FIR Filters in Matlab

Lab 6. FIR Filter Design in Matlab

You only have to answer Qs.1 → Q9.

Digital filters with finite-duration impulse response (referred to as FIR filters) have both advantages and disadvantages when compared to infinite-duration impulse response (IIR) filters. (Note: You will need *sg01* for some of the functions in this Lab.).

FIR filters have the following primary advantages:
- They can have exactly linear phase.
- They are always stable, even when quantized.
- The design methods are generally linear.
- They can be realized efficiently in hardware.
- The filter startup transients have finite duration.

The primary disadvantage of FIR filters is that they often require a much higher filter order than IIR filters to achieve a given level of performance. Correspondingly, the delay of these filters is often much greater than for an equal performance IIR filter.

FIR Methods

The Signal Processing Toolbox supports a variety of methods for the design of FIR filters.
<table>
<thead>
<tr>
<th>Filter Method</th>
<th>Description</th>
<th>Filter Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windowing</td>
<td>Apply window to truncated inverse Fourier transform of desired filter</td>
<td>fir1, fir2, kaiserord</td>
</tr>
<tr>
<td>Multiband with Transition Bands</td>
<td>Equiripple or least squares approach over frequency subbands</td>
<td>firls, firpm, firpmord</td>
</tr>
<tr>
<td>Constrained Least Squares</td>
<td>Minimize squared integral error over entire frequency range subject to maximum error constraints</td>
<td>fircls, fircls1</td>
</tr>
<tr>
<td>Arbitrary Response</td>
<td>Arbitrary responses, including nonlinear phase and complex filters</td>
<td>cfirpm</td>
</tr>
<tr>
<td>Raised Cosine</td>
<td>Lowpass response with smooth, sinusoidal transition</td>
<td>firrcos</td>
</tr>
</tbody>
</table>

**Impulse Response Revisited**

FIR filters are described by difference equations of the form

\[ y(n) = \sum_{m=0}^{M} b(m) x(n-m) \]  

where the filtered signal \( y(n) \) is just a linear combination of current and previous values of the input signal \( x(n) \). The coefficients \( b(b(m)) \) are the numerator coefficients of the transfer function. The denominator of the transfer function will always be \( a = 1 \). The order of the filter is \( n = \text{length}(b) - 1 \).

If the input signal is the unit impulse \( x = [1 \ 0 \ 0 \ \ldots] \), then the corresponding impulse response \( y(n) \equiv h(n) \) is identical to \( b(n) \):

\[
\begin{align*}
    h(0) &= b(0)x(0) = b(0) \\
    h(1) &= b(0)x(1) + b(1)x(0) = b(1) \\
    h(2) &= b(0)x(2) + b(1)x(1) + b(2)x(0) = b(2) \quad \ldots \text{etc.}
\end{align*}
\]

The FIR filter coefficients are the same as the impulse response.

Try this. Let \( b \) represent the coefficients of the x’s in equation (1).

\[
b = [-1 \ 0 \ 2 \ -3 \ 1]; \ % \ no \ need \ to \ specify \ the \ a \ coefficients
\]

\[
\text{stem}(b) \\ \ \text{figure, impz}(b,1)
\]
Q1. Write the (non-recursive) difference equation using the b coefficients given above. Write the impulse response coefficients \( h[n] \), \( n = 0,1,2... \) that you got by doing \texttt{impz(b,1)}. Do the two agree?

Q2. In this particular case when using the function \texttt{impz}, MATLAB indexes the impulse response coefficients in agreement with what they actually are. When is this not so?

**Linear Phase Filters**

A filter whose impulse response is symmetric or antisymmetric about its midpoint is called a (generalized) linear phase filter. That is,

\[
h[n] \leftrightarrow H(e^{j\omega}) = |H(e^{j\omega})|e^{j\theta(\omega)}, \theta(\omega) = K\omega
\]

(We will show this in class). For such filters, as we see above,

- The phase shift \( \theta(\omega) \) of a filtered signal will vary linearly with frequency \( \omega \). (This translates into the fact that the output signal of such a filter has a pure time delay with respect to the input signal. There is no phase distortion).
- The phase delay \(-\theta(\omega)/\omega\) and group delay \(-d\phi(\omega)/d(\omega)\) will be equal and constant. For an order \( n \) linear phase FIR filter, the phase delay and group delay is \( n/2 \). (Do you see this?)

Except for \texttt{cfirpm}, all the FIR filter design functions in the Signal Processing Toolbox design linear phase filters only.

Try:

```matlab
a = 1;
b = fir1(5,0.5);
fvtool(b,a)
```

This is the classic method of windowed FIR linear-phase filter design. (We will see this in class). Look at the phase delay and the group delay in fvtool.

Q3. For this (5th order) 6-point filter, confirm that the phase \( \theta(\omega) \simeq -2.4868\omega \) as seen in the phase plot of fvtool.

Hence, find the value of the phase delay \(-\theta(\omega)/\omega\) and group delay \(-d\theta/d\omega\).

Is that consistent with what you see in fvtool? Elaborate.

(Note that in \( e^{j\omega n} \), \( \omega \) can be considered to have units of radians if \( n \) is considered dimensionless or units of radians/sample is \( n \) is considered to have units of samples.)
FIR Filter Types

(You can skip over this if you wish and go on to Window Based Design).
The symmetric impulse response of a linear phase filter can have an odd or an even number of points, and can have an odd or even symmetry about the midpoint, leading to four filter types: (we see this in class).

- Type I: Odd length, even symmetry
- Type II: Even length, even symmetry
- Type III: Odd length, odd symmetry
- Type IV: Even length, odd symmetry

Depending on the filter type, certain restrictions apply:

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Filter Order</th>
<th>Symmetry</th>
<th>Response (H(0)) (Nyquist)</th>
<th>Response (H(1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I</td>
<td>Even</td>
<td>(b(k) = b(n + 2 - k), k = 1, ..., n + 1)</td>
<td>No restriction</td>
<td>No restriction</td>
</tr>
<tr>
<td>Type II</td>
<td>Odd</td>
<td>(b(k) = b(n + 2 - k), k = 1, ..., n + 1)</td>
<td>No restriction</td>
<td>(H(1) = 0)</td>
</tr>
<tr>
<td>Type III</td>
<td>Even</td>
<td>(b(k) = -b(n + 2 - k), k = 1, ..., n + 1)</td>
<td>(H(0) = 0)</td>
<td>(H(1) = 0)</td>
</tr>
<tr>
<td>Type IV</td>
<td>Odd</td>
<td>(b(k) = -b(n + 2 - k), k = 1, ..., n + 1)</td>
<td>(H(0) = 0)</td>
<td>No restriction</td>
</tr>
</tbody>
</table>

The functions \(fir1, fir2, firls, firpm, fircls, fircls1\), and \(firrcos\) all design type I and II linear phase FIR filters by default. Both \(firls\) and \(firpm\) design type III and IV linear phase FIR filters given a 'hilbert' or 'differentiator' flag. The function \(cfirpm\) can design any type of linear or nonlinear phase filter.

Because the frequency response of a type II filter is zero at the Nyquist frequency (“high” frequency), \(fir1\) does not design type II highpass and bandstop filters. For odd-valued \(n\) in these cases, \(fir1\) adds 1 to the order and returns a type I filter.

Window-Based Design

Under FIR Methods above, Windowing is the first filter design method. It is the common, classic design method for FIR filters.
Recall that \(H(\omega)\) is periodic. Hence we can write it as a Fourier series. That is,
\[
H(\omega) = \sum_{n=-\infty}^{+\infty} h(n)e^{-jn\omega}. 
\]

\(h[n]\) are the Fourier coefficients that are determined using the inverse DTFT. In general, the impulse response cannot be implemented as a digital filter because it is infinite and non-causal.
So, $h(n)$ is symmetrically truncated (multiplied by a finite, symmetric window) and shifted to make it causal, to create an implementation that gives a linear phase finite impulse response. (Recall that from Fourier series properties, if the function is even, the Fourier coefficients are even and if the function is odd, Fourier coefficients are odd. Hence for $H(e^{j\omega})$ even, like for a low pass filter, $h[n]$ will be even, i.e. symmetric.)

The truncated Fourier series can be shown to be an approximation to the ideal filter, which is “best” in a mean square sense, compared to other approximations of the same length. However, the abrupt truncation leads to overshoot (Gibb’s phenomenon) and ripples in the spectrum. The undesirable effects of truncation can be reduced or eliminated by the use of tapered windows.

The Windowing method does not allow you to design a filter, with explicit amplitude response constraints (like we saw in Lab.5), such as passband ripple or stopband attenuation. It needs to be used iteratively and we stop when design specifications are met.

Try:
```
edit windemo
windemo(20)
windemo(50)
windemo(100)
```

**Q4**: What do the arguments (20, 50, 100) specify?
**Q5**: How does the filter frequency response change with increasing the size of the filter?
**Q6**: Even as you increase the number of filter coefficients that you retain, does the largest overshoot in the magnitude frequency response decrease in size?
**Q7**: Explain, very simply what is happening in the 4 figures that you see in windemo.

**Windowing Functions**

In `windemo` you saw the effects (Gibbs phenomena) on the frequency response of the finite length filter when a (default) rectangular window was imposed. Now we see methods of ridding ourselves of Gibbs phenomena - at a price.

The Signal Processing Toolbox supports a variety of windows commonly used in FIR filter design. (We will see this in class.) Typing
```
help window
```
provides a list of available windows:
Individual window functions take inputs for a window length $n$ and window parameters and return the window $w$ in a column vector of length $n$. (Note: Use $w'$ for array products with row vector impulse responses.) The window function serves as a gateway to the individual functions.

```matlab
alpha=0.25
w = gausswin(64,alpha)
```

and

```matlab
w = window(@gausswin,64,alpha)
```

both return a Gaussian window of length 64 with standard deviation equal to $1/\alpha$.

### Windowing and Spectra

As we saw above, when a signal is truncated in time, high-frequency components are introduced in frequency that are visible in the DFT spectrum. By windowing the truncated signal in the time domain, endpoints are assigned a reduced weight. The effect of this on the DFT is to reduce the height of the side lobes in the spectrum, but increase the width of the main lobe. (That is the penalty we pay). Here is a simple demonstration:

Try:

```matlab
edit windft
windft
```
You see a truncated signal (5 Hz sine wave) and its DFT (Figure 1). (Recall the Fourier transform of a sine wave? Do you see how the DFT is different?) Then, the truncated sine wave is multiplied by a Gaussian window and the DFT is taken of the truncated windowed signal, Figure 2.

![Truncated Signal](image1.png)

![DFT of Truncated Signal](image2.png)

Tuncated, windowed signal and its DFT:

![Truncated, Windowed Signal](image3.png)

![DFT of Truncated, Windowed Signal](image4.png)

Q8. For a continuous-time sine wave, its Fourier transform are a couple of delta functions. For a discrete-time truncated sine wave, how is its spectrum (DFT) different to the idealized delta-function spectrum of a (continuous-time) sine
Q9 How does windowing the truncated sine wave ”improve” the DFT spectrum?

Note: You have a 5 hertz sine wave. If you edit windft you see that you are just looking at it for 1 second. So you see 5 cycles of it in 1 second. Hence you are taking a sine wave, infinitely long, and putting a 1 second rectangular window on it. In the analog domain you have,

\[ x_w(t) = (\sin 2\pi 5t).w_R(t) \]

where \( w_R(t) \) is your 1 second window. In the Fourier domain, you have for the spectrum of the truncated sine wave,

\[
X_w(\Omega) = \left[ \frac{-\pi}{j} \delta(\Omega + \Omega_o) + \frac{\pi}{j} \delta(\Omega - \Omega_o) \right] \frac{\sin \Omega/2}{\Omega/2}
\]

\[
= \frac{-\pi \sin(\Omega + \Omega_o)/2}{j \Omega + \Omega_o/2} + \frac{\pi \sin(\Omega - \Omega_o)/2}{j \Omega - \Omega_o/2}
\]

And as you know, the spectrum of a continuous-time signal (truncated sine wave \( x_w(t) \)) is related to the spectrum of the sampled \( x_w(n) \) signal, as

\[
X_w(e^{j\omega}) = \frac{1}{T} \sum_{k=-\infty}^{+\infty} X_w(j(\Omega - k\Omega_o))
\]

Now you can compare the MATLAB results of the DFT with the FT results that you know.

**Window Visualization Tool**

To “window” a filter, the filter impulse response \( h(n) \) of size N is multiplied by a window of size N. The transition bandwidth of a window-based FIR filter is determined, as we shall see in class later on, by the width of the main lobe of the DFT of the window function. The main lobe width, as you can observe, can be adjusted by changing size of N.

Passband and stopband ripples are determined by the magnitude of the side lobe of the DFT of the window function, and are usually not adjustable by changing the filter order. Ideally, the spectrum of a window should approximate an impulse. The main lobe should be as narrow as possible and the side lobes should contain as little energy as possible.

The Window Visualization Tool (WVTool) allows you to investigate the tradeoffs among different windows and filter orders.
wvtool(windowname(n))
opens WVTool with time and frequency domain plots of the n-length window specified in windowname, which can be any window in the Signal Processing Toolbox. Several windows can be given as input arguments for comparative display.

Try:
wvtool(hamming(64),hann(64),gausswin(64),rectwin(64))
wvtool(hamming(32),kaiser(32,2.5),flattopwin(32))
wvtool(kaiser(32,1),kaiser(32,5),kaiser(32,10))

Q10. Comment on the differences - main lobe width and ripples - between the same size rectangular window and say, the Hamming window.

Window Design and Analysis Tool

The Window Design and Analysis Tool (WinTool) is used in conjunction with the Window Visualization Tool.

- Use WVTool for displaying and comparing existing windows created in the Matlab workspace.
- Use WinTool to interactively design windows with certain specifications and export them to the Matlab workspace.

Most window types satisfy some optimality criterion. Some windows are combinations of simpler windows. For example, the Hann window is the sum of a rectangular and a cosine window, and the Bartlett window is the convolution of two rectangular windows. Other windows emphasize certain desirable features. The Hann window improves high-frequency decay (at the expense of larger peaks in the side lobes). The Hamming window minimizes side lobe peaks (at the expense of slower high-frequency decay). The Kaiser window has a parameter that can be tuned to control side lobe levels. Other windows are based on simple mathematical formulas for easy application. The Hann window is easy to use as a convolution in the frequency domain.

An optimal time-limited window maximizes energy in its spectrum over a given frequency band. In the discrete domain, the Kaiser window gives the best approximation to such an optimal window.
opens WinTool with a default 64-point Hamming window. Try it - experiment with different window designs and export them to the workspace.

**Example: Lowpass Filter**

Consider the ideal, or “brick wall”, digital low-pass filter with a cutoff frequency of $\omega_0$ rad/s. This filter has magnitude 1 at all frequencies less than $\omega_0$, and magnitude 0 at frequencies between $\omega_0$ and $\pi$. Its impulse response sequence $h(n)$ is

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega) e^{j\omega n} d\omega = \frac{1}{2\pi} \int_{-\omega_0}^{\omega_0} e^{j\omega n} d\omega = \frac{\omega_0}{\pi} \text{sinc} \left( \frac{\omega_0}{\pi} n \right)$$

This filter is not implementable since its impulse response is infinite and noncausal. To create a finite-duration impulse response, truncate it by applying a window. Retain the central section of impulse response in the truncation to obtain a linear phase FIR filter. For example, a length 51 filter with a lowpass cutoff frequency $\omega_0$ of $0.4\pi$ rad/s has the impulse response,

$$h = 0.4 \ast \text{sinc}(0.4 \ast (-25:25))$$

The window applied here (by default) is a simple rectangular window. (This actually is a length 51 filter that best approximates the ideal lowpass filter, in the integrated least squares sense.) To display the filter’s frequency response in FVTool, type

```matlab
fvtool(h,1)
```

As you see, ringing and ripples occur in the frequency response, especially near the band edge. This “Gibb’s effect” does not vanish as the filter length increases, but a nonrectangular window reduces the magnitude of the side lobes (but not the main lobe).

**Q11:** In the Magnitude Frequency response figure, use Analysis → Analysis Parameters to set Magnitude as linear (not dB). How much overshoot do you see? Is that consistent with the theory?

Multiplication by a window in the time domain causes a convolution in the frequency domain. Apply a length 51 Hamming window to the filter and display the result using FVTool:

```matlab
bw = b.*hamming(51)';
fvtool(bw,1)
```
Using a Hamming window greatly reduces the ringing. This is at the expense of transition width (the windowed version takes longer to ramp from passband to stopband) and optimality (the windowed version does not minimize the integrated least squared error).

Try:

```matlab
b=0.4*sinc(0.4*(-25:25));
fvtool(b,1)
bw=b.*(hamming(51)’);
fvtool(bw,1)
```

Right-click the $y$-axis label in FVTool and choose **Magnitude squared** on both plots.

**Q12:** Where in the plot is the ringing reduced by the window?

### Standard Band FIR Design

The Signal Processing Toolbox functions `fir1` and `fir2` are both based on the windowing method. Given a filter order and a description of an ideal filter, these functions return a windowed inverse Fourier transform of the ideal filter. Both use Hamming windows by default, but they accept any windowing function.

`fir1` resembles the IIR filter design functions in that it is formulated to design filters in standard band configurations: lowpass, bandpass, highpass, and bandstop. The commands

```matlab
n=50;
Wn=0.4;
b=fir1(n,Wn);
```

create a row vector $b$ containing the coefficients of the order $n$ Hamming-windowed filter. This is a lowpass, linear phase FIR filter with cutoff frequency $Wn$. $Wn$ is a number between 0 and 1, where 1 corresponds to the Nyquist frequency, which is defined as one-half the sampling frequency.

For a highpass filter, simply append the string ‘`high`’ to the function’s parameter list. For a bandpass or bandstop filter, specify $Wn$ msu the given as a two-element vector containing the passband edge frequencies; append the string ‘`stop`’ for the bandstop configuration.

```matlab
b=fir1(n,Wn,’high’);
```

uses the window specified in column vector `window` for the design. The vector `window` must be $n + 1$ elements long. If you do not specify a window, `fir1` applies a Hamming window.
The \textit{kaiserord} function estimates the filter order, cutoff frequency, and Kaiser window $\beta$ parameter needed to meet a given set of specifications. Given a vector of frequency band edges, a vector of magnitude, and a maximum allowable ripple, \textit{kaiserord} returns appropriate input parameters for the \textit{fir1} function.

Try:
\begin{verbatim}
edit fir1demo
fir1demo
.............................. Exercise
\end{verbatim}

1. Design a windowed FIR bandstop filter to remove the 300 Hz component from the three-tone signal with noise $y$ from Lab \textit{IIR Filters in Matlab}. Use a sampling frequency of 8192 Hz.

2. Plot the response.

3. Filter the signal $y$ with the designed filter.

4. Compare signals and spectra before and after filtering.

\textbf{Arbitrary Response FIR Filters}

The \textit{fir2} function also designs windowed FIR filters, but with an arbitrarily shaped piecewise linear frequency response. (The IIR counterpart of this function is \textit{yulewalk}).

\begin{verbatim}
b=fir2(n,f,m);
\end{verbatim}
returns row vector $b$ containing the $n+1$ coefficients of an order $n$ FIR filter. The frequency-magnitude characteristics of this filter match those given by vectors $f$ and $m$.

\begin{verbatim}
b=fir2(n,f,m,window);
\end{verbatim}
uses the window specified in column vector \textit{window} for the design. The vector window must be $n+1$ elements long. If you do not specify a window, \textit{fir2} applies a Hamming window.

The function \textit{cfirpm} is used to design complex and nonlinear-phase equiripple FIR filters. It allows arbitrary frequency-domain constraints.

Try:
Multiband Filters

The function \textit{firls} designs linear-phase FIR filters that minimize the weighted, integrated squared error between an ideal piecewise linear function and the magnitude response of the filter over a set of desired frequency bands.

\[ b = \text{firls}(n,f,a) \]

returns row vector \( b \) containing the \( n+1 \) coefficients of the order \( n \) FIR filter whose frequency-amplitude characteristics approximately match those given by vectors \( f \) and \( a \).

The function \textit{firls} allows you to introduce constraints by defining upper and lower bounds for the frequency response in each band.

The function \textit{fircls1} is used specifically to design lowpass and highpass linear phase FIR filters using constrained least squares.

Try:

\begin{verbatim}
edit firlsdemo
firlsdemo

edit firclsdemo
firclsdemo

.................
\end{verbatim}

Raised Cosine Filters

The sinc function, which is the impulse response of an ideal lowpass filter, forms the basis for several other interpolating functions of the form
\[ h(n) = f(n)sinc\left(\frac{n}{n_s}\right), \quad \text{where} \quad f(0) = 1 \]

One commonly-used form is the *raised cosine* function:

\[ h_{rc}(n) = \frac{\cos(\pi R \frac{n}{n_s})}{1 - (2R \frac{n}{n_s})^2}sinc\left(\frac{n}{n_s}\right), \quad 0 \leq R \leq 1 \]

\( R \) is called the *rolloff factor*. Like the sinc function, the raised cosine function is 1 at \( n = 0 \) and 0 at all other sampling instances \( n = n_s \). In contrast to the sinc function, the raised cosine has faster decaying oscillations on either side of the origin for \( R > 0 \). This results in improved reconstruction if samples are not acquired at exactly the sampling instants (i.e., if there is *jitter*). It also uses fewer past and future values in the reconstruction, as compared to the sinc function. The shape of the function’s spectrum is the “raised cosine”.

The ideal raised cosine lowpass filter frequency response consists of unity gain at low frequencies, a raised cosine function in the middle, and total attenuation at high frequencies. The width of the transition band is determined by the rolloff factor.

\[
\text{b=firrcos(n,F0,df,fs)}
\]
\[
\text{b=firrcos(n,F0,df,fs,'bandwidth')}\]

are equivalent, and return an order \( n \) lowpass linear-phase FIR filter with a raised cosine transition band. The cutoff frequency is \( F_0 \), the transition bandwidth \( df \), and sampling frequency is \( fs \), all in hertz. \( df \) must be small enough so that \( F_0 \pm df/2 \) is between 0 and \( fs/2 \). \( b \) is normalized so that the nominal passband gain is always equal to 1.

\[
\text{b=firrcos(n,F0,r,fs,'rolloff')}\]

interprets the third argument, \( r \), as the rolloff factor instead of the transition bandwidth, \( df \). \( r \) must be in the range \([0, 1]\).

Try:

\[
\text{b = firrcos(20,0.25,0.25,2);}
\]
\[
\text{freqz(b)}
\]

Describe what you see.
**Frequency Domain Filtering**

Often, a long (perhaps continuous) stream of data must be processed by a system with only a finite length buffer for storage. The data must be processed in pieces, and the processed result constructed from the processed pieces.

In the overlap-add method, an input signal $x(n)$ is partitioned into equal length data blocks. The filter coefficients (impulse response) and each block of data are transformed to the frequency domain using the FFT, where they can be efficiently convolved using multiplication. The partial convolutions of the signal are returned to the time domain with the IFFT, where they are shifted and summed using superposition.

$fftfilt$ implements the overlap-add method for FIR filters.

```matlab
y=fftfilt(b,x)
```

uses an FFT length of $nfft = 2^{\text{nextpow}(2,n)}$ and a data block length of $nfft-length(b)+1$ (ensures circular convolution).

$firfilt$ incurs an “offline” startup cost when converting the coefficients $b$ to the frequency domain. After that, the number of multiplications $fftfilt$ performs relative to $filter$ (which implements the filter in direct form) is $\approx \log_2(L)/N$, where $L$ is the block length and $N$ is the filter length. (Multiplications are a good measure of performance, since they are typically expensive on hardware.) The net result is that for filters of high order, $fftfilt$ outperforms $filter$.

Try:

```matlab
x=[1 2 3 4 5 6];
h=[1 1 1];
y=conv(h,x)
x1=[1 2 3];
x2=[4 5 6];
y1=conv(h,x1)
y2=conv(h,x2)
Y1=[y1, zeros(1,3)];
Y2=[zeros(1,3),y2];
Y=Y1+Y2
```

**Q12:** Describe what is happening in this code.
The following script takes a few moments to run.

```
edit filttimes
filttimes
```

Above what order can you say that `fftfilt` is always faster?

(The material in this lab derives somewhat from the MathWorks training document “MATLAB for Signal Processing”, 2006.
Revised, February 4, 2011)

©2012GM