Laboratory Project M3:
Fourier Analysis and Filtering of Sound Waveforms

***Abstract*-You will use Fourier theory to analyze the frequency content of sounds. Using plots, you will examine the effects of filtering on frequency content.**

I. PREPARATION

Read *Matlab Primer* Chapters 1 and 2, pages 4-1 to 4-27, and 4-34 to 4-35.

II. LEARNING OBJECTIVES

1) Use a truncated Fourier series to approximate a waveform as a sum of sinusoids.
2) Use a Fast Fourier Transform (FFT) to find the frequency content of a waveform.
3) Use a function of time to create a frequency modulated chirp signal.
4) Use a filter to change the frequency content of a sound.
5) Use an impulse function to characterize a filter.
6) Use noise to characterize a filter.
7) Use a spectrogram to plot changes in frequency content of a signal over time.

III. PROCEDURE

A. Lab Report

In this lab, you will write several script files. At the conclusion of the lab, send your TA an email whose body is the contents of all these script files and other information as specified below. That is, put all the script file contents together, put them in the body of the email unless the TA instructs you to do otherwise. Use comments in the script files to identify them and how they work.

Show your script files (lab report) to your TA before leaving the lab.

B. Fourier Series for Triangle Wave

Create a script file called fourier_series.m for this part of the lab.

fourier_series.m will use the following variables set beforehand:

```
1)      fund_f    % Fundamental frequency of Fourier series (and repetition rate of waveform)
2)           a    % Array of coefficients of Fourier series; (sizes of sinusoids to be summed)
```

In fourier_series.m, create an array of 8000 time values called *t* spaced by 1/8000 sec from 0 to just less than 1 sec. Using *t*, create cosines of amplitude a(1) for frequency fund_f, amplitude a(2) for frequency 2*fund_f, etc. Recall that a cosine of amplitude *A* and frequency *f* is created by the function $A\cos(2\pi ft)$. Hint: use a product of a vertical array containing 1 to *n* (where *n* is the number of entries in *a*) and *t*. The first row of this array will be the *t* array, the second row of this array will be two times the *t* array, and so on. Take the cosine of this array, and sum the resulting cosine waveforms to create a waveform called *y*. In the statement that produces *y*, use a semicolon to suppress the printing of *y*.

Create a script file called fft_plot.m that creates two plots:

1) In figure(1) it plots the first 100 samples of *y* versus *t*

2) In figure(2) it plots the magnitude (absolute value in Matlab®) of the FFT of *y* versus values of frequency from 0 to just less than 8000 Hz. (fft_plot.m will calculate the spacing of frequencies using 8000/((length(y)+1).) We refer to this plot as the spectrum plot.

Add commands to create appropriate labels for the *x*-axis and *y*-axis, and add a title to the plot.

After creating the script files, create a triangle waveform as follows:

1) Set fund_f to 100
2) Set array *a* to (Fourier coefficients) $a_1 = 1$, $a_2 = 0$, $a_3 = 1/9$
3) Run fourier_series.m
4) Run fft_plot.m

You should see one cycle of the approximate triangle waveform in figure(1) and the fft (or spectrum) of the triangle waveform in figure(2). The spectrum will show the two frequencies of 100 and 300 Hz in the triangle waveform. You will also see "aliased" frequencies of 7700 and 7900. It turns out that when these frequencies are sampled 8000 times a second, they produce exactly the same sample values as 300 and 100 Hz. So the FFT cannot distinguish between frequency f and frequency 8000 - f. Thus, the FFT is only able to faithfully report frequency content below 8000/2 = 4000 Hz.

Listen to the triangle wave. Technically, it is a chord rather than a pure tone.

C. Spectral Analysis of Speech

For this part of the lab, load the "handel" snippet (into *y*) using software from previous labs. Set *y* equal to samples 21000 to 26000 from the "handel" snippet. This part of the sound file is a vowel sound. The task in this part of the lab is to use spectral analysis to determine which of several vowels this part of the sound file is. (You could listen to the sound, but see if you can solve this puzzle using spectral techniques first.)

Run your fft_plot script to see the spectrum of your vowel sound. Compare the spectrum to the published spectra for several vowels available at

http://www.sail.usc.edu/~lgoldste/General_Phonetics/Gestural_structure/MRI_Vowels/index.html

(Note that the spaces in the URL are actually underscores.)

Write a comment in your lab report file saying what vowel you think the snippet is, and then listen to the snippet and comment again on what vowel you think the snippet is. Note that spectral analysis is what is done in speech recognition software, but it is combined with large databases on the probabilities of observed sequences of snippets (which are called phonemes). It is easy to see that spectra of speech sounds may be highly variable, making recognition, especially in the presence of noise a great challenge.

D. Creation of Chirp Sound

When we want to create a pure tone of frequency f, we use a cosine waveform such as

$$y = \cos(2\pi ft).$$

The spectrum of the cosine at a fixed frequency would always show a line at frequency f. In speech and other sounds, however, the spectral content is constantly shifting. Consequently, we use short-time spectral analysis and shift from one snippet to another as time proceeds. A spectrogram is a plot of the changing spectrum. We can easily create a waveform with a changing spectrum by making the frequency a function of time. In this part of the lab, you will create a chirp sound by steadily increasing the frequency as time progresses.

Create a script file called chirp_wave.m that sets array *y* to a chirp sound generated as follows:

1) It uses the array *t* defined earlier in this lab. (That it, your script file assumes the *t* array is predefined outside the script file.)
2) The frequency, $f(t)$, starts at 200 Hz and increases by 2000 Hz over one second.
3) Set $y = \frac{1}{20}\cos(2\pi f(t)t)$.

Run your script file and listen to the waveform. You will hear the frequency increasing.

E. Filtering

Matlab® makes it easy to create filters that alter the spectrum of a sound. For example, the following command creates a 4th-order Butterworth bandpass filter that allows frequencies between 0.4 and 0.6 times half the sampling rate of the signal, which in our case is 8000/sec.

```
[b,a] = butter(4,[0.4,0.6]);  % Designs 4th order Butterworth bandpass
                              %  from 8kHz/2 * (0.4 to 0.6)
```

The returned arrays, *a* and *b* are the feedforward and feedback portions of a digital filter. To filter the sound in *y*, we use the following command:

```
y = filter(b,a,y);
```

Listen to the filtered *y* sound. Write a comment describing how it differs from the original *y* sound. Given that the filter only passes frequencies in a certain band, is the sound what you expect?

There are two common ways of characterizing a filter: via the impulse response, and via power spectral density. In the case of the impulse response, we create an impulse signal whose spectrum is the same across all frequencies. We then run this impulse through the filter and calculate the spectrum of the output. This spectrum is the filter's frequency response. If we run a signal, *y*, through the filter, the spectrum of the output will be the spectrum of *y* times the frequency response.

Find the frequency response of the Butterworth filter as follows:

1) Create an array, *y*, of 8000 zeros and then set y(1) to 1.
2) Filter *y* using the filter command and the Butterworth *a* and *b* as above.
3) Use your script file, fft_plot.m, to plot the spectrum of the filter output.
4) Write a comment on the shape of the frequency response of the filter.

In the case of power spectral density, we create a noise waveform via the randn( ) function. We calculate the correlation of this noisy waveform with itself, which results in approximately an impulse function. (We omit the details here.) We pass the noisy signal through the filter and find the spectrum of the output. This spectrum is approximately the frequency response of the filter. (Again, we omit some details and simplify the facts here.) As before, if we run a signal, *y*, through the filter, the spectrum of the output will be the spectrum of *y* times the frequency response.

Find the approximate frequency response of the Butterworth filter as follows:

1) Create an array, *y*, of 8000 noise samples using the randn( ) function.
2) Filter *y* using the filter command and the Butterworth *a* and *b* as above.
3) Use your script file, fft_plot.m, to plot the (approximate) spectrum of the filter output.
4) Write a comment on the shape of the frequency response of the filter.

Note that filters are one way of altering sounds in somewhat predictable ways. That is, we can understand their effect because we are familiar with frequency response from using equalizers (or treble and bass controls) with music players.

F. Spectrogram (Extra Credit)

The chirp signal is a simple example of a sound whose frequency content changes over time. To analyze such sounds, the spectrogram is a convenient tool. Matlab® has a spectrogram command that automatically creates a plot of ffts of successive segments of a waveform. In order to learn about 3-D plotting, however, we will create our own spectrogram script file, which we will call "spectgram.m". This script file has the following content:

1) A command to extract the first 1024 (= 32 x 32) samples in *y* (leaving the result in *y*).
2) A reshape( ) command to create array *s* that is *y* reshaped into a square 32 x 32 array.

3) An fft( ) command to take the FFT of each column of *s*.
4) Commands to create a 32 x 32 meshgrid of frequencies 0 to 8000 spaced by 8000/32 in one direction and frame numbers 1 to 32 in the other direction. The values returned by meshgrid( ) will be called xx and yy. Be careful about which variable should be frequencies and which should be frame numbers.
5) A command to create a 3-D lit surface plot of the FFT's versus frames and frequencies.
6) Commands to add appropriate axis labels for *x*, *y*, and *z*.

Run spectgram.m on the chirp sound. Write a comment on whether the plot is you expect or not and why.

REF: [1] The Mathworks, Inc, *Matlab® Primer,* Natick, MA: The Mathworks, Inc, 2012.