Lab 5. FIR & IIR Filters in MATLAB

Filter Design

The goal of filtering is to perform frequency-dependent alteration of a signal. A simple design specification for a filter might be to remove noise above a certain cutoff frequency. A more complete specification might call for a specific amount of passband ripple (Rp, in decibels), stopband attenuation (Rs, in decibels), or transition width (Wp - Ws, in hertz). A precise specification might ask to achieve the performance goals with the minimum filter order, call for an arbitrary magnitude response, or require an FIR filter.

IIR filter design methods differ from FIR primarily in how performance is specified. For loosely specified requirements, as in the previous paragraph, a Butterworth filter is often sufficient. More rigorous filter requirements can be met with so-called Chebyshev and elliptic filters. The Signal Processing Toolbox order selection functions estimate the minimum filter order that meets a given set of requirements. To meet specifications with more rigid constraints, such as linear phase or arbitrary response, it is best to use direct IIR methods such as the Yule-Walker method or FIR methods.

Filter Configurations

First, recall that when dealing with sampled signals, we can normalize the frequencies to the Nyquist frequency (defined as one-half the sampling frequency). All the filter design functions in the Signal Processing Toolbox operate with normalized frequencies, so that they do not require the system sampling rate as an extra input argument. The normalized frequency is always in the interval $0 \leq f \leq 1$. For example, with a 1000 Hz sampling frequency, 300 Hz is $300/500 = 0.6$. To convert normalized frequency to angular frequency around the unit circle, multiply by $\pi$. To convert normalized frequency back to Hertz, multiply by half the sample frequency. So we have the following:

- Lowpass filters remove high frequencies (near 1)
- Highpass filters remove low frequencies (near 0)
- Bandpass filters pass a specified range of frequencies between 0 & 1
- Bandstop filters remove a specified range of frequencies between 0 & 1

©2012GM
To calculate the normalized frequency (nf):

\[ fs = 1e4; \]
\[ f = 400; \]
\[ nf = 400/(fs/2) \]

Filter Specifications in Matlab
- Wp - Passband cutoff frequencies (normalized)
- Ws - Stopband cutoff frequencies (normalized)
- Rp - Passband ripple: deviation from maximum gain (dB) in the passband
- Rs - Stopband attenuation: deviation from 0 gain (dB) in the stopband

Digital filter specifications are often given in terms of the loss function (in dB),

\[ A(\omega) = -20\log_{10}|G(e^{j\omega})| \]

As per page 490 in the Text, digital filter magnitude specifications are shown as follows. (Band Pass, High Pass specifications follow a similar form).

**Figure 1: Filter Specifications in Text**

In MATLAB, the Filter Design and Analysis Tool (FDATool) shows these specifications are shown below (Figure 2). We follow the MATLAB specifications for this Lab.

As you can imagine, the order of the filter will increase with more stringent specifications: decrease in the width of the transition band, or more stringent requirements on \( A_{\text{pass}} \) or \( A_{\text{stop}} \), that is a narrower transition band.

**In this Lab, do items 1 \(
\rightarrow\) 16. Then go on to answer Q9.**

1. fdatool

2. Design a Lowpass, FIR Eqiripple filter, Minimum Order, Fs=1000hz, Fpass=60, Fstop=200, Apass 1dB, Astop 80dB. Design Filter

3. Invoke “Full View Analysis” (the tenth icon on the top, from the left) and you get the Filter Visualization Tool (FVT). This is much easier to work with. Have the magnitude response showing. Do View \(\rightarrow\) Zoom In, that is check ‘Zoom In’. Place the mouse cursor on the amplitude response curve, click it and see the
Q1: Was the filter designed as per specifications? That is, confirm dB at Fpass, and at Fstop. Are they what was expected. If not, how would you fix it? Once you have fixed it, what penalty are you paying? Note that the filter you designed is 17th order but has 18 coefficients.

Q2: What size are the jumps? Are the points where the jumps occur have anything to do with the points where the amplitude frequency response changes? If so, why is that happening? Discounting the jumps, is the phase linear with frequency? Finally, does the phase exceed $-\pi$? Should it be doing that? Is unwrapped phase being plotted?

Q3: What is the value of this delay as seen in FVT? Is that consistent with your measurement on the Phase Response graph?

Q4: What is its value? Verify the group delay value by finding the slope from the phase response graph. Assume $h[n]$ symmetric about the origin. Therefore $H(e^{j\omega})$ is real, i.e. zero phase. Then we have,

$$h[n] \leftrightarrow H(e^{j\omega})$$

Now if we shift $h[n]$ by N units to the right, to make it causal, we get:

$$h[n - N] \leftrightarrow H(e^{j\omega})e^{-j\omega N}$$

$$\phi(\omega) = -\omega N$$

That is what you have for the filter that you generated.
7. Look at the filter impulse response. (8th icon from the right in FDA Tool).

**Q5:** About what point is it symmetric? (We will relate this value with group and phase delay in class).

8. Look at i (4th icon from the right) - Filter information. Or in FVTool, *Analysis → FilterInformation.*

This gives you an idea about cost for building the filter.

**Q6:** Is the filter stable? Why?

9. Look at the pole-zero plot.

**Q7:** How many poles? How many zeros? The fact that all poles are at zero makes sense for FIR filters. Why?

10. Look at the filter coefficients. This is what gets implemented with varying degrees of precision.

11. Export the filter parameters (Num) to your workspace. Do size(Num) to obtain the size of your FIR filter. Do *freqz(Num,1).*

12. In the command window, do *sptool.* In sptool, Import Num to sptool. Change sampling frequency from the default 1, to 1000.

13. Now under SPTool: startup.spt, view the signal (Recall that Num is just the values of the 18 point FIR filter; i.e. the FIR impulse response). Under Spectra, use “Create” and do a 1024 point fft.

14. Under Options → Magnitude → Linear. Observe the difference between observing on linear and log scales. Do you see the equiripples in the stopband?

15. Redesign your filter with Fstop = 100. Import coefficients to sptool, look at the spectrum under sptool, using linear scale. Now do you see the “equiripple” in the passband in the magnitude response? And in the log scale, do you see the “equiripple” in the attenuation band?

16. Now in fdatool, for the same specs as the first FIR filter, design an IIR Butterworth filter. (Match exactly the passband and stopbands). Look at the amplitude response. Look at the phase response. Linear phase? Look at the impulse response. (Note that it is NOT symmetric). Look at the Filter information. Did it meet the specs? How many poles and zeros?

**Q8:** Does it have any properties dissimilar to that of the FIR filter? Explain.

---

**IIR Filter Design**

The primary advantage of IIR filters over FIR filters is that they typically meet a given set of specifications with a much lower filter order than a corresponding FIR filter. This has the obvious implementation advantages.

IIR filters have nonlinear phase (they cannot have linear phase; they might almost have linear phase). However, if data processing is to be performed off line, then the effects of the nonlinear phase can be eliminated. So let us assume that the entire input data sequence is available prior to filtering. This allows for a noncausal, zero-phase filtering approach (via the *filtfilt* function), which eliminates the nonlinear phase distortion of an IIR filter.

The classical IIR filters - Butterworth, Chebyshev Types I and II, elliptic, and Bessel - all approximate the ideal “brick wall” filter in different ways. The Signal Processing Toolbox provides functions to create all these types of IIR filters in both the analog and digital domains (except Bessel, for which only the analog case
is supported), and in lowpass, highpass, bandpass, and bandstop configurations. For most filter types, you can also find the lowest filter order that fits a given filter specification in terms of passband ripple, stopband attenuation, and the transition band widths.

Try the following:

doc filtfilt

Open the file noisyC.m (it is in sg01). (You can do “which noisyC.m” in the Command line and find out where it is). Run it. You hear it and variables n, sw and swn are saved in your workspace.

You can also generate the noisy sine wave like so:

```
fs = 1e4;
t = 0:1/fs:5;
sw = sin(2*pi*262.62*t);
n = 0.1*randn(size(sw));
swn = sw + n;
```

Create a Butterworth filter of order 2 with a cutoff at 400 Hz:

```
[b a] = butter(2, 400/(fs/2));
```

Note the nonlinear phase:

```
figure, freqz(b,a)
```

Now create a filter with `filter`:

```
y = filter(b,a,swn);
figure, plot(t,y), axis([0 0.04 -1.1 1.1]), title('Using filter()')
soundsc(y,1e4)
```

Filter with `filtfilt`:

```
y2=filtfilt(b,a,swn);
figure, plot(t,y2), axis([0 0.04 -1.1 1.1]), title('Using filtfilt()')
soundsc(y2,1e4)
```

Q9: Do you note the difference? Comment.

(There are no further questions. You may choose to go through the rest of the Lab. if you wish. There are some very interesting analog filter design topics.)

**IIR Filter Types**

- Butterworth filter
  Provides the best Taylor series approximation to the ideal lowpass filter response at analog frequencies \( \Omega = 0 \) and \( \Omega = \infty \); for any order \( n \), the magnitude squared response has \( 2n - 1 \) zero derivatives (that is, it is maximally flat) at these locations. Response is monotonic overall, decreasing smoothly from \( \Omega = 0 \) to \( \Omega = \infty \).
- **Chebyshev Type I filter**
  Minimizes the absolute difference between the ideal and the actual frequency response over the entire passband by using an equal ripple in the passband. Stopband response is maximally flat. The transition from passband to stopband is more rapid than for the Butterworth filter.

- **Chebyshev Type II filter**
  Minimizes the absolute difference between the ideal and the actual frequency response over the entire stopband by using an equal ripple in the stopband. Passband response is maximally flat. The stopband does not approach zero as quickly as the type I filter (and does not approach zero at all for even-valued filter order $n$). The absence of ripple in the passband, however, is often an important advantage.

- **Elliptic filter**
  Equiripple in both the passband and stopband. Generally meets filter requirements with the lowest order of any supported filter type. Given a filter order $n$, passband ripple, and stopband ripple, elliptic filters minimize transition width.

- **Bessel filter**
  Analog lowpass filters have maximally flat group delay at zero frequency and retain nearly constant group delay across the entire passband. Filtered signals therefore maintain their waveform in the passband. Digital Bessel filters, however, do not have this maximally flat property, and are not supported by Matlab. Generally require a higher filter order than other filters for satisfactory stopband attenuation.

**Analog Prototyping**

The principal design technique for digital IIR filters supported by the Signal Processing Toolbox is based on the conversion of classical lowpass analog filters to their digital equivalents. The technique involves three steps:

1. Find an analog lowpass filter with cutoff frequency of 1 and translate this prototype filter to the desired band configuration.

2. Transform the filter to the digital domain.

3. Discretize the filter.

The toolbox provides functions for each of these steps:

<table>
<thead>
<tr>
<th>Design Task</th>
<th>Available Low-Level Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog lowpass prototype</td>
<td>buttap, cheb1ap, besselap, ellipap, cheb2ap</td>
</tr>
<tr>
<td>Frequency transformation</td>
<td>lp2lp, lp2hp, lp2bp, lp2bs</td>
</tr>
<tr>
<td>Discretization</td>
<td>bilinear, impinvar</td>
</tr>
</tbody>
</table>

Alternatively, the `butter`, `cheby1`, `cheby2`, `ellip`, and `besself` functions perform *all steps of the filter design* and the `buttord`, `cheb1ord`, `cheb2ord`, and `ellipord` functions perform minimum order computations for IIR filters. These functions are sufficient for many design problems, and the lower level functions are generally not needed. If you do have an application where you need to transform the band edges of an analog filter, or discretize a rational transfer function, the low-level functions allow you to do so.
Example: Bandpass Filter

To demonstrate, we will use various methods to design a bandpass filter that passes middle C at 262.62 Hz. The filter has the following specifications:

- **Order:** 
  \( n \leq 10 \)

- **Sampling frequency:**
  \( fs = 10,000 \text{ Hz} \)

- **Normalized passband cutoff frequencies:**
  \( W_p = \left[ \frac{150}{2fs} \ 350/(fs/2) \right] = [0.03 \ 0.07] \)

- **Normalized stopband cutoff frequencies:**
  \( W_s = \left[ \frac{100}{2fs} \ 400/(fs/2) \right] = [0.02 \ 0.08] \)

- **Passband ripple:**
  \( R_p = 0.5 \text{ dB} \)

- **Stopband attenuation:**
  \( R_s = 20 \text{ dB} \)

**Butterworth Method: Analog Prototype**

For the Butterworth method, we will follow the three steps of analog prototyping at a low level.

1. **Lowpass analog prototype**

   \([z \ p \ k] = \text{buttap}(5)\); 
   
   Outputs \( z, p, \) and \( k \) contain the zeros, poles, and gain of a lowpass analog filter of order 5 with a cutoff frequency of 1 rad/s. The bandpass filter will have order 10 (2 cutoffs).

2. **Frequency Transformation**

   Use the \( \text{lp2bp} \) function to transform the lowpass prototype to a bandpass analog filter. First, convert to state-space form so the \( \text{lp2bp} \) function can accept it:

   \([A \ B \ C \ D] = \text{zp2ss}(z,p,k)\);

   Now, find the bandwidth and center frequency, and call \( \text{lp2bp} \):

   \[ u1 = 0.03*2*\pi; \]  
   \[ u2 = 0.07*2*\pi; \]  
   \[ Bw = u2-u1; \]  
   \[ Wo = \text{sqrt}(u1*u2); \]  
   \[ [A \ Bt \ Ct \ Dt] = \text{lp2bp}(A,B,C,D,\text{Wo},Bw); \]

   Finally, calculate the frequency response and plot its magnitude:

   \([b \ a] = \text{ss2tf}(At,Bt,Ct,Dt);\)  
   \[ w = \text{linspace}(0.01,1,500)*2*\pi; \]  
   \[ h = \text{freqs}(b,a,w); \]
semilogy(w/2/pi,abs(h))
grid on
xlabel('Normalized Frequency');
ylabel('Analog Response')

Explain the function of each line of code in this section individually.

**Butterworth Method: Discretization**

3. Discretization

- **Impulse invariance**
  The `impinvar` function creates a digital filter whose impulse response is sampled from the analog impulse response.

  \[
  [bz1 \ az1] = \text{impinvar}(b,a,fs);
  \]
  returns transfer function coefficients \(bz1\) and \(az1\) of the digital filter whose impulse response is equal to the impulse response of the analog filter with transfer function coefficients \(b\) and \(a\), using a sampling frequency of \(fs\). There should be negligible frequency content above \(fs/2\), since high frequency content would be aliased into lower bands when sampling. The `impinvar` function works for some lowpass and bandpass filters, but it is not appropriate for highpass and bandstop filters.

- **Bilinear transformation**
  The bilinear transformation is a nonlinear mapping of the continuous \(s\)-domain to the discrete \(z\)-domain given by \(z = (k + s)/(k - s)\). The transfer function \(H(s)\) of the ideal analog filter is mapped to the transfer function \(H(z)\) of a corresponding discrete filter.

  \[
  [bz2 \ az2] = \text{bilinear}(b,a,fs);
  \]
  returns transfer function coefficients \(bz2\) and \(az2\) of the digital filter that is the bilinear transformation of the analog filter with transfer function coefficients \(b\) and \(a\), using a sampling frequency of \(fs\). By default, the frequency warping constant \(k\) is set equal to twice the sampling frequency. Band edges can vary slightly from the specified cutoffs because of the nonlinear nature of the transformation.

Omitted the following:
Chebyshev Type I Method
Elliptic Method

**Direct IIR Design**

Direct design methods find a filter based on specifications in the discrete domain. Unlike the analog prototyping method, direct designs are not constrained to the standard lowpass, highpass, bandpass, or bandstop configurations. Filters with an arbitrary, perhaps multiband, frequency response are possible.

The `yulewalk` function designs IIR digital filters by fitting a specified frequency response. It finds the inverse FFT of the ideal desired power spectrum and solves modified Yule-Walker equations using the resulting autocorrelation function samples. The statement
[b a] = yulewalk(n,f,m)
returns vectors b and a containing the n + 1 numerator and denominator coefficients of the order n IIR filter whose frequency-magnitude characteristics approximate those given in vectors f and m. f is a vector of frequency points ranging from 0 to 1, where 1 represents the Nyquist frequency. m is a vector containing the desired magnitude response at the points in f. f and m can describe any piecewise linear magnitude response, including a multiband response.

Note that yulewalk does not accept phase information, and no statements are made about the optimality of the resulting filter.

Example: Arbitrary Response IIR Filter

Problem
Design a 12th-order bandstop filter fitting the following frequency response data:
f = [0 0.3 0.3 0.4 0.4 0.5 0.5 0.6 0.6 1];
m = [1 1 0.5 0.5 0 0 0.5 0.5 1 1];

Solution
Compute the Yule-Walker fit and frequency response:
[b a] = yulewalk(12,f,m);
[h w] = freqz(b,a,128);

Overplot ideal and designed responses:
plot(f,m,'r--',w/pi,abs(h),'b-')
legend('Ideal','Yule-Walker Design')
title('Comparison of Response Magnitudes')

Try:
edit directstop
directstop(10)
directstop(20)
directstop(30)

Exercise
1. Create a 3-second signal containing the sum of three tones of equal amplitude, each entering with 1-second delays. The signal should
   • Have a sampling frequency of 8192 Hz
   • Contain a 220 Hz tone for 0 < t < 3
   • Contain a 300 Hz tone for 1 < t < 3
   • Contain a 440 Hz tone for 2 < t < 3

   View the signal before proceeding.

   Now create a noise signal to add to the tone signal.
n = randn(size(t));

Create the sum of the tone and the noise.
y = x + n;

2. Design an arbitrary response IIR filter to remove the 300 Hz component from the three-tone signal with white noise y.
3. Plot the designed response and a Yule-Walker fit together.
4. Filter the signal y with the designed filter.
5. Compare signals and spectra before and after filtering.

Filter Design with SPTool

The Filter Designer in the SPTool allows you to design and edit IIR and FIR filters of various lengths and types, with lowpass, highpass, bandpass, and bandstop, and multiband configurations. To activate the Filter Designer, click either the New button or the Edit button under the Filters list box in SPTool.

Try:
sptool

Filter Design with FDATool

The Filter Design Toolbox works with Matlab and the Signal Processing Toolbox to provide a complete environment for start-to-finish filter design. Its graphical user interface, the FDATool, supports many advanced techniques not available in SPTool.

Use the FDATool to:
• Design filters
• Quantize filters
• Analyze filters
• Modify existing filter designs
• Realize Simulink models of quantized direct form FIR filters
• Perform digital frequency transformations of filters

Try:
fdatool

Filter Analysis with FDATool

The Filter Visualization Tool (FVTool) can be accessed from within the FDATool by choosing the Full View Analysis option in the Analysis menu or by clicking the Full View Analysis button on the toolbar.

The FVTool is dynamically linked to the FDATool, so changes to a filter design automatically update plotted responses for analysis.
Importing Existing Filters

The FDATool can be used to edit and analyze existing filters by importing them from the Matlab workspace. To do so:
1. Select **Import Mode**.
2. Choose the desired **Filter Structure**.
3. Enter the coefficients.
4. Enter the **Sampling Frequency**.
5. Click **Import Filter**.

Exporting from the FDATool

Filters designed in the FDATool can be exported to
- The Matlab Workspace
  (filter coefficients can be vectors or objects)
- The SPTool
  (for capabilities not available in FDATool, such as spectral analysis)
- MAT-files
- Text files
- C header files

To export a filter, choose **File → Export** in the FDATool.

(The material in this lab handout derives principally from the MathWorks training document “MATLAB for Signal Processing”, 2006.
Revised February 16, 2012)