University of Washington
Department of Electrical Engineering
EE 235 Lab 6:
Applications of Fourier Transforms

In this lab (final lab, yay!), we will investigate two important Fourier Transform applications: filtering and modulation. We will apply both of these to a real world problem of creating and decoding a message encoded using Morse Code.

Important Concepts From Lecture You Will Use In Lab 6

- Classifying filter type using magnitude response of LTI systems
- Using the convolution property of Fourier Transforms
- Using the modulation property of Fourier Transforms

What Is Expected From You In Lab 6

- Completion of 4 pre-lab exercises (5 points)
- Completion of 2 in-lab check offs with TA (5 points)
- Completion of a lab report (10 points)

EXERCISE #1: Filtering

In this exercise, we will learn two ways to implement a filter in Matlab. We will use these techniques in the last two exercises.

1) ***Pre-Lab Analysis***:
   a) Consider the following low-frequency signal **x(t) = cos(0.1t)**. Find its Fourier Transform **X(jw)**.

   b) Suppose a real-valued LTI filter has the frequency response: $\boldsymbol{H(jw)} = \frac{\mathbf{3}}{\mathbf{3}+\boldsymbol{jw}}$
      i) Evaluate the magnitude response **|H(jw)|**.
      ii) Draw a rough sketch of **|H(jw)| vs. w**.
      iii) Based on your sketch in (ii), classify the filter type.
      iv) Do you expect this filter to pass or reject the low-frequency signal in (a)?

   c) Using **X(jw)** and **H(jw)** from parts (a) and (b), as well as the convolution property, find the output Fourier Transform **Y(jw)** and show that
      $$Y(jw) = \frac{3\pi}{3+j0.1}\delta(\omega - 0.1) + \frac{3\pi}{3-j0.1}\delta(\omega + 0.1)$$
      Hint: you will need to use the sampling property for unit impulses to get the final answer.

   d) **Read Background Section (3) on filters** and then answer the following question: Consider the frequency response used in (b), what should the filter coefficients **b** and **a** be for the function **lsim**?

   e) **Read Background Section (4) on multiplication and division** and then answer the following question: Suppose you have the following Matlab variables

$x1 = 2$ and $x2 = [1 \;\; 3]$

Which of the following operations are valid in Matlab?
i) x1 * x1, x1 .* x1, or both?
ii) x2 * (x1 + x2), x2 .* (x1 + x2), or both?
iii) x1 * (1 + x2), x1 .* (1 + x2), or both?
iv) x1 / (2 * x1), x1 ./ (2 * x1), or both?
v) x1 / (1 + x2), x1 ./ (1 + x2), or both?

f) **Read the Background Section** on these topics to prep for the upcoming lab exercise

Matlab Concepts/Functions to Review	New Matlab Concepts/Functions
• Generating signal in time domain • Computing magnitude of FFT • Plotting with subplots	• Labeling a figure window • Rules for multiplication and division • Using inverse transform **ifft** function • Compute output of system with **lsim** function

2) ***Lab Exercise***:
a) Create a new working directory in your U Drive called **Lab 6**. In this directory, open up a script and call it **Ex1.m**. Make sure to clear all variables and close all figures.

b) We will first generate the input signal in the time and frequency domains:

i) Generate signal $\boldsymbol{x(t) = \cos(0.1t)}$ from 0 ≤ t < 500 using a sampling rate **Fs** = 10. Store the time samples in **t** and the signal in a variable called **x**.

ii) Compute the magnitude of the FFT of **x** using **N** = 8192 frequency samples. Store this result in variable **x_abs**.

iii) Compute the corresponding frequency samples **w** using **N** and **Fs**.

iv) Load figure window #1. **Refer to Background Section (5) on labeling figure windows, if needed**. Using a 2 x 2 subplot window, plot:
- 1st subplot: **x vs. t** with x-axis limits [100, 400] and y-axis limits [-2, 2]
- 2nd subplot: **x_abs vs. w** with x_axis limits [-0.5, 0.5] and y-axis unchanged

We will plot two more signals later in the lab exercise.

v) Run your script and verify your FFT plot is consistent with your equation for X(jw) in pre-lab to within a scaling factor.

c) Let us now compute the frequency response of the filter:

i) Using the same frequency samples **w** that was used for **x_abs**, generate the frequency response of the filter $\boldsymbol{H(jw) = \frac{3}{3+jw}}$. Make sure to use the correct division operator. **Refer to Background Section (4), if needed**. Store the values in the variable **H_fft**.

ii) Using the **abs** function, compute the magnitude of the FFT and store it in **H_abs**.

iii) Load figure window #2, and plot **H_abs vs. w**. Adjust the x-axis to be between -25 and 25, and then leave the y-axis unchanged. Title and label your plot.

iv) Run your script and verify your plot of **H_abs** matches with your pre-lab.

d) Let us now compute the output signal **y(t)** using frequency domain methods first.

i) Recall the convolution property: Y(jw) = X(jw)H(jw). Using **x_fft** and **H_fft**, compute the output Fourier Transform **y_fft**. Make sure to use the correct multiplication operator.

ii) We can now convert the signal **y_fft** back to the time domain with an inverse Fourier Transform. Using the functions **ifft**, **fftshift**, and **real**, compute the time domain signal **y**. **Refer to Background Section (2) on how to compute an IFFT**.

iii) Compute the corresponding time samples vector **t_y** for the signal **y**.

iv) Go back and load (or access) figure window #1. Plot **y vs. t_y** as the 3rd subplot. Adjust the limits of the x-axis to be between 100 and 400, and the y-axis to be between -2 and 2. Title and label your plot.

v) Run your script and verify your plot for **y** is consistent with your answer to the question in pre-lab as to whether the filter passes or rejects the input signal **x(t)**.

e) Let us now compute the output signal **y(t)** using a special function in Matlab called **lsim**:

i) Using your pre-lab, define the filter coefficients **b** and **a** for the filter **H(jw)**. **Refer to the Background Section (3) on filters, if needed**.

ii) Using the **lsim** function, pass the input signal **x** (along with its time samples **t**) through the filter defined by coefficients **b** and **a**. Call the output of the filter **y**.

iii) Access figure #1 again and plot **y vs. t** as the 4th subplot. Use the same limits as the 3rd subplot. Title and label your plots.

f) Comment your code, and then run your script to view output plot **y**. Verify it matches the plot from (d).

3) ***Lab Check-Off #1 of 2***: Show your plots to a Lab TA and demonstrate you know how to implement a filter and compute its output in Matlab.

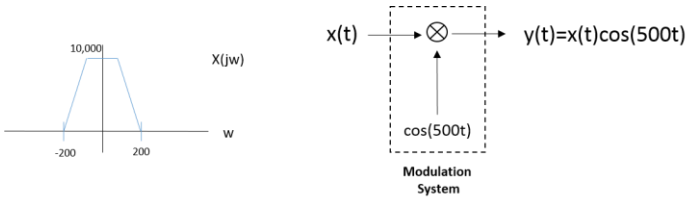
4) ***Lab Report Question #1***: A rushed student forgets to call the **figure** function with 1 argument, and instead types in the **figure** command with no function argument. Describe what would change if you ran **Ex1.m** again. Does Matlab produce the same number of figure windows? Do any of the figures change? If so, how do they change?

EXERCISE #2: Amplitude Modulation

In this exercise, we will shift the focus to another application of Fourier Transforms: communication systems and modulation. Modulation, in general, is an important part of all communication systems. In short, modulation is a process that will transform a signal of low frequency to a signal of higher frequency. Transforming a signal to a higher frequency signal is important because low frequency signals cannot travel long distance over the air, but higher frequency signals can.

We will use this exercise to implement and analyze amplitude modulation in Matlab. The signals we will experiment with will be our representation of dash and dot signals used in International Morse Code.

1) ***Pre-Lab Questions***:
   a) Consider the modulation system below, with input **X(jw)** provided on the left:



   Using Fourier Transform properties, find and expression for **Y(jw)** in terms of **X(jw)**. Then, using the plot for **X(jw)** above, sketch **Y(jw) vs. w**

   b) You should notice that **X(jw)** (originally centered around **w = 0**) is now centered around two higher frequencies in **Y(jw)**. These two frequencies in communications are called the carrier frequencies. What are these two carrier frequencies $\mathbf{w_c}$ in **Y(jw)**?

   c) In Matlab, suppose you are given the row vector **x** for the input signal and the row vector **t** for the time samples. How would you implement the multiplication between **x(t)** and **cos(500t)** for **y(t)**? Would you use matrix multiplication * or array multiplication .* ?

   d) Consider the International Morse Code table below:

A	· -	H	· · · ·	O	- - -	V	· · · -
B	- · · ·	I	· ·	P	· - - ·	W	· - -
C	- · - ·	J	· - - -	Q	- - · -	X	- · · -
D	- · ·	K	- · -	R	· - ·	Y	- · - -
E	·	L	· - · ·	S	· · ·	Z	- - · ·
F	· · - ·	M	- -	T	-		
G	- - ·	N	- ·	U	· · -		

   Suppose in Matlab you are given the prototype signals **dash** and **dot**. Using these variables, how would you construct the signal **x** for letter **X**? That is, fill in the blanks:

   x = [_____ _____ _____ _____]

e) **Read the Background Section** on these topics to prep for the upcoming lab exercise:

Matlab Concepts/Functions to Review	New Matlab Concepts/Functions
• Create time samples vector • Find magnitude of FFT of signal • Plotting with subplots • Concatenating vectors	• Multiplication rules

2) ***Lab Exercise***:
   a) Open up a script file and call it **Ex2.m**. Clear all variables and close all figures.

   b) From the class website, download and load the Matlab data file **MorseCode.mat**. This file will contain the following signals:
      - **Fs**: sampling rate of signals
      - **dash**: first prototype signal for International Morse Code
      - **dot**: second prototype signal for International Morse Code

   c) Create the time samples vector **t_morse** using sampling rate **Fs** and the length of either **dash** or **dot**. Both signals are the same length, so using either prototype signals will work.

   d) To view the two Morse code signals we will be using throughout Lab 6, plot the two prototype signals using a 2 x 1 subplot as follows:
      - **dash vs. t_morse**: x-axis unchanged, but y-axis range = [-40, 40]
      - **dot vs. t_morse**: x-axis unchanged, but y-axis range = [-40, 40]

   e) Now let us use Matlab to construct one letter in the International Morse Code table:

      i) Using results from pre-lab, use vector concatenation of the signals **dash** and **dot** to construct the Morse Code signal for the letter X. Call this variable **x**.

      ii) Create the corresponding time samples vector **t** using **x** and **Fs**.

      iii) Using the functions **fft**, **fftshift**, and **abs**, compute the magnitude of the Fourier Transform of signal **x** using **N** = 8192 frequency samples. Call this variable **x_fft**.

      iv) Create the vector of frequency samples **w** using the number of frequency samples **N** and the sampling rate **Fs**.

      v) Using a 2 x 1 subplot, plot the signal **x** in the time and frequency domain as follows:
         - **x vs. t**: leave both x-axis and y-axis unchanged
         - **x_abs vs. w**: x-axis range = [-1000, 1000] and y-axis unchanged

      vi) Run your script and verify your plot for **x_abs**. The envelope should be similar to the envelope of the signal used in pre-lab.

f) Now we will implement amplitude modulation in Matlab:

i) Generate the signal **y(t) = x(t)cos(500t)** using the signal **x** and the same time samples vector **t** used by signal **x**. Call the resulting signal **y**. Note that both **x** and **cos(500t)** are row vectors, so make sure to use the correct type of multiplication. **Refer to Background Section (4), if needed**.

ii) Using the functions **fft**, **fftshift**, and **abs**, compute the magnitude of the Fourier Transform of signal **y** using **N** = 8192 frequency samples. Call this variable **y_fft**. Note: you do not need to compute a separate frequency samples vector since **N** and **Fs** are the same as the input signal.

iii) Using a 2 x 1 subplot, plot the signal **y** in the time and frequency domain as follows:
- **y vs. t**: no need to adjust axes
- **y_abs vs. w**: x-axis range = [-1000, 1000] and y-axis range = [0, 10000]

iv) Run your script and verify your plot for **y_abs** and the location of the carrier frequencies $\mathbf{w_c}$.

3) ***Lab Check-Off #2 of 2***: Show your script **Ex2.m** to a Lab TA and demonstrate you understand how to modulate a signal.

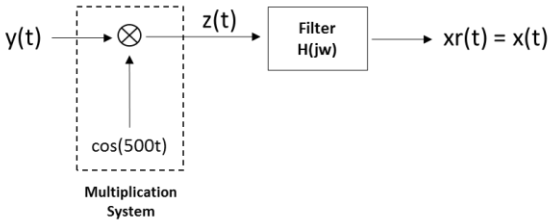
4) ***Lab Report Question #2***: A student accidentally uses the modulation signal **cos(50t)** in Matlab instead of **cos(500t)**. The student claims s/he sees the exact same graph for **Y(jw)** with a modulation frequency $\mathbf{w_M}$ = 50 as his/her pre-lab. Explain why this student is incorrect. Also, explain why using any carrier frequencies where $\mathbf{w_M}$ < 200 will not work.

5) ***Lab Report Question #3***: When graphing the three plots in this exercise, the student uses the line of code **figure(1)** at all times instead of using the command **figure**. What changes do you expect when you run **Ex2.m** again?

EXERCISE #3: Amplitude Demodulation

In this exercise, we will learn how to do undo the modulation operation in Exercise #2. This process in communication systems is called demodulation. Once a signal is transmitted to the intended receiver, demodulation helps to undo that transformation and recover the original signal for the receiver.

1) ***Pre-Lab Questions***: Demodulation is performed using a two-stage process, as shown below.



The first stage requires another multiplication operation using the same sinusoidal signal that was used for modulation. The second stage, on the other hand, requires an additional filter to fully recover the signal **x(t)**. The purpose of these two stages is to recover the original **X(jw)**, which is centered on **w = 0**.

a) Suppose **y(t) = x(t)cos(500t)**, where **X(jw)** and **Y(jw)** are the same as in Exercise #2. Using your results from Exercise #2 and Fourier Transform properties, find an expression for **Z(jw)** in terms of **Y(jw)**. Using your sketch for **Y(jw)** in Exercise #2, sketch **Z(jw) vs. w**. You should notice that your plot for **Z(jw)** contains two parts:

- Original signal **X(jw)** centered around **w = 0** with amplitude scaled by 1/2.
- Other copies of **X(jw)** centered at higher frequencies

b) The low-pass filter we will use to recover **X(jw)** has the following LCCDE relating input **z(t)** to output **xr(t)**:

$$(240)\frac{d^4xr(t)}{dt^4} + (3\times10^4)\frac{d^3xr(t)}{dt^3} + (2.2\times10^6)\frac{d^2xr(t)}{dt^2} + (10^8)\frac{dxr(t)}{dt} + (2\times10^9)xr(t) = (2\times10^9)z(t)$$

i) Find the filter's frequency response $\boldsymbol{H(jw)} = \frac{\boldsymbol{Xr(jw)}}{\boldsymbol{Z(jw)}}$.

ii) What should the filter coefficients **b** and **a** be for the function **lsim**?

iii) What is the DC gain |H(j0)| of this filter?

c) **Read the Background Section** on these topics to prep for the upcoming lab exercise:

Matlab Concepts/Functions to Review	New Matlab Concepts/Functions
• Finding magnitude of FFT of signal • Plotting with subplots	• Computing output of system using **lsim** function • Multiplication rules

2) ***Lab Exercise***:

a) Open up a script file and call it **Ex3.m**. Clear all variables and close all figures.

b) From the class website, download and then load the data file **Ex3.mat**. It will consist of:
- **Fs**: sampling rate of signal y
- **t**: vector of time samples for signal y
- **y**: modulated signal from Exercise #2

a) We will begin by performing and analyzing the first step in the demodulation process:

i) Generate the output of the multiplier, **z(t) = y(t)cos(500t)**, from the system diagram in pre-lab, and call the signal **z**.

ii) Using the functions **fft**, **fftshift**, and **abs**, compute the magnitude of the Fourier Transform of signal **z** using **N** = 8192 frequency samples. Call this variable **z_abs**.

iii) Create the vector of frequency samples **w** using the number of frequency samples **N** and the sampling rate **Fs**.

iv) Using a 2 x 1 subplot, plot the signal **z** in the time and frequency domain as follows:
- **z vs. t**: leave x-axis and y-axis unchanged
- **z_abs vs. w**: x-axis range = [-1500, 1500] and y-axis range = [0, 10000]

v) Run your script and verify your plot of **z_abs** matches **Z(jw)** from pre-lab.

b) We will now perform the second and last step in the demodulation process, and filter out the original signal.

i) Define the filter coefficients **b** and **a** for filter **H(jw)** from pre-lab.

ii) Using the **lsim** function, pass the input signal **z** (along with its time samples **t**) through the first filter defined by coefficients **b** and **a**. Call the output of the filter **xr**.

iii) Using the functions **fft**, **fftshift**, and **abs**, compute the magnitude of the Fourier Transform of signal **xr** using **N** = 8192 frequency samples. Call this variable **xr_abs**.

iv) Using a 2 x 1 subplot, plot the following:
- **xr vs. t**: no need to adjust axes
- **xr_abs vs. w**: x-axis range = [-1000, 1000] and y-axis range = [0, 10000]

c) Comment your code, and then run your entire script and verify you were able to recover your original signal and that the signal matches the Morse Code representation for letter X to within a scaling factor.

3) ***Lab Report Question #4***: Observe that the FFT of the recovered signal **xr(t)** has a max value around 5000, but the FFT of the original signal **x(t)** in Exercise #2 had a max value around

10,000. Clearly, there is something wrong with the low-pass filter. Explain the possible mistake with the filter and how it should be modified for next time.

4) ***Lab Report Question #5***: In this exercise, we analyzed a system with input-output relationship **y(t) = x(t)cos(500t)**. A student in class claims this system is LTI. Explain why s/he is incorrect.

EXERCISE #4: Decoding a Morse Code Message

In this experiment, we will put together what we learned in the last three exercises to decode a Morse Code signal that consists of not just one letter, but rather three letters that make up an actual word. This lab exercise is a modification from an exercise in ***Computer Explorations in Signals and Systems using Matlab*** by John R. Buck, Michael M. Daniel, and Andrew C. Singer.

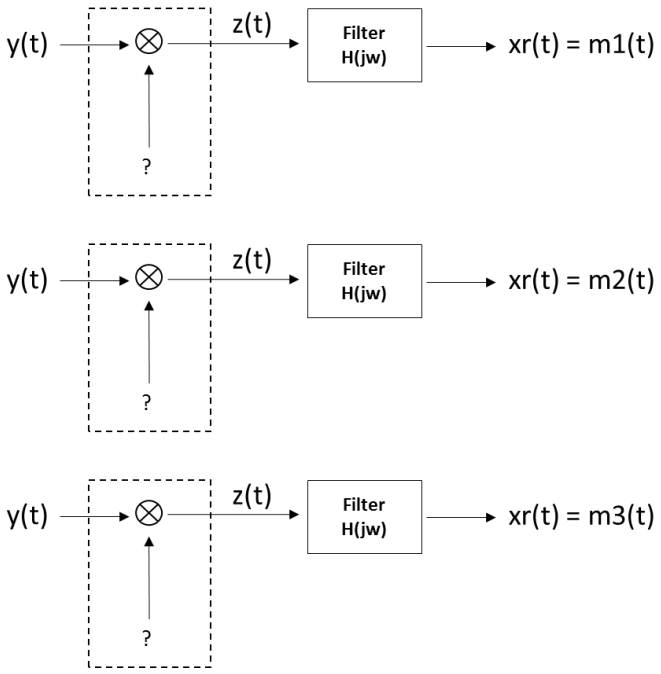
Here is the scenario for this week: You are given a signal **y(t)** that contains a simple message from Agent 007. When loading the file, you magically transform into Agent 008, the code-breaking sleuth. The last words of the aging Agent 007 were "The future of technology lies in…" at which point Agent 007 saved the remaining message to a Matlab data file and decided to return to school and give up the next four years to pursue the American dream of getting a PhD. Your job now is to decipher the message encoded in **y(t)** and complete Agent 007's final words.

1) ***Pre-Lab Questions***: Here is what is known about the signal. Signal **y(t)** is of the form

$$\boldsymbol{y(t) = m1(t)\cos(1000t) + m2(t)\cos(2000t) + m3(t)\cos(3000t)}$$

where **m1(t)**, **m2(t)**, and **m3(t)** each correspond to a single letter of the alphabet which has been encoded using International Morse Code. Note that this signal consists of a sum of three different modulated signals. This communication systems technique is called frequency division multiplexing.

a) In the last exercise, you learned how to recover a single signal **x(t)** from **y(t) = x(t)cos(500t)**. That same process can actually be used to individually recover **m1(t)**, **m2(t)**, and **m3(t)** using three separate systems, as shown below:



What are the three cosine signals you need for each system above?

b) To pick the correct cosine signal in Matlab, we will make use of a decision statement inside of a loop with index variable **i**. Write a decision statement for the scenario below. Assume you are already given the time samples vector **t**:
   - If index **i** is 1, then assign your first cosine signal to the variable **c**
   - If index **i** is 2, then assign your second cosine signal to the variable **c**
   - If index **i** is 3, then assign your third cosine signal to the variable **c**

   Refer to Matlab Review section, if needed.

c) Suppose the variable **i = 2**. Using this variable **i**, vector concatenation, and the function **num2str**, write the Matlab code to construct the variable **str** with value "Message m2(t)". **Refer to Matlab Review section, if needed**.

d) **Read the Background Section** on these topics to prep for the upcoming lab exercise:

Matlab Concepts/Functions to Review	New Matlab Concepts/Functions
• Loading a figure window • Plotting using subplots • Vector concatenation • Creating a string variable • Using for loops • Using decision statements	• Computing output of system using **lsim** function • Multiplication rules

2) ***Lab Exercise:*** You will now implement your design for the message recovery system.

a) Open a script file and call it **Ex4.m**. Clear all variables and close all figures.

b) From the class website, download and load the data file **Ex4.mat**. You should have:
   - **y**: message signal y(t)
   - **t**: time samples vector
   - **Fs**: sampling rate

c) We will perform the following tasks to prep for the upcoming loop:

   i) Load a figure window.

   ii) Define the filter coefficients **b** and **a** for the same low-pass filter you used in Exercise #3.

d) We will now implement the three systems using a for-loop. The outline for the for loop is as follows

```
% LOOP through exactly three times using the index variable i starting at i = 1

    % Implement your decision statement from pre-lab to decide which correct
    % cosine signal to use.  Store the cosine signal in variable c.
```

% Compute the output of the multiplier using variables **y** and **c**. Store the output
% in the signal **z**.

% Pass the signal **z** (and its corresponding time sample **t**) through your low-pass
% filter using function **lsim**. Store the output in the signal **xr**.

% Using a 3 x 1 subplot, plot **xr vs. t** as the **ith** subplot.

% Label the axes.

% Using a string variable, create the title of the plot so that it displays the
% message index number in the title. So if i = 1, then the title will be **"Message**
% **m1(t)"**.

% END OF LOOP

e) Run your entire Matlab script and make sure you get a 3 x 1 plot window.

3) ***Lab Report Question #6***:

A	· -	H	· · · ·	O	- - -	V	· · · -
B	- · · ·	I	· ·	P	· - - ·	W	· - -
C	- · - ·	J	· - - -	Q	- - · -	X	- · · -
D	- · ·	K	- · -	R	· - ·	Y	- · - -
E	·	L	· - · ·	S	· · ·	Z	- - · ·
F	· · - ·	M	- -	T	-		
G	- - ·	N	- ·	U	· · -		

Using the International Morse Code table above, decode the letter from each signal plot and complete Agent 007's final words: **"The future in technology lies in _____."**

Explanation of Final Message: Hopefully, you agree with it. We sure do! It's precisely why we like to work with, and continue to support and champion, all the EE undergrads. So what are you waiting for? It's time to invent the next greatest "thing." But before you do, please don't forget to study hard for your upcoming final. And when you guys do end up becoming super successful and inventing the next greatest "thing," don't forget all the little people along the way. For instance, don't forget about the people who taught you how take a Fourier Transform and also how to flip and shift to successfully perform a convolution ☺

4) ***Lab Report Question #7***: In this lab, we decoded dash and dot signals by visual inspection of each graph. However, there are alternative ways to decode the message. A clever student in class realized you can decode a dash and dot signal using what we learned in Lab 4. Explain how you can use techniques from Lab 4, and in words, how you might implement that in Matlab.

ONE LAST MESSAGE

This quarter you have battled an evil EE235 student (twice!), decoded an insane amount of messages in Matlab, and learned how to perform convolution and convert signals from the time

domain to the frequency domain, to name a few. What an accomplishment! Unfortunately, all good things must come to an end. It was a pleasure working with you, Agent 008, but it is now time for a new group of up and coming EE students to follow in your footsteps and take on the EE235 challenge. Good luck and keep in touch!

FOR NEXT WEEK: LAB REPORT DUE

1) Use the lab report format on the class website
2) Turn in a PDF of your lab report (one per team) online via link on class website
3) Remember to include your M-files with your submission
4) Report is due prior to the start of your next lab section

BACKGROUND

1) **Matlab Review**

Concepts/Functions	Sample Code
Generating a signal in time domain	% x(t) = cos(πt) where 0 ≤ t ≤ 5 with Fs = 2 t = 0:(1/Fs):5; % Time samples vector x = cos(pi*t); % Actual signal vector % x(t) = cos(πt) where 0 ≤ t < 5 with Fs = 2 t = 0:(1/Fs):5 – (1/Fs); % Time samples vector x = cos(pi*t); % Actual signal vector

Computing magnitude of FFT	% Convert signal x and use N = 1024 freq samples
	N = 1024;
	x_fft = fftshift(fft(x, N));
	x_abs = abs(x_fft);
	
	% Compute frequency samples vector
	w_period = 2*pi*Fs/N;
	w = (-N/2:(N/2)-1)*w_period;

Using a 2 x 1 subplot and plotting on 1st figure	subplot(2, 1, 1); plot(……..);
Plotting a signal x vs. t	plot(t, x);
Changing axes limits	xlim([0 10]); ylim([-5 5]);

Labeling axis and adding plot title	xlabel('Time');
	ylabel('x(t)');
	title('Signal x(t)');

if-else Decision Statements	if x == -2
	  % Code
	elseif x > 2 && x < 5
	  % Code
	else
	  % Code
	end

for loops	for i = 1:5 % Code end
Creating string variables	str = 'Here is a string'; % Initialize string str = ['The value is', num2str(x)]; % Concatenation

2) **Function ifft: Computing Inverse Fourier Transform**

- The purpose of the Inverse Fourier Transform is to convert a signal X(jw) in the frequency domain back to signal x(t) in the time domain. We will accomplish this in Matlab by using the function **ifft**.

- Usage of **ifft**: Suppose we have a Fourier Transform **X_fft** that has **N** = 1024 samples in the frequency domain. Suppose we want to generate **N** = 1024 samples in the time domain. To convert **X_fft** back to the time domain, we would call **ifft** as such:

```
N = 1024;
x_ifft = ifft(fftshift(X_fft), N)
x = real(x_ifft);
```

- A call to function **fftshift** is needed to undo the shift that we performed to center the Fourier Transform **X_fft** around **w = 0**

- A call to function **real** is needed because numerical round-off errors in **fft** and **ifft** can introduce a very small nonzero imaginary component to the output **x**. In general, the time domain signals we analyze will be real-valued, so we need to remove the insignificant imaginary part.

3) **Matlab and Filters**

- LTI filters whose input and output satisfy an LCCDE will have the general frequency response

$$H(jw) = \frac{b_M(jw)^M + b_{M-1}(jw)^{M-1} + \cdots + b_1(jw) + b_0}{a_N(jw)^N + a_{N-1}(jw)^{N-1} + \cdots + a_1(jw) + a_0}$$

- A filter is defined in Matlab by the numerator and denominator coefficients in H(jw).

- The coefficients are defined as a row vectors and are listed out in the same order as the frequency response above, starting with the coefficients from the highest order of jw. Therefore, the filter coefficients for the general frequency above are defined as such:

```
b = [bM  bM-1  …  b1  b0]
a = [aM  aM-1  …  a1  a0]
```

- As an example, suppose $H(jw) = \frac{1+2jw}{1-2jw}$. Then, the filter coefficients are given by

```
b = [2   1];
a = [-2  1];
```

- To simulate filtering and obtain the output response in time of an LTI system to an arbitrary input, we can use the function **lsim** with one return value. The first two arguments are the filter coefficients of the numerator and the denominator, followed by the input and time samples:

```
y = lsim(b, a, x, t);
```

Note: the time samples for output **y** will be the same as the input **x**, so we can use the vector t for the input and output signals

4) **Rules for Multiplication and Division**

- So far, we have learned about the symbol * to perform a scalar multiplication, a type of multiplication in which a number is multiplied to every element in a matrix:

```
3 * [1   2]  =  [3 * 1   3 * 2]  =  [3   6]
```

- In Matlab, there are two other types of multiplication operations:
  - array multiplication with operator .*
  - matrix multiplication with opreator *

- In array multiplication, matrices of equal dimensions are multiplied together, where each element in the first matrix is multiplied with a similarly located element in the second matrix. Array multiplication is sometimes called element-by-element multiplication:
  - Valid Example with Matrices: $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} .* \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix} = \begin{bmatrix}1*2 & 2*3\\3*4 & 4*5\end{bmatrix} = \begin{bmatrix}2 & 6\\12 & 20\end{bmatrix}$
  - Valid Example with Vectors: $\begin{bmatrix}1\\3\end{bmatrix} .* \begin{bmatrix}2\\4\end{bmatrix} = \begin{bmatrix}1*2\\3*4\end{bmatrix} = \begin{bmatrix}2\\12\end{bmatrix}$
  - Valid Example with Scalars: 1 .* 2 = 2

- If you have taken linear or matrix algebra before, then you are already familiar with the notion of matrix multiplication. In matrix multiplication, the two matrices must have a common inner dimension. This means the number of columns in the first matrix must equal the number of rows in the second matrix:
  - Valid Example : $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} * \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix} = \begin{bmatrix}1*2+2*4 & 1*3+2*5\\3*2+4*4 & 3*3+4*5\end{bmatrix} = \begin{bmatrix}10 & 13\\22 & 29\end{bmatrix}$
  - Valid Example: $\begin{bmatrix}1 & 2 & 3\end{bmatrix} * \begin{bmatrix}1\\2\\1\end{bmatrix} = [1*1+2*2+3*1] = [8]$
  - Invalid Example: $\begin{bmatrix}1\\3\end{bmatrix} * \begin{bmatrix}2 & 3\\4 & 5\end{bmatrix}$

- Note: scalar multiplication can be performed using the operator .*, in addition to the operator * since scalar multiplication is essentially an element-by-element multiplication with the same number. To be consistent, we will use the operator * for scalar multiplication as we have been doing, but be aware of this special exception to the rule.

- Similarly, with division, there are the following division operators:
  - scalar division with operator ./
  - array division with operator ./
  - matrix division with operator /

- Note: unlike scalar multiplication, scalar division with / only works when both operands are scalar values. It will not work when either one is a matrix.

- Array division is an element-by-element division operator, and hence can only be performed with matrices of equal dimensions

- Matrix division is a strange operator, one which we will not use. You just need to be aware that matrix division only works if both matrices have the same number of columns

5) **Labeling Figure Windows**
- So far, we know that the command **figure** will create a new figure window

- In some cases, we will want to go back to a previous figure window after we have already created several other figure windows. We can accomplish this by using the function **figure** with 1 argument. The argument allows you to label each figure window with a number and access a figure window later on.

- Example Code:

```
figure(1);                      % Create figure #1
subplot(2,1,1); plot(t, x);
…
figure(2);                      % Create figure #2
plot(t, y);
…
figure(1);                      % Go back and access figure #1
subplot(2,1,2); plot(t, z);
```