ECEn 487 Digital Signal Processing Laboratory

2014 Lab 2
Finite Impulse Response Filtering

Due Dates

This is a three week lab. All TA check off must be completed by Friday, February 21, at 3 PM or the lab will be marked late.

Lab book write-up submission, beginning of lab class Friday, February 21.

Learning Objectives

The purpose of this lab is to demonstrate real-time filtering of signals. An additional purpose of this lab is for each student to design an FIR digital bandpass filter using MATLAB and use it to extract a desired Morse code signal from interfering signals at nearby audio frequencies. The extracted message will then be decoded. Students will become familiar with practical aspects of simple filter design techniques and gain an appreciation for its possible uses.

Reading Assignment

1. Textbook, Oppenheim and Schafer, Sections 7.0 , 7.5, 7.6.1, and 8.7.3 (for the later portions of the lab).

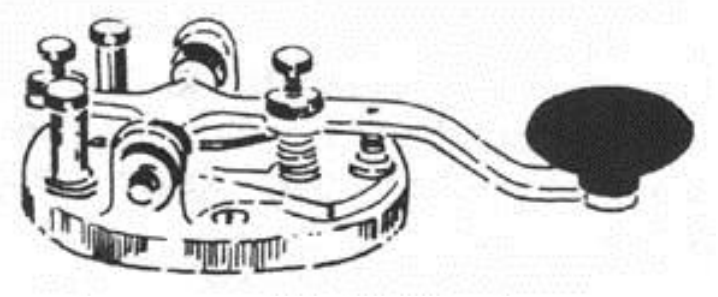
2. MATLAB online help and Signal Processing Toolbox User's Guide tutorial and documentation on the signal processing toolbox (e.g. “help signal”) functions "freqz,", "fft," "conv," "filter," "fir1," "hamming," "bartlett,"blackman," "boxcar," "kaiser," and "chebwin."

Introduction

One of the most important and practical real-time DSP operations is digital filtering of sampled signals. Though IIR filters are sometimes used, we will concentrate on FIR filter implementations because they are always stable, can be designed to be exactly linear phase (no phase distortion or dispersion), and are easy to design to given filter specifications. Your goal in this lab will be to extract a particular Morse code signal from a hopelessly scrambled recording so that you can decode the message. This simulates a scenario that an amateur radio operator might encounter when trying to copy a weak overseas signal in a channel packed with local

interference from other amateurs. Your only hope is that the interfering signals are centered on slightly different audio frequencies, so that a frequency selective filter may help reject them. We will use the windowed filter design technique from your textbook to design the filter.

Morse code is an early binary on-off keying method used to manually communicate text characters. The transmitting operator operates a switch by hand to generate a series of longer "dash" tones, and shorter "dot" tones to form a character code. Characters are separated by longer spaces than the dots and dashes within a character. One ingenious aspect of the code is that more frequently used characters (in English) are assigned shorter codes. This anticipates the later practice of entropy coding for efficient digital communications. Today, the code is used primarily by amateur radio operators for fun and emergency communications. You can find a code table (which you will need to decode the message) at many websites, like http://www.qsl.net/4f5aww/module3c.htm. With a little work on google you can find programs to practice sending and receiving code (though not required for the lab).



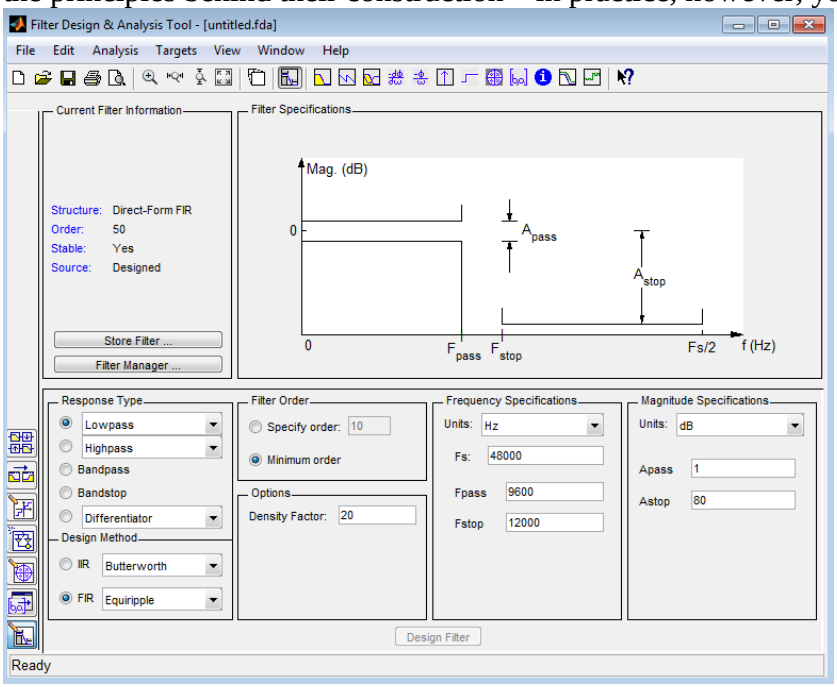
A classic Morse Code key

The corrupted Morse code signal you must filter is in the file "mix10.wav" available on the class web page. This is a single channel (mono) signal sampled at 16.2 kHz with 8 bits per sample. You can read it into MATLAB with the command "x = wavread('mix10.wav');" Also check out the functions "wavwrite" "wavplay," and "sound" which you can use to output your filtered result.

Note: The experiments below are not intended to each be one week long. There are five experiments for a three-week lab. If you finish one before the end of the lab period, start on the next. The latter experiments are harder and take longer!

***Experiment 1* Design of filters using pole-zero representations**

1. Launch "fdatool" from MATLAB. The "Filter Design & Analysis Tool" is extremely powerful and can be used to design and implement all of the different techniques in the book. Some may say that you should only see this tool after the class is finished, but I think that knowing and using the tool can help to reinforce the concepts in this class. We want you to learn how to design filters using various methods so that you can understand the principles behind their construction – in practice, however, you will often use a tool.



2. Now that you have the filter tool open, you can enter various parameters and it will produce the filter for you. For example, set the units to Normalized (0 to 1). Then set up ω_{pass} and ω_{stop} to be 0.3 and 0.8 respectively. You can leave Apass to be 1 and Astop to be 80. Then press the "Design Filter" button. This actually uses what is called the Parks-McClellan algorithm to design an optimal filter (FIR Equiripple).

3. Now you can look at all of the properties of this filter. On the top sidebar, you can now look at "Magnitude Response", "Phase Response", "Group Delay", "Impulse Response", "Step Response", "Pole/Zero Plot", and the "Filter Coefficients". Look through these and see how the filter is being implemented. This is an incredible resource! Basically, most of what we do in class can be represented here.

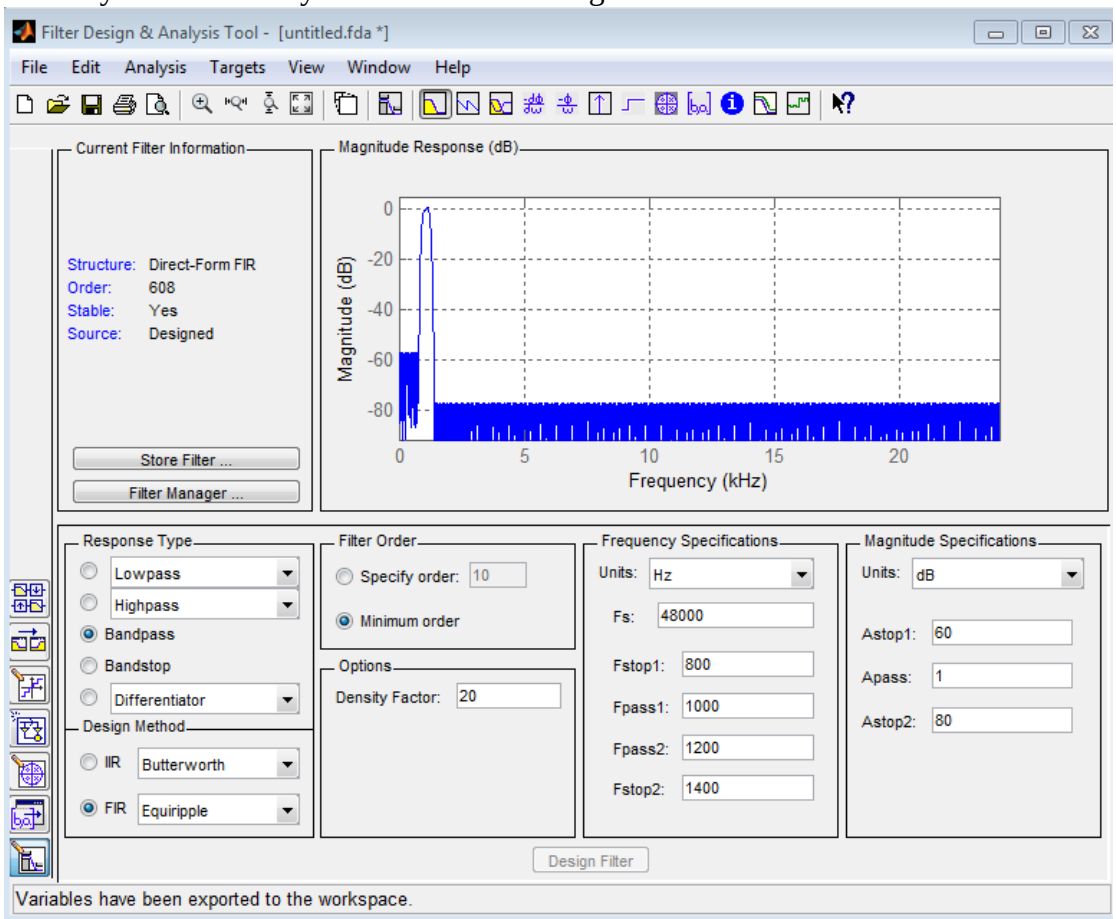
4. Next, on the left sidebar, choose the Pole/Zero editor. Here you can actually move the poles and zeros around and look at the response. Move some of the zeros and poles around and look at the resulting "Magnitude Response". You should be able to see very clearly here that when you move a pole from the central location to near the unit circle,

you get a strong increase in the strength of the signal at that radian frequency and when you move a zero near to the unit circle you get a strong decrease at that radian frequency. Spend some time with this plot moving things around. You will also notice that man of the poles and zeros are in conjugate pairs, which gives us symmetric responses and keeps our filter real.

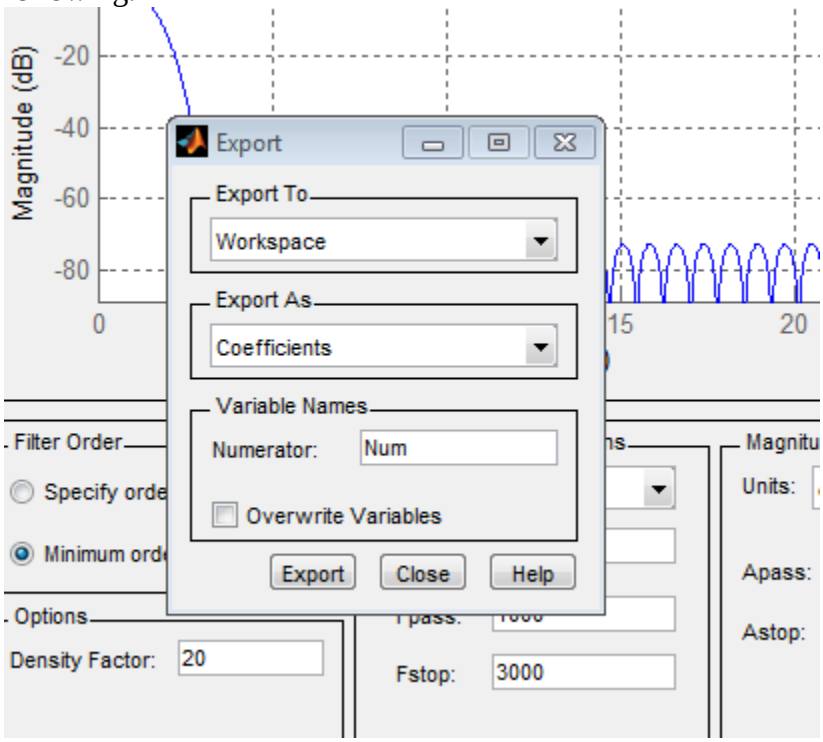
5. Go back to the "Design Filter" section on the left sidebar. Now change the design method to be "IIR Butterworth". Compare the placement of the poles and zeros with that of the filter designed using the FIR Equiripple method. Consider the "Order" of the filter and the number of coefficients. You will notice that there are now many coefficients in the denominator which corresponds to feedback. Compare the "Group Delay" of the two filters as well. Notate for yourself some of the major differences of these two different types of filters. You can also play with the poles and zeros and see the responses.

<u>Experiment 2</u> Build a "real time" continuous FIR filtering program in MATLAB.

1. Design a bandpass filter with the following specifications using the "fdatool". This is certainly the easiest way to make the following filter.



2. Next, go to the toolbar and click on File -> Export… You should get a box that looks like the following.



3. When you press the “Export” button a new variable, “Num” will appear in your workspace. These are the FIR filter coefficients for the filter that you designed.

4. Set up your laboratory area like in lab 1 so that you can provide an input from the function generator, see it on the scope, pass the signal through the computer, and then see the output on the scope and listen to it using the headphones.

5. Now, modify the “loopback” code from lab 1. Include the following call for the output:

```
y_data = filter(Num, 1, input_data);
```

   This will filter each block of “input_data” that you record and then output it at the next “step” command.

6. Filter some sampled music, or speech with this new code. Why do you hear some clicking at a repeated interval?

7. Read the documentation about "filter" regarding the "Zi" and "Zf" optional parameters. Adjust your code to now use these optional parameters. You will need some code (and some initialization before calling a line similar to this line):

```
[y_data zi] = filter(Num, 1, input_data, zi);
```

These parameters allow you to save and restore, between successive calls, the filter's data memory values in its shift register. Compare what the output sounds like with and without using these. Why does it make a difference? Why is this an important capability in any DSP filter implementation? (explain in your report).

8. Now use the function generator and the oscilloscope to very that your filter is appropriately filtering the frequencies that you have specified. Sweep over the attenuated and bandpass frequencies and record the amplitudes on your oscilloscope. Does it match your specification?

9. Using the continuously running filter-loop program, listen to the filter output with a CD music signal input. How does it compare to no filter?

10. Drive your filter with a periodic, very narrow analog pulse. Observe the output on the scope. What are you looking at? How does it compare with what you have calculated for this filter using "fdatool".

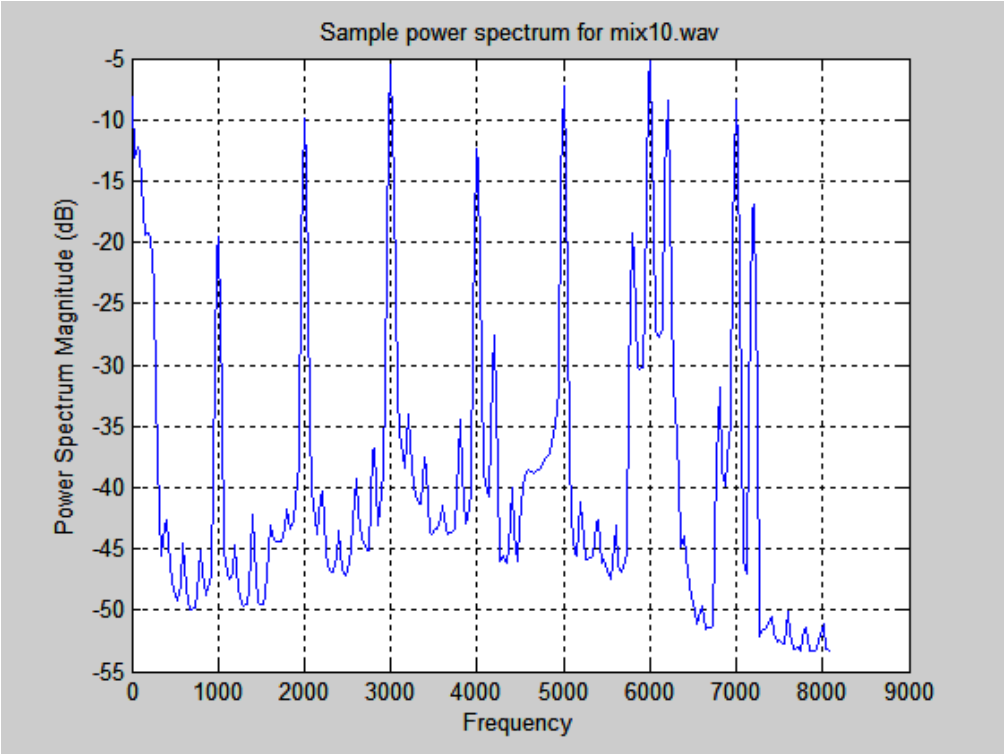
11. Design your own filter using "fdatool". Use it to filter signals using your function generator and oscilloscope. Record the amplitude differences using the filter. Demonstrate your running program to the TA for sign off.

12. Note: the following steps will now go into more details about how this filtering is actually implemented. We wanted you to see how this can be applied first, and then now dive into more specifics about how this really works.

Experiment 3 Design an FIR bandpass filter using the window method

Procedure

1. Generate a set of specifications for a filter to extract the desired Morse code signal while rejecting interfering neighbors. Specification must include passband and stopband edge frequencies and maximum ripple levels. Base your design on the prior knowledge that the desired signal is centered on 3.0 kHz, and that there are about 7 interfering signals present. All 8 (1 desired and 7 interferers) signals are spread fairly uniformly in frequency from 250 Hz to 7 kHz, as shown in the figure below. Of course you must translate the analog frequencies (like 1.0 kHz) to the corresponding discrete-time radian/sample frequencies, ω, based on the sample rate. Specify passband and stopband corner frequencies and stopband attenuation levels (in dB). There is no single correct answer here; be creative.



2. Use the approach described in the textbook, Section 7.2, to design an FIR filter $h[n]$ using the windowed method so that it matches your specification. You may use any of the listed window types that you think will work best (i.e. rectangular, Bartlett, Hanning, and Manning). Note that MATLAB has built-in functions to generate each of these windows. Also note that the text only talks about designing a lowpass filter. You can compute the $h_d[n]$ (desired ideal impulse response) for a bandpass filter as the difference between two *same length* lowpass filters which have different corner frequencies. Let $h_{1,d}[n]$ be the impulse response for an ideal lowpass filter with corner frequency ω_1, and $h2_d[n]$ likewise with corner $\omega_2 > \omega_1$. $h_d[n] = h_{2,d}[n] - h_{1,d}[n]$ is the ideal bandpass filter impulse response with corners ω_2 and ω_1. The ideal response equation which you will window is thus

$$h_d[n] = \frac{\sin[\omega_2(n - M/2)]}{\pi(n - M/2)} - \frac{\sin[\omega_1(n - M/2)]}{\pi(n - M/2)}$$

$$= \frac{\omega_2}{\pi}\,\mathrm{sinc}\left[\frac{\omega_2(n - M/2)}{\pi}\right] - \frac{\omega_1}{\pi}\,\mathrm{sinc}\left[\frac{\omega_1(n - M/2)}{\pi}\right]$$

where $N = M + 1$ is the filter length. Note that the second form may be easier to implement in MATLAB, because the built-in sinc function handles the division by zero that occurs when $n = M/2$.

In your design, make N as short as is practical so that the DSP computational requirements are minimized. For a given window choice, N controls transition bandwidth. For a given choice of N, the window shape controls stopband attenuation level. In general, windows with lower stopband ripple also have wider transition bands. Table 7.2 and equations 7.73-7.76

may help with picking *N* and selecting the window. Also, in MATLAB, type "help signal" to look at a list of all the built-in signal processing functions. Check out the sections titled "Digital Filters > FIR filter design," and "Windows" to see if there is anything helpful. You may not use the function "fir1" or "fdatool" which does a complete windowed filter design. Your code must show that you generated the desired ideal impulse response and multiplied it by a window.

3. Plot the impulse response and frequency response of your design (using "freqz.")

4. If the filter frequency response does not look acceptable, repeat steps 1-3.

5. Show your filter design and frequency response plot to a TA for check-off.

***Experiment 4* Build a batch-mode FIR filtering program in MATLAB.**

Procedure

1. Write a simple FIR filter loop program in MATLAB which uses your FIR impulse response designed in experiment 3. Use a "for" loop structure which increments the output sample index with each pass through the loop to implement the convolution sum directly:

$$y[n] = \sum_{k=0}^{L-1} h[k]\, x[n-k].$$

2. Verify basic operation by filtering short pure sinusoid sequences you generate (within MATLAB, i.e. not sampled) at three frequencies: one in the filter passband, and one for each of the two stopbands. Compute and record the total power levels at each frequency, and compare them to see if the desired stopband attenuation, in dB was achieved.

3. Read in (using "wavread") the entire Morse Code signal. Your "for" loop filter implementation is (probably) too slow to filter it in a reasonable time. Try it, but stop operation with ^c (control c) when you get bored. Then, do this using the built-in functions "conv" and "filter."

4. Filter the entire corrupted Morse code signal in memory (using "conv" or "filter" functions) and play the result through the sound card (using "sound" or "wavplay"). This is referred to as "batch" mode processing, where all the data of interest is in memory at one time, and you process (filter) it as a single batch.

   If your filter is designed correctly you should hear a single signal with very little interference. If the filter performs poorly in rejecting interference, or if it is too long (too many taps) to run in a reasonable time, go back to experiment 3 and redesign appropriately.

5. What is the coded message? Have the TA verify your result.

Experiment 5 Build a FIR filtering program using the FFT-based overlap-add method.

1. Rewrite your batch mode FIR filter code using the DFT-based "overlap add" method described in textbook section 8.7.3. The basic code structure from Experiment 4 step 1 is a good starting point. Do not use "filter." By the way, this is how "fftfilt" is implemented internally. In MATLAB use the "fft" function to implement the DFT (fft is just a fast algorithm to compute DFTs).

2. Verify proper operation by filtering the Morse code signal and comparing with the results of experiment 2. You should hear no new noise, pops, clicks, or distortion.

3. Compare the speed of the "filter" function and the "fftfilt" function using "tic" and "toc". You can do this by producing random vectors for both the filters and the data. You may have to run your code once to load the functions in memory so that the second time you execute will be much faster.

4. Estimate (just by timing with a watch, or using the "tic" and "toc" MATLAB functions) how much faster this implementation is than your experiment 2 direct convolutions sum "for loop" solution. Explain why it is faster. Try different FFT lengths to see which is fastest for your filter. Demonstrate your running program to the TA for sign off.

Conclusions

Document your design procedures and results, including plots of the designed impulse and frequency responses and listings of the MATLAB scripts and/or functions you wrote. Record the Morse code message you decoded. Write a paragraph or two of conclusions for your lab experience. Discuss any additional implications of what you observed. Describe what you feel are the important principals demonstrated in this lab, and note anything that you learned unexpectedly. What debug and redesign procedures did you need to perform to get it to work?

Laboratory 2 TA Check-off Page (to be submitted with laboratory report)

Student Name: _____

_____ Task 2.11 – Demonstrate your working real-time filtering to the TA.

_____ Task 3.5 – Show your filter design for the bandpass filter to the TA.

_____ Task 4.5 – Show the decoded message to the TA.

_____ Task 5.4 – Demonstrate your working FFT-based overlap-add filtering technique.