\sin(\theta))$ when performing computations. If you want a length 64 FFT, you will need to "decimate" the twiddle table to length 32, or in other words, only keep one out of every 16 lines in the twiddle tables and discard the rest. We will provide a MATLAB function, `edit_twiddle.m` for this purpose. The function call in this example would be:

⁶"Low-Pass Filter Implementation: Filter Specification" <<http://cnx.org/content/m11056/latest/>>

```
edit_twiddle('TWIDDLE1','new_twiddle1',16)
```

You should verify that the new twiddle tables you generate indeed have 32 elements. To perform an inverse FFT, you can use the standard FFT algorithm and then appropriately scale and shift the outputs. Lecture 43 of the ECE 310 notes on the Discrete Fourier Transform suggests how this may be done (Property 3 of “Properties of the DFT”).