University of Washington
Department of Electrical Engineering

EE 235 Lab 5:

Time Domain to Frequency Domain

In this lab, we will learn how to transform signals in Matlab from the time domain to the frequency domain. In addition, we use will Matlab to identify the frequencies of $e^{j\omega t}$ that make up a signal. Once you have a good handle of identifying a signal's frequency content, we will then apply it to a fun problem of decoding a phone number based on its touch-tone signals.

Important Concepts From Lecture You Will Use In Lab 5

- Finding the Fourier Series coefficients a_k of x(t)
- Identifying the frequency content x(t), and understanding the relationship between Fourier Series coefficient index k and frequency ω

What Is Expected From You In Lab 5

- Completion of 3 pre-lab exercises (5 points)
- Completion of 2 in-lab check offs with TA (5 points)
- Completion of a lab report (10 points)

EXERICSE #1: Identifying Frequency Content of a Signal

In this exercise, we will learn how to use Matlab to identify the frequency content of a signal **x(t)**. In particular, we will focus on the case when **x(t)** is periodic and a sum of sinusoidals. The signal we will analyze represents a touch-tone telephone signal, an application we will study more in Exercise #2.

Remember from class that periodic signals can be written as a sum of complex exponentials $e^{j\omega t}$ at different frequencies ω. When we say a signal **x(t)** has a certain list of frequencies ω, what we are really doing is identifying the frequencies of $e^{j\omega t}$ that make up, and can be summed up, to produce **x(t)**. Identifying a signal's frequency content is important because working in the frequency domain is ideal for filter design and in the analysis of LTI systems.

1) ***Pre-Lab Questions***: Consider the following sum of two sinusoids:

$$\boldsymbol{d0(t) = \sin(2\pi(941)t) + \sin(2\pi(1336)t)}$$

a) Transform this signal into its Fourier Series representation by finding the Fourier Series coefficients $\mathbf{a_k}$, given that $\mathbf{w_o} = 2\pi$. Note: you do not need to prove $\mathbf{w_o} = 2\pi$.

b) Each Fourier Series coefficient $\mathbf{c_k}$, for which $\mathbf{c_k}$ is nonzero, corresponds to a frequency **ω** of $e^{j\omega t}$. Using your results from (i), how many frequencies **ω** do you expect signal **d0(t)** to have?

c) Each Fourier series coefficient index **k** (with nonzero $\mathbf{a_k}$) corresponds to a frequency $\boldsymbol{\omega = kw_o}$. Using this fact, identify the frequencies **ω** of signal **d0(t)**. Round these frequencies **ω** to the nearest tenth.

d) Note that the frequencies in (c) are angular frequencies (rad/sec). Convert these angular frequencies **ω** to frequencies **f** in units of Hz. These should match up with the values that are in parentheses in **d0(t)**.

e) **Read Background Section (5) and (6)** and answer the following questions**:**

$$A = [0 \quad 1 \quad 5 \quad 10 \quad 5 \quad 1 \quad 0 \quad 2 \quad 7 \quad 10 \quad 7 \quad 2 \quad 0]$$

i) What indices in the matrix **A** will the line of code **find(A > 8)** return?

ii) Suppose **index** = [3 4 5 9 10 11]. What will the output be for the line of code **A(index)**?

iii) What should you pass to the **find** function if you wanted to extract all the indices where the values 7 and 10 in matrix **A**? Note: there are several answers to this.

iv) Fill in the rest of this code to extract the 0's from matrix A:

```
index = FILL IN CODE;
A(index)
```

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	• Creating time samples 0 ≤ t < t_end
• Playing sound signal	• Using **fft** and **fftshift** functions
• Using the **pause** function	• Using **abs** function
• Plotting with subplots	• Using **find** function
• Adding title and labels to plot	• Extracting list of elements in a vector

2) ***Lab Exercise***: Let us now use Matlab to verify our pre-lab results.

a) Create a new working directory called **Lab 5**. In this directory, open up a script and call it **Ex1.m**. Make sure to clear all variables and close all figures.

b) The first thing we will do is generate the tone signal from pre-lab and analyze the signal in the time domain. The point of this portion of the exercise is to show that viewing signals in the time domain is sometimes not useful:

i) First, create the time samples vector **t** in the range 0 ≤ t < 0.25 using sampling rate Fs = 8000. Note that t is only less than 0.25 and not less than or equal to 0.25. **Refer Background Section (2), if needed.**

ii) Using time samples vector **t**, generate your signal vector **d0** using the equation from pre-lab: $\boldsymbol{d0(t) = \sin(2\pi(941)t) + \sin(2\pi(1336)t)}$

iii) Using the **sound** and the **pause** functions, play sound signal **d0** twice with a 1-second pause in between.

iv) Run your script to hear the tone signals. The sound should resemble what you hear when dialing a number 0 twice on a touch-tone telephone.

v) Using a 2 x 1 subplot, plot **d0 vs. t** as the 1st subplot. No need to adjust your axes, but label and title your plot.

c) Now let us convert our signal from the time domain to the frequency domain, and show how useful it is to view signals in a different domain:

i) Using the **fft** function, create **N** = 4096 samples of the signal in the frequency domain and store it in a variable called **D0_fft**. Make sure to also shift the output of the **fft** using function **fftshift** so that the results are centered around w = 0. **Refer to Background Sections (3) and (4) on these functions, if needed**.

ii) Because the output of the **fft** is complex, we will want to take the magnitude of the fft. As we will see, the magnitude of the fft contains the most useful information about the signal's frequency content. Using function **abs**, compute the magnitude of **D0_fft** and store the return value in variable **D0_abs**. **Refer to Background Section (6) on this function, if needed**.

iii) Define the frequency samples **w** using the sampling rate **Fs** and the number of frequency samples **N**. **Refer to Background Section (4), if needed**.

iv) On the same 2 x 1 subplot used in (b), plot **D0_abs vs. w** as the 2nd subplot. Adjust your axes so that the x-axis is between -9000 to 9000 and the y-axis is between 0 and 1000. Label and title your plots.

v) Run your script to view your plots in the frequency domain. Verify your pre-lab answer and that you have the correct number of frequencies **ω** for **d0(t)** and that the spikes roughly fall at the values of **ω** you calculated.

b) Now let us identify the frequency content of **d0(t)** by determining exactly where in frequency (in Hz) these nonzero spikes occur:

i) Using the **find** function, identify the indices in **D0_abs** where **D0_abs** is greater than a threshold value of 900. Store your results in **index_tone0**. **Refer to Background Section (6) on this function, if needed**.

ii) We can now identify the frequency content of our signal. Using **index_tone0**, extract the frequencies in **w** that represent the frequency content of **d0(t)**. Convert and report these frequencies in Hz by dividing the extracted frequencies in **w** by 2π. Make sure to wrap your code for 2π in parentheses. Store your result in variable **freq_tone0** and display the output on the COMMAND window. **Refer Background Section (5) on vector element extraction, if needed**.

iii) Run your script and view the frequency content of **d0(t)** on the COMMAND window. Verify that the values of the frequencies (in Hz) match up with your answers from pre-lab.

2) ***Lab Check-Off #1 of 2***: Show **Ex1.m** to a lab TA and demonstrate you know how to plot a signal in the frequency domain and that you can identify the frequency content of a signal in Matlab.

3) ***Lab Report Question #1 of 6***: To find the frequency content, we used the **find** function with a threshold of 900. Using your FFT magnitude plot, explain why a threshold of 100 would not work. Would that threshold identify more or fewer frequencies than the threshold of 900? Explain your answer.

4) ***Lab Report Question #2 of 6***: Suppose we altered the signal for **d0(t)** and used $\boldsymbol{d0(t) = 1 + \sin(2\pi(941)t) + \sin(2\pi(1336)t)}$ instead. A student incorrectly reports that the graph for the FFT of D0 would not change. Explain why this student is incorrect.

EXERCISE #2: Classifying Touch-Tone Telephone Signals

In this exercise, we will learn more about the signals of a touch-tone telephone system by expanding on one of the signals we started analyzing in Exercise #1. We will also write a function to decode and identify the number corresponding to an arbitrary touch-tone signal from its frequency content.

The touch-tone system on a telephone uses signals of different frequencies to indicate which key has been pushed. In particular, it uses a type of signaling scheme called dual-tone multi-frequency (DTMF) signaling. In DTMF signaling, each touch-tone on the keypad is represented as a sum of two sinusoidal tones, one tone at a low frequency and another tone at a high frequency. The DTMF keypad frequencies (in Hz) are shown in the table below:

	1209 Hz	1336 Hz	1477 Hz
697 Hz	**1**	**2**	**3**
770 Hz	**4**	**5**	**6**
852 Hz	**7**	**8**	**9**
941 Hz		**0**	

To generate key 2, we need the signal $\boldsymbol{d2(t) = \sin(2\pi(697)t) + \sin(2\pi(1336)t).}$

1) ***Pre-Lab Questions***:
   a) Write the equations for the following DTMF key signals: **d4(t), d5(t), d8(t)** and **d9(t)**.

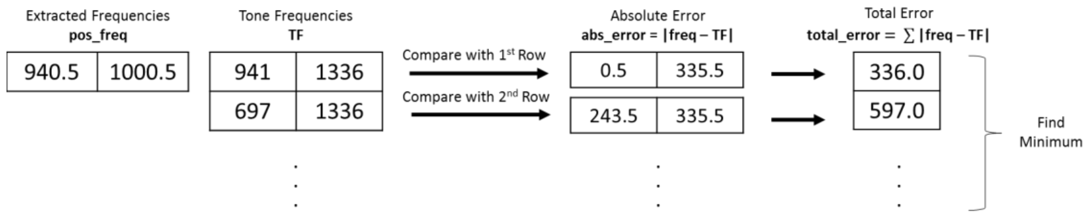
   b) Write the header for a function called **classify**, which takes in an input tone signal **x** sampled at sampling rate **Fs** (also provided as an input) and then outputs the classified phone number to the variable **num**.

   c) To classify an arbitrary tone signal, we will compare its frequency content to a table of tone frequencies. Each row contains the two frequencies for one touch tone number 0 - 9. Therefore, row 1 (which will represent tone number 0) will contain the values 941 and 1336. Row 2 (tone number 1) will then have the values 697 and 1209. Write the Matlab code to create a 10 x 2 matrix that contains all the tone frequencies, and call the matrix **TF**. **<u>Refer to Matlab Review Section on how to create a matrix, if needed</u>**.

   d) What is the line of code to create a <u>column vector</u> of zeros called **total_error** that has exactly 10 elements?

   e) Using **length(total_eror)**, what is the for-lop statement to loop through all the rows in column vector **total_error** starting at **i = 1**?

   f) The frequency content we extract from tone signal **x** will not be integers. As a result, we cannot do a simple equality check through each line of matrix **TF** to identify the correct tone number. Instead, we need to calculate and identify the minimum absolute error measurement between the frequencies we extract and the frequencies in each row of matrix **TF**, as illustrated below:



Read Background Section (6) on the function **abs** and **min**, and then answer the following:

i) Using the function **abs**, write the line of code to calculate the absolute error measurement **abs_error** between the extracted frequencies **pos_freq** and **ith** row of the matrix **TF**. Note that **abs_error** is a 1 x 2 row vector.

ii) What is the line of code to sum up the 1st and 2nd elements of **abs_error**, the value of which will be stored as the **ith** element of vector **total_error**?

iii) Using the function **min** with two outputs, write the line of code to find the minimum value **min_e** and corresponding location index **index_min_e** in vector **total_error**?

iv) Note that if the correct tone number is 0, **index_min_e** will equal 1. If the correct number is 1, **index_min_e** will equal 2. If the correct number is 9, **index_min_e** will equal 10. How do we calculate the tone number **num** from **index_min_e**?

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

Matlab Concepts/Functions to Review	New Matlab Concepts/Functions
• Creating variables • Using for loops • Writing functions • Vector concatenation • Generating signal in time domain • Using the **zeros** function • Creating a vector of **zeros** that last ***t*** seconds	• Creating time samples 0 ≤ t < t_end • Using **fft** and **fftshift** functions • Using **abs** function • Using **find** function • Extracting list of elements in a vector • Using **min** function

2) ***Lab Exercise***: We will now write and test the function to classify a tone signal.

a) Open up a new file and call it **classify.m**. Using your pre-lab, write the function header for **classify**.

b) Using pre-lab, define the tone frequency matrix **TF**.

c) Using the same procedure from Exercise #1, let us now extract the frequency content of tone signal **x**:

i) First, we need to convert the signal from the time domain to the frequency domain. Using function **fft**, **fftshift**, and **abs**, compute the magnitude of the **fft** of **x** consisting of **N** = 4096 samples, and store the result in a variable called **X_abs**.

ii) Define the frequency samples **w** using the sampling rate **Fs** and the number of frequency samples **N**.

iii) Now, let us proceed to identify the frequency content of signal **x**. Using the **find** function, identify the indices in **X_abs** where **X_abs** is greater than a threshold value of 900. Store your results in **index_tone**.

iv) Using **index_tone**, extract the frequencies in **w** that represent the frequency content of **x**. Convert and these frequencies to Hz by dividing the extracted frequencies in **w** by 2π. Make sure to wrap your code for 2π in parentheses. Store your result in variable **freq_tone**.

v) Recall from Exercise #1 that **freq_tone** contains both the negative and positive frequencies of a signal for a total of 4 frequencies. We only need the positive frequencies, which are located in the last two elements of **freq_tone**. Grab the positive frequencies by extracting the 3rd and 4th elements of **freq_tone**, and store these frequencies in variable **pos_freq**.

d) Now that we the frequency content of signal **x**, let us classify it to the correct tone number using some of the lines of code you wrote in pre-lab:

i) Create a <u>column vector</u> of zeros with 10 elements called **total_error**. You will use this to store the total absolute error measurement for each tone number.

ii) We will now calculate **total_error** one row at a time using a for loop:

```
% LOOP through all the elements in total_error:

    % Calculate the absolute error measurement abs_error between pos_freq
    % and the ith row in matrix TF.

    % Sum up the first and second elements of abs_error to compute the total
    % error, and store the value in the ith element of total_error.

% END OF LOOP
```

iii) Using the function **min**, find the minimum error **min_e** and the corresponding location index **index_min_e** in vector **total_error**.

iv) From **index_min_e**, calculate the tone number and store the value in the output variable **num**.

e) Provide function header and in-line comments.

f) Let us now test your function on a separate script. Open up a new script and call it <u>**Ex2.m**</u>. Clear all variables and close all figures.

g) Create a vector of time samples called **t** in the range $0 \le t < 0.25$ using sampling rate **Fs** = 8000. Note that t is <u>only less than</u> 0.25 and not less than or equal to 0.25.

h) Using pre-lab, write the equations for the tone signals **d4**, **d5**, **d8**, and **d9**.

i) Now let us test your function **classify** on each of these tone signals. To start, call your function **classify** with the tone signal **d4**, and store the output in variable **num**. Display the output on the COMMAND window. Run your script and verify your function is correct and that you get the value 4 for **num** on the COMMAND window.

j) Repeat (i) with **d5**, followed by **d8**, and then **d9**.

k) As an extra test, let us test the function and show it still works even if we modify the signal **d4** with extra zeros padded to the end of the signal:

   i) Create a row vector of zeros called **z** that lasts a duration of 0.25 seconds using sampling rate **Fs** = 8000. <u>**Refer to Matlab Background Section, if needed**</u>.

   ii) Append the row vector **z** by concatenating it to the end of row vector **d4**, and call the result **d4_z**. <u>**Refer to Background Section (7), if needed**</u>.

   iii) Call function **classify** with signal **d4_z** and store the output in the variable **num**.

   iv) Run your script and verify you still get the correct tone number.

l) Comment your script.

3) <u>***Lab Check-Off #2 of 2***</u>: Show your script to a lab TA and demonstrate you can classify a tone signal to the correct tone number.

4) <u>***Lab Report Question #3 of 6***</u>: In (2k), you concatenated your tone and zeros signals (both row vectors) using either a space or a comma as your delimiter. If you had used a <u>semicolon</u> as your delimiter instead, describe how signal **d4_z** would change.

5) <u>***Lab Report Question #4 of 6***</u>: Suppose you created a new signal called **d(t)**, where

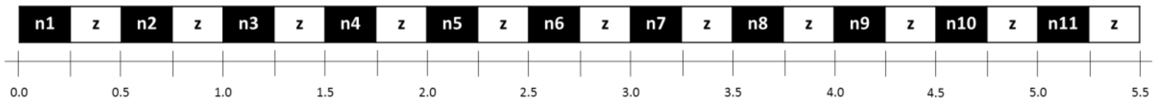
$$\mathbf{d(t) = d4(t) + d8(t)}$$

If you called **classify** with this signal, the values for **pos_freq** end up being negative. Explain why this happens. What are these negative frequencies?

EXERCISE #3: Decoding Phone Number from Touch-Tone Signals
In this experiment, we will put together what we learned in the last two exercises to decode an entire phone number from its touch-tone signals.

Here is the scenario for this week: Your TA has a secret to tell you. It is the name of the TA's favorite sit-down restaurant in San Diego. Yelp users agree – it has over 10,000 reviews! Your TA would love to tell you the name of this restaurant, but your TA feels it would be more fun for you to figure it out yourself by decoding the restaurant's telephone number from a sound file. Your mission, EE235 students, if you choose to accept, is to show your TA that you've got mad Matlab skills and can unlock the name of the best sit-down restaurant in San Diego ever!

1) ***Pre-Lab Questions***:
   a) The phone number signal will be in the following format consisting of 11 tone signals **n1 to n11** of duration 0.25 seconds and a silence signal **z** (of zeros) also of duration 0.25 seconds (but not including the last sample) following each tone:



The numbers can be decoded by extracting each tone signal (along with its follow-on silence signal) and classifying its tone number from its frequency content. We can again use a loop to perform this task. Assuming you know the sampling rate **Fs** and loop index variable **i**, what is the mathematic equation for **start_index** and **end_index**, which you will use to extract the **ith** tone and silence signals?

To help you determine these equations, you may fill in the table below to list out the formulas to determine the start and end indices for each tone and silence signal. Then, from all these formulas, determine a general equation in terms of **i** and **Fs**. **Refer to Background Section (5), if needed.**

Section	**start_index**	**end_index**
n1 + z: $0.0 \le t < 0.5$	0.0 * Fs + 1	0.5 * Fs
n2 + z: $0.5 \le t < 1.0$		
n3 + z: $1.0 \le t < 1.5$		
n4 + z: $1.5 \le t < 2.0$		
n5 + z: $2.0 \le t < 2.5$		
n6 + z: $2.5 \le t < 3.0$		
n7 + z: $3.0 \le t < 3.5$		
n8 + z: $3.5 \le t < 4.0$		
n9 + z: $4.0 \le t < 4.5$		
n10 + z: $4.5 \le t < 5.0$		
n11 + z: $5.0 \le t < 5.5$		

Section	**start_index**	**end_index**
ith tone and silence signal		

   b) **Read Background Section (8) on strings** and answer the following question: Suppose you have a Matlab variable **x** with the value 1. Using vector concatenation, the variable

x, and the function **num2str**, what is the line of code to create the string “The value of x is 1”, which will be stored in the variable **str**?

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

Matlab Concepts/Functions to Review	**New Matlab Concepts/Functions**
• Using the **zeros** function • Creating time samples vector • Plotting basics and subplots • Using for loops • Vector extraction	• Vector concatenation • Creating string variables • Concatenating string variables

2) ***Lab Exercise:*** Let us now decode an entire phone number and plot all the extracted signals on one subplot.

a) Open a new script and call it **Ex3.m**. Again, clear all variables and close all figures.

b) From the website, download the data file **phonenum.mat** and then load it. The file contains two variables:
- **Fs**: sampling rate of phone number signal
- **x**: actual phone number signal you need to decode

c) To see the signal you will be analyzing, plot the signal **x**. Do not forget to compute the corresponding time samples of **x** first, so that you can plot **x vs. t_x**. Leave axes unchanged. Title and label your plot.

d) To prep us for the upcoming for loop that will decode the phone number, we need to do two things:

i) Create a vector to store each individual digit in the phone number. Declare a row vector of zeros called **phone_num** with 11 elements.

ii) Load a new figure window, which we will use to plot all extracted tone signals and their follow-on silence signal.

e) Using a for-loop, we will now fill in **phone_num** one element at a time. We will extract each tone signal, decode it, and plot it on one figure window using subplots. The final plot should have the following format, where each subplot has both the tone signal and the follow-on silence signal:

Signal #1	Signal #2	Signal #3	Signal #4
Signal #5	Signal #6	Signal #7	Signal #8
Signal #9	Signal #10	Signal #11	

The pseudocode for the for-loop is given below:

```
% LOOP through all elements in phone_num using loop variable i

    % Using pre-lab, compute start_index and end_index for the ith tone and silence signals

    % Using start_index and end_index, extract the ith tone and silence signals from phone
    % number signal x, the values of which are stored in the variable signal.

    % Call the classify function to decode the tone number from signal, and store the output
    % as the ith element in the vector phone_num.

    % Create time samples vector t_signal for the extracted signal.

    %  Plot signal vs. t_signal as the ith subplot.  Leave the axes unchanged.
    % To save space, the axes will not be labeled.  However, it will have a title.

    % Using the loop variable i and the function num2str, create the title string str,
    % which will display the decoded ith number.  As an example,
    % if the ith number is 7, then the title should say: Number = 7

    % Using str, add the title to the current subplot.

% END OF LOOP
```

f) Display the contents of **phone_num** on the COMMAND window.

g) Run your script to view the decoded phone number. Verify that the digits in the decoded phone number are displayed on the subplot titles.

3) ***Lab Report Question #5 of 6***: What is the name of this well-known fast-food restaurant?

4) ***Lab Report Question #6 of 6***: A student accidentally calls the **title** function outside the loop, following the **end** statement. Explain how the final plot changes and why these changes happen.

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 in one week prior to the start of your next lab section

BACKGROUND SECTION

1) **Matlab Review**

Concepts/Functions	**Sample Code**

Creating Matlab variables	x = 1; % Scalar
	y1 = [2; 4 ; 6]; % Column vector
	y2 = [1 3 5]; % Row vector
	z = [3 6 9; 1 1 1]; % 2 x 3 Matrix

Extract elements in a vector/matrix	a = y1(1); % Extract one element
	b = y1(1:2); % Extract multiple elements
	b1 = y1(3:end); % Extract until end of vector
	b2 = y1(:); % Extract all elements
	
	c = z(1,2); % Extract one element
	d = z(2, 1:2); % Extract multiple elements
	e = z(2, :); % Extract all elements in 2nd row

Changing elements or storing values in a vector/matrix	y1(1) = 3; % Change one element y1(1:2) = -1; % Change multiple elements z(1,2) = 0; % Change one element z(2, 1:2) = 2; % Change multiple elements
Dimensions of a vector or matrix	length(y1) % Num elements in vector size(y1) % Dimensions (rows, columns) size(z, 1) % Num of rows size(z, 2) % Num of columns

Generating a signal in time domain	% Generate time samples
	t = 0:1/Fs:5 % $0 \le t \le 5$ with Fs = 2
	
	% Generate actual signal
	x = cos(pi*t); % x(t) = cos(πt)

Creating a vector that lasts ***t*** seconds	x = zeros(1, 3*Fs + 1); % zeros vector that lasts 3 sec

Relationship between index and time:
$i = t \times F_S + 1$
	% Extracting t = 5 and storing in y
	index = 5 * Fs + 1;
	y = x(index);
	
	% Accessing t = 5 and changing value to 1
	index = 5 * Fs + 1;
	x(index) = 1;
	
	% Extracting $0 \le t \le 5$ and storing in y
	start_index = 0 * Fs + 1;
	end_index = 5 * Fs + 1;
	y = x(start_index:end_index);

Opening a new figure window	figure;

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)');

Loading and play sound file	load file.mat; % Contains variables y and Fs sound(y, Fs);
Display to COMMAND window	x = x + 1 % Display output of calculation A % Display contents of matrix A

for loops	for i = 1:5
	  % Code
	end

Function Header Examples	function y = myexample(x) function [y1, y2] = myexample(x) function [y1, y2] = myexample(x1, x2)
Example Function with Header	% ADDME Add two values together. % USAGE: C = ADDME(A,B) adds A and B together % AUTHOR: [FILL IN NAME HERE] function c = addme(a, b) % Add a and b together and store in c c = a + b end

2) **Creating Time Samples Vector**

- Review So Far:
  - Creating time samples given signal x: t_x = (0:length(x)-1) * (1/Fs);
  - Creating time samples over time range $0 \le t \le 1$: t = 0:(1/Fs):1;

- New Scenario: creating time samples over time range $0 \le t < 1$
  - To do this, we need to remove the last time sample at t = 1
  - We can remove sample t = 1 by going back one period 1/Fs

   - Matlab code: t = 0:(1/Fs):1-(1/Fs)

3) **Function fft: Converting Signal from Time Domain to Frequency Domain**
   - We will use a function called **fft**, where **fft** stands for Fast Fourier Transform. We will learn about Fourier Transforms in Weeks 6 and 7 in class, and you will learn about the FFT in more detail in EE 341. For now, all you need to know is that the FFT is an algorithm that converts a signal from the time domain **t** to the frequency domain **w**.

   - Similar to how time domain signals can only be defined at discrete samples, the **fft** can only compute samples of the signal in the frequency domain as well.

   - <u>Usage of **fft**</u>: Suppose we have a signal **x(t)** that is sampled at some sampling rate **Fs**. Also, suppose we want to generate **N** = 1024 samples in the frequency domain. To convert **x** to the frequency domain, we would call **fft** as such:

     ```
     N = 1024;
     X_fft = fft(x, N);
     ```

   - The period between the frequency samples is given by: $\frac{2\pi F_S}{N}$

   - The **fft** computes samples over the range: $0 \leq w < 2\pi F_s$. Note that the upper bound is only less than and not less than or equal to, which means the last frequency sample actually occurs at $2\pi F_s - \frac{2\pi F_S}{N}$

4) **Function fftshift: Centering Frequency Domain at w = 0**
   - The **fft** actually starts computing samples in the frequency domain starting at w = 0. To center our frequency samples around w = 0, we need to use a function called **fftshift**.

   - We can convert x to the frequency domain and make sure the signal is centered at w = 0 with:

     ```
     N = 1024;
     X_fft = fftshift(fft(x, N));
     ```

   - The period between the frequency samples is still given by: $\frac{2\pi F_S}{N}$

   - After the **fftshift**, the range of frequency samples is now: $-\pi F_S \leq w < \pi F_s$

   - We can define the frequency samples vector, given the sampling rate **Fs** and the number of frequency samples **N**:

     ```
     w_period = 2*pi*Fs/N;
     w = (-N/2:(N/2)-1)*w_period;
     ```

5) **Summary of Element Extraction**
   - Review So far:
     - To extract the first three elements: y = x(1:3);
     - To extract all the elements starting from the 3rd element: y = x(3:end);
     - To extract all columns from 1st row of matrix: y = A(1, :)

   - New Scenario: Suppose we want to extract the third, fourth, and sixth elements a vector

- Instead of using the colon operator, you would list out the elements in a vector:

  ```
  y = x([3 4 6]);
  ```
- Alternatively, you could define the list of elements in a vector and then extract:

  ```
  index_x = [3 4 6];
  y = x(index_x);
  ```

6) **Other New Functions in Matlab**

- Function **abs**: Used to compute the magnitude of a complex number
  - Usage: Suppose we want to compute the magnitude of a variable X

    ```
    X_abs = abs(X)
    ```

- Function **find**: In the last lab, we used this function to find the maximum of a vector. In this lab, we will use the function find to locate indices of a vector where the values of a vector cross a certain threshold. Consider the following row vector X:

  $$X = [1 \quad 3 \quad 4 \quad 2 \quad 1 \quad 3]$$
  - Suppose we want indices in X where X > 2
  - Then, we would use the **find function** as such:

    ```
    index_X = find(X > 2)
    ```
  - In this example, index_X = [2 3 6], which are the indices in X where X > 2

- Function **min**: Used to find the minimum value and corresponding index location in a vector
  - Usage: Find minimum and location of minimum in vector x

    ```
    [min_x, index_min_x] = min(x);
    ```

7) **Review: Appending to a Vector and Vector Concatenation**

- We can **append** column vector **y2** to column vector **y1** and create a new vector called **y3** by using the semicolon delimiter as such:

  ```
  y3 = [y1; y2];
  ```

- If **y1** and **y2** are row vectors instead, we can concatenate **y1** and **y2** in a similar fashion but without the semicolon:

  ```
  y3 = [y1 y2];
  ```

8) **String Variables**

- Like other programming languages, a string in Matlab is a vector of characters
- In Matlab, string values are enclosed in single quotes
- To create a string, we use the same format **<name> = <initial_value>**:

  ```
  s = 'Any characters';
  ```
- We can concatenate two strings together using square brackets:

  ```
  s = ['Any ', 'characters'];
  ```
- Alternatively:

  ```
  str1 = 'Any ';
  str2 = 'characters';
  str = [str1, str2];
  ```
- To convert a number to a string, use the function **num2str** as such:

  ```
  x = 1;
  str = ['x is ', num2str(x)];    % Value of str is "x is 1"
  ```