EC581 DSP Projects: Lab Project #3

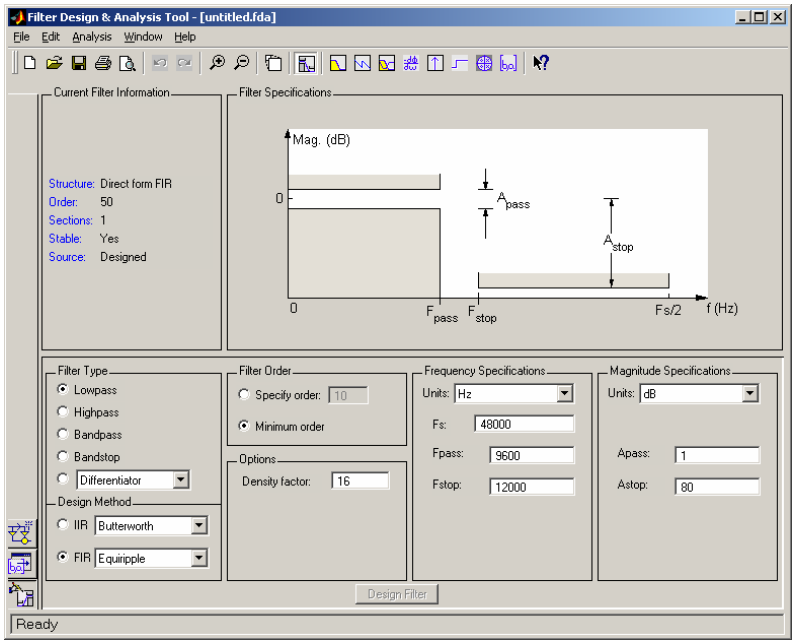
Dept. of Electrical and Computer Engineering
Rose-Hulman Institute of Technology

IIR Digital Filter Design Using MATLAB fdatool Filter Design Program
IIR Filter Implementation via C-language Program,
and Digital Wah-Wah Sound Effect
Last Modified on 3/15/2002 (KEH)
Adapted to MATLAB and 6711DSK by Mark A. Yoder (1-Apr-2004)

Part 1. Use of fdatool Digital Filter Design Program in MATLAB

1. In this project we consider the design and real-time implementation of an infinite impulse response (IIR) digital filter. We again use MATLAB to design, evaluate, and even perform (off-line) IIR digital filtering in much the same way as we used MATLAB to design FIR filters in Lab Project 2. To learn more about designing IIR filters in MATLAB, enter MATLAB and type HELP followed by one of these keywords (BUTTER, BESSELF, CHEBY1, CHEBY2, or ELLIP) Here we shall use an even more convenient and powerful digital filter design program called `fdatool`. You can read more about this program by entering `helpwin fdatool` in Matlab, then clicking on **index** and searching for **fdatool**.

2. Start this program by staring Matlab and entering `fdatool`. You should see:



3. Click the **IIR** (infinite impulse response) option option on the bottom left and select **elliptic**. Also select the desired filter type. (In this case, choose **Band pass)**.

4. A **Band pass Filter** prototype will appear, below it are boxes into which the user must enter the desired parameters of the bandpass filter. As an example, let's design a band pass filter (BPF) that

band-limits audio speech digitized at an 48000 Hz sampling rate, which extends from 1800 to 12,000 Hz. Enter the following numbers into the dialog box:

Sampling Frequency	**48000**
Fstop1	**1500**
Fpass1	**1800**
Fpass2	**12000**
Fstop2	**13000**
Units	**dB**
Astop1	**20**
Apass	**3**
Astop2	**20**

To implement this design, hit the **Design Filter** button at the bottom of the window. In a moment the frequency response for the filter will appear in the graph. To the left, note that for this particular set of specifications, a 6^{th} order elliptic filter is needed. You should keep in mind, however, that sometimes other filter types should be chosen, even if they are of higher order. For example, if no pass-band or stop-band ripple is wanted, you should pick the Butterworth (maximally flat) design. If nearly constant delay over the filter's passband (linear phase shift) is required to eliminate delay distortion of the in-band waveshape, you should pick the Bessel design. You could also elect to enter a different order design for the selected filter, although if the order number that is entered is lower than the one recommended, the indicated filter specifications will not be met.

The filter will be implemented and its associated performance plots will be displayed. Note from the "***log magnitude (dB)***" frequency response Bode plot that the filter performance that was specified above is met. You may have to zoom in to clear see the following. For example, note that in the passband (between 1800 and 12000 Hz), the filter's gain magnitude ripples between 0 dB and -3 dB. Likewise, in the stop bands (between 0 Hz and 1500 Hz, as well as between 15000 Hz and above), the filter's gain magnitude must never exceed –20 dB. Obtain a hard copy of this plot for your report, and include this as ***Attachment A***. Mark on your printout the various design specs.

Go to the **Analysis** menu and select **Group Delay Response**. The "***Group Delay***" plot shows that the filter will offer different delays at different frequencies within the 1800 to 12000 Hz passband. This means that this filter will somewhat distort the waveshape of "in-band" multi-frequency signals, since different frequency components within the passband will arrive at the filter's output at different times. However, this is no problem when filtering audio signals, since the human ear has the remarkable property of being essentially "phase insensitive". Obtain a hard copy of this plot for your report, and include this as ***Attachment B***.

Finally, obtain a copy of the resulting "***Pole-Zero***" plot of the filter's transfer function, H(z) (**Analysis:Pole/Zero Plot**). In your report, you must "graphically interpret" the sinusoidal steady state response of the filter represented by this plot, showing that the indicated positions of the poles and zeros correspond to a rather sharp 1800 – 12000 Hz bandpass filter, as represented by the companion "log magnitude" plot. Recall that the sampling frequency, Fs = 48 kHz.

5. Save your design specifications and filter design coefficients. Do these by clicking on **File –> Save Session As**. This saves your design specifications. To save your coefficients in a 'C' readable from click **File->Export to C Header file**. Include a printout of this IIR filter coefficient text file as ***Attachment D***.

Part II. Real-Time IIR Digital Filtering Program in C

1. A C-language digital filtering program is given in **Appendix A** that implements the following digital filtering equations for each:

$$w(n) = x(n) + a_1 w(n - 1) + a_2 w(n - 2)$$
$$y(n) = w(n) + b_1 w(n - 1) + b_2 w(n - 2)$$

Note that the output value from one 2nd-order stage becomes the input value for the next stage. Note that this program also performs a single final multiplication by the "Overall Gain Coefficient", H_d. This routine is designed to use the `.h` file generated by **fdatool**. To use it call:

```
IIRfilter(Right, Right, NUM, DEN, BUFFER_COUNT, DLY, MWSPT_NSEC);
```

`DLY` is a buffer that holds past sample values. Define it in your calling routine as:

```
static float   DLY[4][2] = {0,0,0,0,0,0,0,0};
```

Look in the file fdatool generated file to see how the other variables are defined.

2. Set the function generator to deliver a 6-kHz sine wave whose amplitude is adjusted so as not to distort the output waveform over the entire passband frequency range. As you vary the frequency of the input sine wave between 600 Hz and 24 kHz, observe that the sine wave is only heard over the intended passband (1800 to 12000 Hz).

3. Now connect an oscilloscope in place of the audio amplifier, and measure the gain at several (at least 10) frequencies, which include points in the filter's stop bands, transition bands, and in its passband. Plot these measured gain values (in dB) ***directly over*** the log-magnitude gain plot obtained by `fdatool`. Your observations should fall reasonably close to the predicted filter's gain magnitude. As in the previous FIR digital filter lab, you may have to add a constant offset (in dB) to the measured dB gain data, in order to account for the unknown gain of the mixer box (unless the mixer box is bypassed). Include this gain plot, along with the hand-plotted gain values ***plotted directly over the fdatool predicted filter gain plot***, as ***Attachment E***.

Part III. Digital Wah-Wah Audio Special Effect Program

The "*Wah-Wah*" audio sound effect mimics the characteristic human "wah" sound that is made as the lips are gradually changed from a small "o" to a big "O" shape. At the same time, the vocal cords are forced to emit an impulse train of constant pitch (frequency). This impulse train is created by "puffs of air" periodically escaping through the cords held together under a certain tension, which sets the pitch period. Try making this sound yourself right now! The vocal cords typically vibrate at a relatively low fundamental frequency, perhaps 100 Hz. However, the spectrum of an impulse train is "spectrally flat", thus all harmonics are present with equal amplitude. (Of course, the waveform emitted by the vocal cords is not a perfect impulse train, so the harmonics are still strong, though they do decrease with harmonic number.) Thus many frequencies are present in the impulse train produced by the vocal cords, consisting

of a 100 Hz, 200 Hz, 300 Hz ... sinusoids. Thus, this waveform is "spectrally rich". As the lip and mouth shape gradually change while making the "wah" sound, the vocal tract acts something like a lowpass filter with a pronounced resonance peak near its cutoff frequency. As the lips are gradually changed from the small "o" to the big "O" shape, the cutoff frequency of the lowpass filter formed by the vocal chords changes in two ways. The break (cutoff) frequency gradually changes from a low value to a higher value and the height of the resonance peak (Q of the filter) changes. Thus, as the vocal tract filter goes through these changes, at first only a few of the lower frequency harmonics are emphasized at the output (mouth). However, as the mouth becomes wider and wider as the "wah" sound is carried to completion, the break frequency of the lowpass filter rises, and more and more of the higher frequency harmonics come out of the mouth.

The classic "Wah Wah" audio effects pedal consists of a pedal-operated, lowpass analog filter with a continuously variable cutoff frequency. The pedal uses a variable analog lowpass filter with a strong resonance peak near its cutoff point. Because this peak especially emphasizes the harmonics near the break frequency, it is a little hard to characterize this as a lowpass filter or a bandpass filter. This effect is especially interesting when applied to a "spectrally rich" audio signal, such as that obtained by playing a guitar through a signal clipping network (distortion box), which serves to spectrally richen the guitar waveform. To hear audible examples of classic guitar "wah-wah", visit the following URL:

http://www.betterguitar.com/Equipment/Effects/WahTechniques/WahTechniques.html

Your assignment is to see how close you can come to digitally emulating the sound of a classic analog Wah-wah circuit on the C67 DSK. Modify the digital IIR filtering program of Appendix A so it switches, at regular intervals, from one set of filter coefficients to another. Each successive set of coefficients corresponds to a bandpass filter of slightly higher passband.

Use **fdatool** to determine coefficients for eight (or more!) 4th-order Chebysheff IIR lowpass filters (LPFs). The eight LPFs should have break frequencies at 200 Hz, 400 Hz, 800Hz, 1200 Hz, 1600 Hz, 2000 Hz, 2500 Hz, and 3000 Hz. (You can put in any reasonable pass frequency, stop frequency, and stopband and passband ripple values that you want.)

Now modify the digital filtering program of Appendix A, so that it uses the first set of coefficients for about 0.1 seconds, the second set for about 0.1 seconds, the 3rd set for about 0.1 seconds, etc. Then allow the process to repeat. You will probably want to do this by setting up a periodic software interrupt and changes a global variable. You should be able to hear a "wah" sound as you listen to the output of this filter, whose LPF break frequency is varied, as a 50 Hz square wave is played into it. *Why will the "wah" effect not be apparent if you use a 440 Hz sine wave instead of the spectrally rich 50 Hz square wave?* You may have a problem with an annoying click or pop as you switch between different sets of filter coefficients. If this is a problem, can you suggest ways to reduce the click? Demonstrate this program to the lab instructor, get his signature on your program listing, and include this program listing as ***Attachment F***.

Appendix A. Real Time C-Language IIR Digital Filtering Program

```
typedef float real32_T;          // real32_t is generated by fdatool.

void IIRfilter(float *inbuf, float *outbuf,
                          const real32_T num[][3],
                          const real32_T denom[][3],
                          int buffSize,
                          float DLY[][2], int numSections)
{
    int i,j;
    float floatsamp, out, w;

       for(j=0; j<buffSize; j++) {

           floatsamp = inbuf[j];        /* Get sample */

           for (i = 1; i < numSections; i++) {
                 w = floatsamp - denom[i][1]*DLY[i][0] - denom[i][2]*DLY[i][1];
                 out = w  + num[i][1]*DLY[i][0] + num[i][2]*DLY[i][1];
                 DLY[i][1] = DLY[i][0];
                 DLY[i][0] = w;
                 floatsamp = out;
               }

           out *= num[0][0];                     /* Overall Gain Coefficient */

           outbuf[j] = out;       /* Put back in buffer                       */
           }
}
```

I. Optional

Recall that the magnitude of the transfer function of a filter that has a multiplicative constant "K" (whose value is not revealed by the Pole-Zero plot) contains *n* zeros (*z1, z2, … zn*) and *k* poles (*p1, p2, … pk*) is given by

$$|H(z)| = \left| \mathrm{K} \frac{(\mathrm{z} - \mathrm{z}_1)(\mathrm{z} - \mathrm{z}_2)\cdots(\mathrm{z} - \mathrm{z}_n)}{(\mathrm{z} - \mathrm{p}_1)(z - p_2)\cdots(z - p_k)} \right|$$

It is easily demonstrated that the magnitude of a product of complex numbers is the same as the product of the magnitude of the numbers, thus we may write

$$|H(z)| = K \frac{|z - z_1||z - z_2|\cdots|z - z_n|}{|z - p_1||z - p_2|\cdots|z - p_k|}$$

But each of the numerator terms of the form $|z - z_n|$ can be geometrically interpreted on the Pole-Zero plot as the distance between the observation (input signal) frequency and the n^{th} zero, while each of the denominator terms of the form $|z - p_k|$ can be geometrically interpreted as the distance between the observation frequency and the k^{th} pole. Thus the magnitude of the digital filter's transfer function |H(z)| is proportional to the product of the distances of the zeros to the observation frequency (signal frequency), divided by the product of the distances of the poles to the observation frequency. Note that we say "proportional to" because we do not know the value of the multiplicative constant "K", unless additional information is given, such as the gain magnitude at a given frequency. In this case, we desire the gain of the filter to be unity in the middle of the bandpass filter's passband, and from this additional requirement, K can be determined, if desired. If the input signal to the filter is a sinusoid ranging in frequency from 0 to Fs/2 = 24 kHz, recall that the analog sinusoidal steady state input signal frequency corresponds to a point somewhere on the upper half of the unit circle in the z-plane, where $z = e^{j2\pi fT}$ where T = 1/Fs.

Recall that the rightmost point on this unit circle corresponds to zero frequency, f = 0 Hz; the uppermost point on the unit circle corresponds to a steady-state input sinusoid of frequency f = Fs/4 = 12 kHz., and the leftmost point on the unit circle corresponds to a steady-state input sinusoid of frequency f = Fs/2. Locate as precisely as possible (you will need to use a protractor) the points on the upper-half of the unit circle on this filter's Pole-Zero plot that represent the frequencies 1500 Hz, 2400Hz, 16800 Hz, and 21000 Hz. Note that the middle two of these given frequencies are just ***inside*** the specified filter passband, while the outer two are ***outside*** the passband). Draw lines from the observation frequency on the unit circle to each of the poles and zeros. Then measure the lengths of these lines carefully with a ruler (using any desired units – just so you use the same units for all distance measurements.) Calculate the relative gain that corresponds to each of these four observation frequencies. Recall that these gains will be accurate to within a multiplicative constant "K"; that is why I call them "relative gains". Include this Pole-Zero plot, along with your graphical measurements and calculations for the filter gains at the four sinusoidal test frequencies, as ***Attachment C***. Because this is a bandpass filter, the gains at the two middle test frequencies should be considerably greater than the gains at the two end test frequencies. What is a reasonable value of "K" that will force the filter to exhibit unity passband gain?