Due Dates

This is a three week lab. All TA check off must be completed by the start of lab class, February 22, or the lab will be marked late.

Submit answers to the questions from the last page of this handout at the beginning of lab class on Friday, Feb. 8.

Lab book write-up copy submission, beginning of lab class Friday, Feb. 22

Learning Objectives

The 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.


Introduction

One of the most important and practical real-time DSP operations is digital filtering of sampled signals. Though infinite impulse response (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. 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')."

Note: The experiments below are not intended to each be one week long. There are four 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 an FIR bandpass filter

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 2.0 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, $\omega$, 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.

![Sample power spectrum for mix10.wav](image)

2. Use the approach described in the textbook, Section 7.5, 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 $\omega_1$, and $h_{2,d}[n]$ 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 $\omega_2$
The ideal response equation which you will window is thus

\[ h_0[n] = \frac{\sin(\omega_2(n - M/2))}{\pi(n - M/2)} - \frac{\sin(\omega_1(n - M/2))}{\pi(n - M/2)} \]

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

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" 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 and have him/her sign your lab book to indicate completion of this experiment.

**Experiment 2 Build a block mode FIR filtering program in MATLAB.**

**Procedure**

1. Write a simple FIR filter loop function in MATLAB which uses your FIR impulse response designed in experiment 1. 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 too slow to filter it in a reasonable time. Try it, but stop operation with ^c (control c) when you get bored. MATLAB uses faster frequency domain FFT based implementation in the built in functions "conv" and "filter." You will develop a similar code in experiment 3.

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 1 and redesign appropriately.

5. What is the coded message? Have the TA verify your result and sign the lab book.

**Experiment 3** Build a real-time continuous FIR filtering program in MATLAB.

1. Modify your code from experiment 2 so that even though the sample data resides in memory all at once as a single input batch (data matrix), you break it into smaller 1024 sample blocks for successive calls to "filter" in order to build up the output batch data matrix one block at a time. In other words, you will write a loop that calls "filter" on each pass with 1024 new samples, then build the output data matrix by concatenating the filters result blocks together.

Filter some sampled music, or speech with this new code. Read the documentation for the "filter" MATLAB function regarding the "Zi" and "Zf" optional parameters. 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 lab book).

2. Using techniques similar to the loopback program of Lab 1, and building on the code from step 1 in this experiment, write a continuously running program to filter signals seen on the ADC input, and play them back through the DAC output. Use block sizes no longer than one second. Use the 'async' option on "wavplay" to reduce inter-block dead times. Use "filter" as the filtering code, with the "Zi" and "Zf" optional parameters to carry over filter memory from block to block.

3. Using the code from step 2, verify that your filter's corner frequencies and maximum sidelobe amplitude are as you designed, by sweeping the frequency of a signal generator as the input signal, and observing the output amplitude on an oscilloscope. Record at least three amplitude points in each band (one passband and two stopbands). Hint: If you have trouble
with excessive delays between blocks your filter may be too long for the PC to keep up with computation at the specified sample rate.

4. 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, but using the same sample rate? Demonstrate your running program to the TA for sign off.

5. Drive your filter with a periodic, very narrow analog pulse (ask the TA how to set the function generator to do this. Note that sometimes a lowpass filter will produce easier to understand results in this experiment so you may want to design one rather than use your bandpass filter). Observe the output on the scope. What are you looking at?

Experiment 4 Build a FIR filtering program using the FFT-based overlap-save 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 3 step 1 is a good starting point. Do not use "filter." By the way, this is how "filter" 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. 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 (part of your grade on this project will depend on whether you received the correct message.) 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?
Questions (Due at beginning of second lab session)

1. List two advantages of using FIR filters rather than IIR filters.

2. Assume a filter is specified with maximum allowable passband ripple of $\Delta_p = 1.0$ dB and stopband ripple of $\Delta_s = -40$ dB. What are the corresponding linear scale ripple levels, $\delta_p$ and $\delta_s$?

3. Which of the window types discussed in the text (excluding the Kaiser window, which is adjustable) requires the shortest filter length, $N$, for a given transition bandwidth? Which has the highest stopband attenuation?

4. What MATLAB function can do everything described in Experiment 1 step 2 for you? (Note that you are not permitted to use this function in the lab!!!! We can't make life too easy for you.)

5. Give an equation, and cite sources from the text (page and equation numbers) for the impulse response, $h_d[n]$, of an ideal high pass filter.