BEE 235 Lab #3

Fourier Series and Gibbs Phenomenon

(Modified based on the modules created by: University Of Washington Dept. of Electrical Engineering.)

Summary: Fourier series, sums of cosines. This development of these labs was supported by the National Science Foundation under Grant No. DUE-0511635. Any opinions, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Introduction

In this lab, we will look at the Fourier series representation of periodic signals using MATLAB. In particular, we will study the truncated Fourier series reconstruction of a periodic signal.

Some Useful MATLAB Commands

- `abs`, compute the complex magnitude.
- `angle`, compute the phase angle.
- `clear`, clears all variables.
- `help <command>`, online help.
- `whos`, list all variables and their sizes.
- `fft`, Fourier transform of a signal.

Part #1 Signal Synthesis

We will see that we can approximate a square wave with the Fourier series, but first let us approximate something more interesting, say a musical instrument? Many instruments produce very periodic waveforms.

Synthesizer

1. Create a script file called `signal_synthesis.m` to put your code in for this problem.
2. Download the trumpet sound sample `trumpet.mat` from the course webpage. The sample rate, `Fs`, of the trumpet is 11,025 Hz. Play this sound with the `sound` command (remember to include the correct sample rate).
3. Plot only a small section of the trumpet sound to show three or so periods (try 150 samples or so).

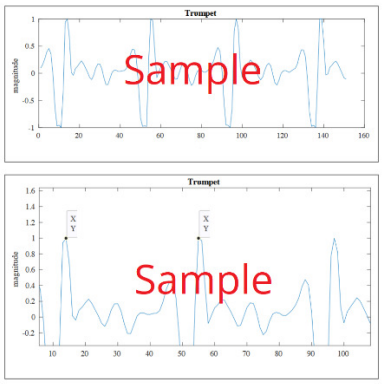
   Note: to plot a small section of a signal `ss` (vector), you can use:

Index=1000; % the starting index number
plot(ss(index : index+150)); % 150 samples starting from the 1000th element in ss

Note: you need to change **ss** to **trumpet.**

Move the data cursor(+) to one peak first and then to another adjacent peak (to cover just one period) on your figure to find the number of samples over one single period. Record these numbers in the following table. Then, calculate its fundamental frequency F0 based on the given sample rate **Fs**.

Peak No.1	Peak No.2	Period (# of samples)	F0 (Hz) =Fs/(# of samples in one period)
X1= Y1=	X2= Y2=	|X2-X1|	



4. View the frequency spectrum of the sound by entering the following command:

```
Fs = 11025;                % our sample rate is 11025 Hz
Y = fft(trumpet, 512);     % FFT (Fourier transform) of the trumpet sond
Ymag = abs(Y);             % take the mag of Y
f = Fs * (0:256)/512;      % get a meaningful frequency axis
plot(f, Ymag(1:257));      % plot Ymag (only half the points are needed)
xlabel('Frequency (Hz)');
ylabel('Magnitude');
```

You should now see a series of peaks (these are the harmonics of the instrument).

Type **>>help fft** in your command window to see what **fft** does in Matlab.

5. You can use the "data cursor" tool in MATLAB's figure window to easily read graph data. Write down the frequency and its strength (magnitude) for the first **7 harmonics (**first 7 peaks**)**.

harmonics	Magnitude	Frequency (Hz)
1		

2		
3		
4		
5		
6		
7		

Determine the fundamental frequency F0 from the table above. You can estimate F0 by finding the frequency differences between any two neighboring harmonics and then average them. Does this match the F0 frequency that you found previously? What is the relationship between these harmonics?

6. We will now synthesize the instrument using only the peak information. You can use the "data cursor" tool again in MATLAB's figure window to easily read graph data. Write down the frequency and its strength (magnitude) for **ten** of the **strongest** peaks in descending order of intensity.

Peaks	Magnitude	Frequency (Hz)
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

7. In the script file **signal_synthesis.m,** define three vectors using the values from the table above: time vector `t`, frequency vector `freq`, and magnitude vector `mag`. Have your function use a **for-loop** to add together cosines, one for each frequency/magnitude pair in the `freq` and `mag` vectors. Remember to normalize your output vector after you add up all the cosines (the output should be between -1 and 1) (e.g. **ss=ss/max(abs(ss));).** Use the data you collected from the frequency plot of the trumpet sound with your new function to sum cosines at the noted frequencies.

   For example, if you had two harmonics, one at 100 Hz with magnitude 1 and another at 150 Hz with magnitude 2, then your vectors will be,

```
t = 0:1/Fs:2;  % two-second time vector at Fs
freq = [100 150]; % for example: 2 peaks
mag = [1 2];
```

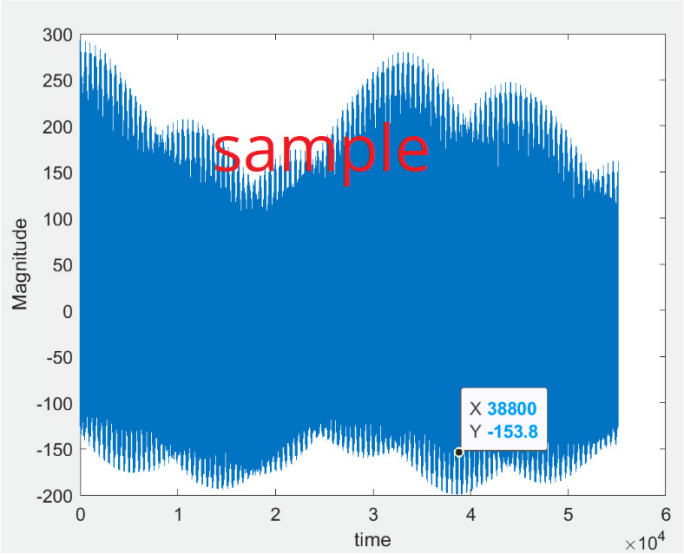
   Here are some hints for the above. Use a for-loop to create a cosine at each frequency in the freq vector. Your cosine function should look something like this:

```
mag(i)*cos(2*pi*freq(i)*t);
```

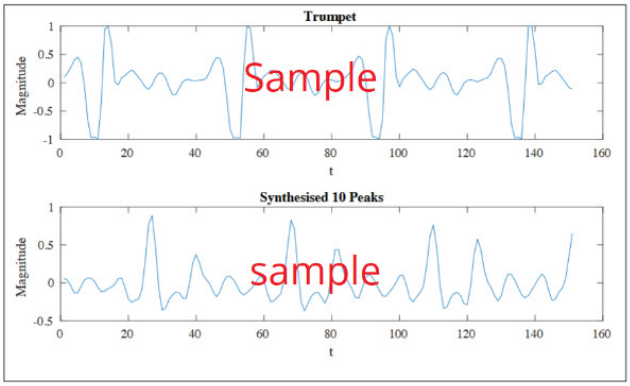
Remember your time vector will have the form **0:1/Fs:time_in_seconds**.

8. Play the original trumpet and your new synthesized sound with 10 peaks. Do they sound the same? Use subplot to plot a small section of your new synthesized sound along with the trumpet sound, does it look the same? Save your plot.

   The entire synthesized signal:



   A small section:



Part #2 Truncated Fourier Series

In this section, we'll reconstruct the periodic function `x(t)`, shown in Figure 1, by synthesizing a periodic signal from a variable number of Fourier Series coefficients and observe similarities and differences in the synthesized signal.

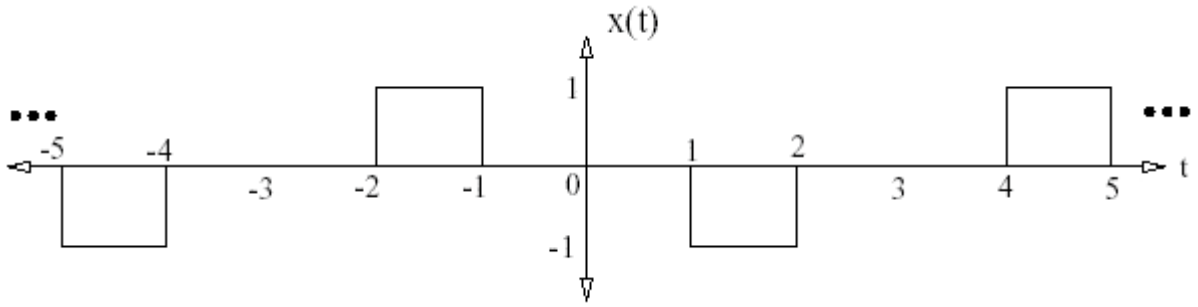


Figure 1: Periodic Signal

Gibbs phenomena

1. Create a script file called **`gibbs.m`** to put your code in for this problem.
2. Go to the course webpage to download the MATLAB function **`Ck.m`**. Take a look at the contents of the function. This function takes one argument k, and creates the k th Fourier series coefficient for the squarewave above:

$$C_k = \begin{cases} 0 & \text{if } k = 0,\ k \text{ even} \\ \dfrac{1}{jk\pi}\left[\cos\left(\dfrac{2k\pi}{3}\right) - \cos\left(\dfrac{k\pi}{3}\right)\right] & \text{if } k \text{ odd} \end{cases} \quad (1)$$

$C_k(1) = \frac{-1}{j\pi} = 0 + j0.3183$. Plot the magnitude and phase of the coefficients C_k for $k \in \{-10, -9, \ldots, 9, 10\}$. The magnitude and phase should be plotted separately using the subplot command, with the magnitude plotted in the top half of the window and the phase in the bottom half. Place frequency ω on the x axis. Use the MATLAB command **`stem`** instead of **`plot`** to emphasize that the coefficients are a function of integer-valued (not continuous) k. Label your plots.

`gibbs.m`

```
...
k = -10:1:10;
% call Ck function and generate your plots
...
```

3. Write whatever script/function files you need to implement the calculation of the signal x(t) with a truncated Fourier series:

$$x(t) = \sum_{k=-K_{\max}}^{K_{\max}} C_k e^{\mathrm{j}k\omega_0 t} = \sum_{k=0}^{K_{\max}} 2|C_k| \cos\left(k\omega_0 t + \angle C_k\right) \quad (2)$$

for a given K_{max}

Note:

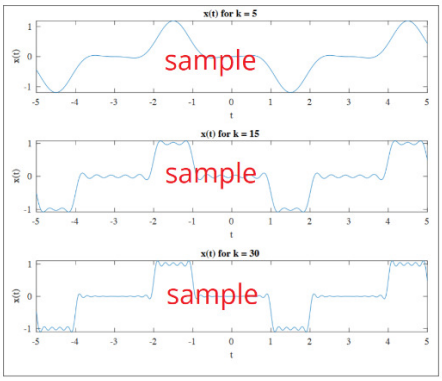
You can avoid numerical problems and ensure a real answer if you use the cosine form. For this example, $\omega_0 = \pi/3$.

4. Produce plots of x(t) for $t \in [-5,5]$ for each of the following cases: K_{max} = 5; 15; and 30. For all the plots, use as your time values the MATLAB vector `t=-5:.01:5`. Stack the three plots in a single figure using the `subplot` command and include your name in the title of the figure.

```
Kmax=5; % you can also create a new function with an input argument of Kmax
t=-5:.01:5;
sum=0;
for k= 0:Kmax

        % call Ck function
        % generate the cosine components as specified in equation (2) above
        % add to the sum;

end;
```



As you add more cosines you'll note that you do get closer to the square wave (in terms of squared error), but that at the edges there is some undershoot and overshoot that becomes shorter in time, but the magnitude of the undershoot and overshoot stay large. This persistent undershoot and overshoot at edges is called Gibbs Phenomenon.

In general, this kind of "ringing" occurs at discontinuities if you try to synthesize a sharp edge out of too few low frequencies. Or, if you start with a real signal and filter out its higher frequencies, it is "as if" you had synthesized the signal from low frequencies. Thus, low-pass filtering (a filter that only passes low-frequencies) will also cause this kind of ringing.

For example, when compressing an audio signal, higher frequencies are usually removed (that is, the audio signal is low-pass filtered). Then, if there is an impulse edge or "attack" in the music, ringing will occur. However, the ringing (called "pre-echo" in audio) can be heard only before the attack, because the attack masks the ringing that comes after it (this masking effect happens in your head). High-quality MP3 systems put a lot of effort into detecting attacks and processing the signals to avoid pre-echo.

Your lab report should include a copy of all your script files `and` all the plots required.