ECE 340 – Computer Assignment #1

Filters and Demodulation (Draft)

Due by 5 pm on Friday, Nov. 11

The first computer assignment will explore filters, window functions, calculation of the Fourier Transform, and demodulation of communication signals. In the first part of the project, you will first write a program to numerically calculate the Fourier Transform of a signal. Then, you will write another script to modify the non-causal impulse response of an ideal lowpass filter by truncating it with a window function (similar to the demonstration I showed in class), and then convert it to a bandpass filter through modulation. You will then use your Fourier Transform program to evaluate the spectral properties of the resulting filters. In the second part, you will write a program to demodulate communications signals that I provide in order to remove the carrier frequency and hear the audio signal.

I will talk you through the steps of the assignment using this handout. You should observe the results of your simulations, and you should make (well-labeled) plots to demonstrate important details about the project. Use those observations to prepare a typed report explaining your methods and results for each section. There is not a specific format for the report, but be sure to include details that show you did the work and that you understand what is happening. Include plots only when they convey something significant. I will be looking for insight that shows a depth of understanding about these experiments. At a minimum, your report should include an introduction describing what the project is about and what the objectives are; a section discussing the techniques or methods that you used (including theoretical results if appropriate); a section describing your results; and a section with your conclusions. Any questions that I ask in this assignment should be answered, but more importantly these questions are designed to get you started in thinking about the assignment. The best reports will demonstrate analysis and thought beyond the basic questions posed in this handout. Listings of your Matlab code, appropriately labeled and commented, should be included.

You should work with **one** partner and turn in a joint report. I prefer for everyone to have a partner rather than working on your own. Groups may talk to each other about the assignment, **but all Matlab code and report materials must be the result of your own group's work**. Code copied from another group is unacceptable.

Part I: Fourier Transform and Filter Analysis

The Fourier transform is defined by

$$X(f) = \int_{-\infty}^{\infty} x(t)\exp(-j2\pi ft)\,dt\,. \tag{1}$$

In Part I, you will first write a program to numerically evaluate the Fourier Transform. The signals that you will be analyzing are stored on a computer, and therefore cannot be continuous in time. Instead, they are sampled versions of the analog signal. Suppose that the signal $x(t)$ in (1) is sampled at uniformly spaced instants in time and placed in a discrete-time sequence $x[n]$ where n is the sample index. For example, the first sample is $x[0]$, the next is $x[1]$, and so on. If the uniform time between samples is T_s, then the Fourier transform in (1) can now be approximated by

$$X(f) \approx T_s \sum_{n=-\infty}^{\infty} x[n] \exp(-j2\pi f n T_s). \tag{2}$$

This expression actually has a name – it is the discrete-time Fourier Transform. But unfortunately, we also cannot store a function of the continuous frequency variable f. Instead, we must also represent $X(f)$ with samples at different frequencies f_k. We haven't discussed the Nyquist sampling frequency in class yet, but it turns out that if the signal $x(t)$ is sampled at a rate F_s, then the highest frequency in the original $x(t)$ should be limited to $F_s/2$. Otherwise, there are some problems that occur. Therefore, set up an array of frequency values that are uniformly spaced from $-F_s/2$ to $+F_s/2$. For each frequency value, evaluate the Fourier Transform at that frequency using (2). In other words, for the kth frequency value f_k, you can use Matlab to evaluate

$$X(f_k) = T_s \sum_{n=0}^{N-1} x[n] \exp(-j2\pi f_k n T_s) \tag{3}$$

where N is the number of samples in the (necessarily) finite-length sequence $x[n]$. Repeat this computation for each frequency value and you will have a sampled version of the frequency spectrum. You can plot the magnitude and/or the phase of the resulting spectrum to get a good feel for the frequency content of the signal you are analyzing. Note: even though you have to repeat the calculations *for* each frequency, try to avoid using "for" loops. They slow Matlab down.

Next, you will use your Fourier Transform program to evaluate the spectral shape of bandpass filters made from different window functions. First, we know from class that the impulse response of an ideal lowpass filter with cutoff frequency of $B/2$ is an infinitely long *sinc* function described by $h(t) = B\text{sinc}(\pi B t)$. We can make a causal, practical lowpass filter by delaying the peak of the impulse response and by multiplying by a window function $w(t)$ to give the impulse response a finite length. The resulting impulse response is

$$\hat{h}(t) = B\text{sinc}(\pi B(t - t_d)) \cdot w(t). \tag{4}$$

The width and shape of the window function will determine the ultimate frequency response $\hat{H}(f)$ of the practical lowpass filter.

Finally, we can convert this filter to a bandpass filter using the modulation property of the Fourier Transform. If we multiply in the time domain by $\cos(2\pi f_c t)$, the bandpass filter is

$$\hat{h}_{BP}(t) = B\text{sinc}\left(\pi B(t - t_d)\right) \cdot w(t) \cdot \cos\left(2\pi f_c t\right) \tag{5}$$

with Fourier Transform

$$\hat{H}_{BP}(f) = \frac{1}{2}\left[\hat{H}(f - f_c) + \hat{H}(f + f_c)\right]. \tag{6}$$

Therefore, the modulation takes the practical lowpass filter and applies a frequency shift such that the filter is centered at frequency f_c.

Design a bandpass filter using this approach. Start with an ideal lowpass filter with a cutoff frequency of 20 kHz. Then, apply the window function to limit the time duration of the filter and modulate the impulse response to obtain a bandpass filter with a cutoff frequency of 100 kHz. Use your Fourier Transform program to evaluate the frequency spectrum at different steps in the process (for example, before and after modulation), and also for different window shapes (e.g., hanning, rectangular, and others). Can you make any conclusions about how the shape and width of the window functions affect the spectrum of the bandpass filter?

I will provide the script that I used in class to look at lowpass filters. This will give you a good start on delaying the ideal impulse response and applying the window function. However, this script was not for looking at bandpass filters. After modulation, the highest frequency in your filter response will be approximately $f_c + B/2$ = 100 kHz + 20 kHz = 120 kHz. When you create the discrete-time version of your impulse response prior to modulation, the rate that you sample the impulse response is critical – it must be at least twice this highest frequency. Preferably, the sample rate would be 5-10 times the highest frequency to be sure you generate a clean, distortion-free plot. This sample rate should be an input parameter to your Fourier Transform program, not something that is hard-coded inside the program.

Part II: Double-Sideband, Suppressed-Carrier Demodulation

Double-sideband, suppressed-carrier (DSB-SC) modulation is perhaps the simplest modulation technique. As seen in Section 7.7-1 of your textbook, a DSB-SC signal is created by directly multiplying an information-containing signal by a sinusoid at the carrier frequency. For example, let $m(t)$ be a signal that carries some information of interest such as data as music. In our case, $m(t)$ will be a clip from an audio file. Most of the frequency content in the audio file will be below 20 kHz because those are the frequencies we can hear, but it is difficult to transmit the signal to another location at these frequencies. Furthermore, if we wanted to transmit two audio signals at the same time, we couldn't because the two signals would combine during transmission in a way that prevents us from separating them at the receiver. To get around this, we typically use the information-containing signals to modulate a carrier in some way. For DSB-SC, this modulation is performed by generating the signal

$$x(t) = m(t)\cos\left(2\pi f_c t\right) \tag{7}$$

where f_c is the carrier frequency. From the modulation property of the Fourier Transform, we know that if the original Fourier Transform of $m(t)$ is $M(f)$, then the Fourier Transform of $x(t)$ is

$$X(f) = \frac{1}{2}\left[M(f - f_c) + M(f + f_c)\right]. \tag{8}$$

See Figure 7.35 in your textbook for diagrams of how the modulation looks in the time and frequency domains.

Suppose that if I have several audio signals, each with a maximum frequency of 20 kHz. This means that the Fourier Transform before modulation will have all of its energy in the frequency band from -20 kHz to +20 kHz. After modulation, the modulated signal will have all of its energy contained in the frequency bands [$-f_c$ – 20 kHz, $-f_c$ + 20 kHz] and [f_c – 20 kHz, f_c + 20 kHz]. If I use the different audio signals to modulate different carrier frequencies, then I can transmit multiple signals at once. The primary requirement for the carrier frequencies is that they should be at least 40 kHz apart so that the individual modulated signals do not overlap.

Figure 1 shows a diagram of this idea for two signals. Because the two signals are placed on two different carrier frequencies, the receiver will be able to recover both. But first consider demodulation of a single signal like the one in the equations above. The goal of the receiver is to recover the message-bearing part of the signal $m(t)$ from the modulated signal $x(t)$. Again, Section 7.7-1 of your textbook explains how to do this. The first step is to multiply the received signal $x(t)$ by the carrier term. This step results in the signal

$$\begin{aligned} e(t) &= m(t)\cos(2\pi f_c t)\cos(2\pi f_c t) \\ &= \frac{1}{2}\left[m(t) + m(t)\cos(2\pi 2 f_c t)\right], \end{aligned} \tag{9}$$

which by linearity and the modulation property has the Fourier Transform

$$E(f) = \frac{1}{2}\left[M(f) + \frac{1}{2}\left(M(f - 2f_c) + M(f + 2f_c)\right)\right]. \tag{10}$$

Studying (10), we see that there is now a copy of the original message's Fourier Transform centered again at zero frequency (called *baseband*), but scaled by ½. There are also two other copies: one centered at $2f_c$ and another at $-2f_c$, and both scaled by ¼. The message signal can now be completely recovered by passing $e(t)$ through a lowpass filter that rejects the two copies that are not located at baseband.

Finally, we now see that if there are multiple signals, each located at a different carrier frequency, that we can recover any one we want by demodulating with the correct carrier frequency. Each carrier frequency will cause a different signal to be centered at zero frequency. We then pass the signal through a lowpass filter to reject other signals, and we've recovered the one we want.

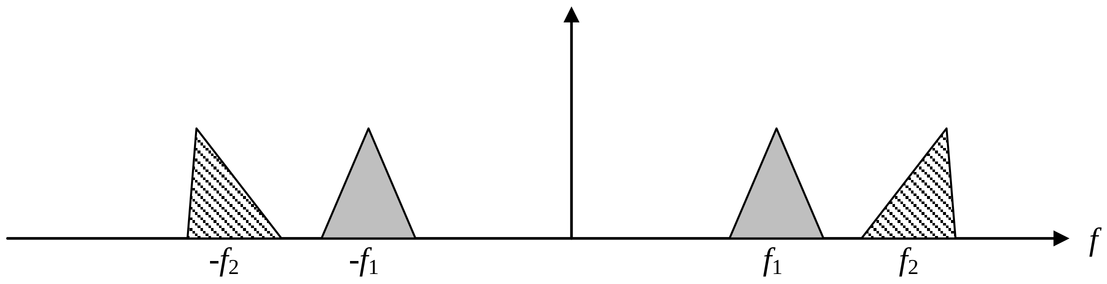

Figure 1. The Fourier spectrum of two signals (distinguished by different spectral shapes) modulating two different carrier frequencies.

In Part II, you will write a simulation that recovers a double-sideband, suppressed-carrier (DSB-SC) communications signal (see Section 7.7-1 in your textbook). I have taken eight different audio files and captured approximately 12-second snapshots from each. I then made one data file that has each of the eight audio files in it. Each audio signal is modulated to a different carrier using DSB-SC modulation. The carrier frequencies are 40 kHz, 80 kHz, 120 kHz, …, 280 kHz, and 320 kHz.

You should write a simulation that loads the data file and has carrier frequency as an input parameter. The simulation should recover the signal at the desired carrier by multiplying the data by a cosine with the desired carrier, then filtering the signal to capture the signal of interest. You can perform the lowpass filtering by convolving the signal (after multiplying by $\cos(2\pi f_c t)$ with the filter's impulse response using Matlab's "conv" command. You must design a lowpass filter using the window truncation techniques from Part I. The quality of your recovered signal will depend on the quality of your filter.

After performing these steps, you can listen to the signal using Matlab's "sound" command. For more information on this command, type 'help sound' at Matlab's command prompt.

In this section, we are studying an analog modulation technique. However, any data that I give you in a computer file must necessarily be digital. I can't provide you with analog signals unless I give them to you on something like a magnetic audio tape, which isn't very practical. And even then, you would have to sample and quantize the signals if you want to process them on a digital computer. So what I have done is to give you signals that are sampled much higher than the carrier frequency.

Your program should allow you to "tune" to one of the carriers and recover the audio signal on that carrier. To properly demodulate you need a perfectly synchronized carrier. To do the modulation, I used a cosine function with zero initial phase according to $\cos\left(2\pi f_c t\right)$. You should use this exact same signal to demodulate; therefore, you not only need to know the phase of the carrier, but how my time array was defined.

My time array starts at $t = 0$. In order to stay in phase, you will also need to start your time array at $t = 0$. The sampling rate for this data file is F_s = <u>705.6 kHz</u>. So you should make a time array that starts at 0, increments by $T_s = 1/F_s$, and is the same length as the input data. (Note: <u>do not</u> use a "for" loop to make the time array. You will be waiting all day. Matlab is optimized for matrix and vector operations. Using for loops is very inefficient in Matlab.) As mentioned above, the carrier frequencies are spaced by 40 kHz.

Use your Fourier Transform script from Part I to analyze the frequency spectrum of the original DSB-SC signal. Then, compare the spectra for some of the downconverted signals before and after lowpass filtering. Does the filtered spectrum look like the bandpass spectrum at the corresponding carrier frequency?

Try using a low order lowpass filter. What happens if the LPF isn't very good? Also, what happens if you add a phase of 90 degrees to the carrier?

Part III. Name That Tune from Dr. Goodman's Childhood

There are eight songs from Part II of this project. All of the songs were popular around Dr. Goodman's grade-school years. If you would like, try to name all eight artists and song titles.

Finally, some additional notes:

The data files for this project will be distributed by posting to the web. The files are in 32-bit floating-point binary format. The files were written using Matlab on a Windows PC, so if you are using something other than Windows on an Intel-type processor, there may or may not be "endian" issues. To load them into Matlab, do the following commands:

```
fid = fopen('filename.bin','rb');
a = fread(fid,'float32');
fclose(fid);
```

The first command opens the data file for reading in binary format. The second command reads all the data in the file. It also tells Matlab that the data was stored in 32-bit floating-point format. The third line closes the data file. For debugging your code, you may want to generate some shorter data files that won't take as long to process. If you actually want to store a shorter file on disk, then you can use the fwrite command. Or, you can just add the following two lines to obtain the beginning of the data file and clear the original full-length signal:

```
b = a(1:iend);
clear a;
```

(Note that you will need to define the variable iend to be, say, 1/5 or 1/10 of the original signal length). After debugging, you can go back to using the full-length data record for your analysis and listening pleasure.

Some commands that might be useful for this project include: flipud, fliplr, log10, fid, fwrite, fread, fclose, axis, xlabel, ylabel, title, figure, plot, subplot, abs, sound, soundsc, help.