EE 301 Lab 1 – Introduction to MATLAB®

1 Introduction

In this lab you will be introduced to MATLAB® and its features and functions that are pertinent to EE 301. This lab is written with the assumption that students have some familiarity with programming languages and constructs but have not used MATLAB® before this course.

2 What you will learn

This lab will focus on using MATLAB® to create signals (vectors and matrices in MATLAB® speak). With this ability you will be prepared for the future lab exercises.

3 Background Information and Notes

MATLAB® is the leading standard software package for modeling signals and systems. The reason for this is the ease with which it allows one to represent and define signals. Data is represented as arrays of varying dimensions. A **scalar** is a single number (which can be thought of as a zero dimensional array). A **vector** is a 1-D array (1xN or Nx1 sequence of numbers). A **matrix** is a 2-D array (an MxN array of numbers where M=# of rows and N=# of columns). Higher dimension arrays can also be used (3-D arrays are common when representing images and other data). [Note: Unlike some other computer languages, variables in MATLAB® do not have to be declared ahead of time and can be created at any point in time. Similarly vector and matrix variables can be resized on the fly (i.e. if initially 'x' was a 1x4 row vector, assigning a new value to the 5^{th} column will make x a 1x5 vector).] The other advantage of MATLAB® is the wide array of built-in functions and processing techniques that can work on these data types whether they are scalars, vectors, or even matrices. As you work with these functions you can always get more information and help by typing "help *function_name*" at the prompt (e.g. >> help plot).

4 Guided Exercises

Let us focus on different data representation techniques in MATLAB® by providing some examples. Though some of these examples are obvious, you are encouraged to actually type these in at the command prompt to see the effect.

Important: In MATLAB®, the default is to print the result of each statement or line to the screen. However, this is often undesirable, especially for large vectors and matrices. To suppress printing of the result, end each statement with a semicolon (;). Example: Entering: x = 1+1 at the prompt will print '2' to the screen. Entering: x = 1+1; will suppress the result being printed, though x will still be assigned the value of 2.

a **Scalars**: Scalar variables are single values (i.e. dimensions of 1x1) The following are examples of scalar variable creation/assignment.

```
>> x = 5
>> y = 8.2e5          % notice the use of scientific notation…same as y=820000
>> z = -1.26e-3       % same as z = -0.00126
```

Q1. Scalars: In your command window initialize a variable z by assigning to it the value 35000. While doing so, use scientific notation and suppress printing of this variable. Now create another variable called y by initializing it to 4. Display the variable y. Now assign the value stored in z to y. Display y again.

b **Vectors**: Vectors are one dimensional arrays. A 1xn vector is known as a row vector since all n elements are located in a single row. A nx1 vector is a column vector. Row and column vectors are simply the transpose of each other and can be converted by using the transpose operator (‘). Elements of a vector can be specified explicitly or specified using the start:end or start:step:end notation. The start:end notation will automatically use a stepsize of 1 as shown in the following examples. Finally, accessing individual elements of a vector can be done using parentheses as an index operator: vector_name(index_location) where indices of a vector start at 0.

```
>> a = [0 1 2 3 4 5]  % Creates a 1x6 row vector with elements 0 through 5
>> b = [0:5]          % Creates the same 1x6 row vector in a
>> c = [0:1:5]        % Creates the same 1x6 row vector in a and b
>> d = [0; 1; 2; 3; 4; 5]    % Creates a 6x1 column vector w/ elements 0 - 5
>> e = [0:5]'         % Transposes a row vector to create a column vector
>> f = [0:2:9]        % Creates a 1x5 row vector w/ elements 0, 2, 4, 6, 8
>> g = [5:-1:0]       % Creates a 1x6 row vector w/ elements 5, 4, 3, 2, 1, 0
>> a(6)               % Returns the sixth element in the vector a (i.e. 5)
>> a(1:2:5)           % Selects the 1st, 3rd, and 5th elements from the vector a
                      % and places them in a new 1x3 vector (i.e. [0 2 4])
```

Q2a. Vectors: Create a vector ‘i’ by initializing it to [9 8 7 6 5 4 3]. Create a vector ‘j’ by using the command j=[9:-1:3]. What is the result of the operation i-j? Store this result in a vector called m. What is the 5th element of this vector?

Q2b. Vectors: There is a command in MATLAB® to create a vector of zeros of a desired size (e.g. z=zeros(1,3) creates the vector [0 0 0]). Is there a similar command which can create a vector of ones (for e.g. [1 1 1])?

c **Matrices**: Matrices are two dimensional arrays. If the dimensions are N x N then it is called a square matrix. Matrices can be composed explicitly or built up from vectors. Selection from a matrix is also accomplished using parentheses as the selection operator. X(2,1) returns the element in the second row, first column. Entire rows or columns of a matrix can be returned using the colon (:) operator (see below).

```
>> h = [1 2 3; 4 5 6]          % Creates a 2x3 matrix
>> i = [1:3; 4:6 ]             % Creates the same 2x3 matrix
>> j = i(1,:)                  % Creates a row vector using the first row of the
                               %  matrix i as its elements
>> k = [i(1,:) 11 12 13; 14 15 16 i(2,:)]   % Creates a 2x6 matrix
```

Q3a. Matrices: Create a matrix M=[1 4 5; 4 6 9; 5 9 3]. Display the element which belongs to the third row and first column. Display the second column.

Q3b. Matrices: Create a column vector u=[0.2996;0.5361;-0.7892]. Multiply the matrix M of the previous exercise with the vector u and store the result in r. Now multiply each element of the vector u by -5.0125 and store the result in s. Compare r and s, do you get the same result? Try >>help eig to learn more about vectors like 'u' which are called eigenvectors of a matrix.

Q3c. Matrices: What is the command in MATLAB® which can create an identity matrix (a matrix with 1s on the diagonal and 0s elsewhere) of a desired size? Use this command to create a 4x4 identity matrix I.

d **Data creation functions**: MATLAB® has some built-in functions that will create certain kinds of scalars, vectors, and matrices for you. Each function can return a vector or matrix of the desired size by passing it the desired M x N dimensions. Other functions are helpful when determining the size of an array.

```
>> m = rand            % Returns a random scalar value between 0 and 1
>> n = rand(4,2)       % Returns a 4x2 matrix of random values between
                       %   0 and 1
>> o = zeros(1,10)     % Creates a row vector of 10 zeros
>> r = length(o)       % Returns the length of a vector (i.e. r = 10)
>> [s,t] = size(p)     % Returns the dimensions of an array (i.e. s = 10, t = 2)
```

e **Arithmetic functions, Vectorization, & Figures**: MATLAB® has many built-in arithmetic functions. Some are simply operators: +, -, *, /, ^ (exponentiation) while others are functions that must be passed a value as input and return the result: sqrt(), sin(), cos(), tan(), atan(), exp(), log(), log10(), abs(). One of the most useful features of MATLAB® is that it will apply the function to all elements of the array passed as input, which is known as vectorization. In addition to arithmetic functions, there are functions to create figures and plots. To bring up a figure window, simply type 'figure'. To create an X,Y plot, use the plot function and pass a vector for the x values followed by a vector for the y values. (e.g. plot(t, sin(t))] Note: The vectors must have the same dimensions. Also, subsequent calls to plot will replace the current plot. You can open a new figure using the 'figure' command with future plots being sent to that new window.

> **Important**: Care must be taken when multiplying and dividing vectors and matrices. Using the * or / operator on matrices or vectors implies that matrix multiplication or division should be performed (e.g. matrix multiplication can only be performed on matrices of dimension MxK and KxN resulting in a MxN matrix). This is different from the point-wise multiplication (.* or ./)of two vectors or matrices which is multiplies or divides the elements at corresponding locations in a vector or matrix.

```
>> sin(pi/2)          % Returns a scalar with the result of sin(π/2)(i.e. 1)
>> sin( [ 0  pi/2  pi  3*pi/2  2*pi]) % Returns the vector [0 1 0 -1 0]
>> t = [0  pi/2  pi  3*pi/2  2*pi]); sin(t) % A vector variable can be passed as well
>> 2*sin(t)                  % A scalar times a vector will multiply each element in
                             %  the vector with the scalar value (i.e. [0 2 0 -2 0])
>> figure; plot(t, 2*sin(t)) % Plot will connect each point with a line segment creating
                             % a triangle wave
>> A = [1 2 3 4];  A*sin(t)  % Error!  This attempts to perform matrix multiplication
                             % Can't perform matrix mult. of a 1x4 * 1x4 vector.
>> A = [1 2 3 4]; A .* sin(t)% We want the point-wise multiplication. Yields [0 2 0 -4].
>> plot(t, A .* sin(t))      % We can nest function calls w/in others
>> abs(sin(t))               % We can nest functions… First the sine of the vector t is
                             %  taken resulting in another vector whose absolute value
                             %  is then returned
```

f **Sampling of functions**: A key issue when using a computer to model signals and mathematical functions is that computers are finite (discrete) machines. Even the simplest continuous function cannot be numerically represented with a computer because there are an infinite number of points in any interval of the domain. Instead, the usual method for representing signals is to select a discrete number of points and evaluate the function only at those points. By selecting enough points, continuous functions can be accurately modeled and processed. This process is known as *sampling* and will be discussed in detail later in the semester. The time or interval between samples is known as the *sampling period,* T_s. The reciprocal of the sampling period is known as the *sampling*

frequency, F_s. Below are some examples that will show you how to sample some given functions. You will see that to sample a function, we usually create a vector of points that we want to sample and usually the sampling period is usually the step size.

Example 1: Enough sampling points

```
>>  Ts = pi/2; t = [0:Ts:2*pi]; plot(t,sin(t))% Ts = π/2 is not enough points to
                                               %  visually distinguish a sine wave
                                               % plot() takes two vectors.  The first is
                                               % interpreted as the x values and the
                                               % second as the y values
>> t1 = [0:0.01:2*pi]; plot(t1,sin(t1))        % Ts = 0.01 is enough points to
                                               %  visually distinguish a sine wave
```

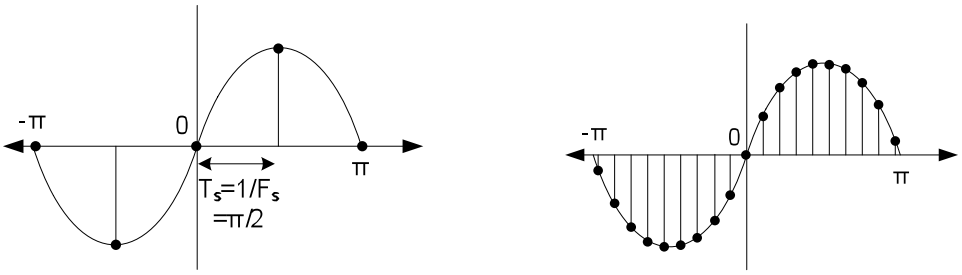


Figure 1 - Sampling of a continuous sine wave. The time between samples is known as the sampling period. Its reciprocal is the sampling frequency. More samples yields a greater approximation of the continuous function

Q4a. Sampling: Do the above example in your command window. Additionally add labels to the x axis, y axis using the commands xlabel and ylabel (see >>help xlabel). Add a suitable title to the plot you have generated by using the title command.

Q4b. Sampling: Is there a way in MATLAB® to obtain the two plots on the same figure? If so, create a new figure and plot the two graphs on the same figure using different colors.

Example 2: Another example

```
>> fs = 8000;                % Sampling frequency is 8000 samples/sec.
>> tstart = 0; tend = 2;     % We will plot our signal from t=0 sec. to 2 sec.
>> t = [tstart : 1/fs : tend];
>> f = 3;  w = 2*pi*f;       % Our signal will be a sine wave with freq. = 3 Hz
>> plot(t,sin(w*t));  % We should see 6 cycles of a sine wave from 0-2 seconds
```

Example 3: Adding amplitude effects (amplitude increases linearly from 1 to 6 over the 2 second interval)

```
>> astart = 1; aend = 6;
>> da = (aend-astart)/(length(t)-1)  % We need one amplitude for each sampled
                                     % point of our sine wave so we set da equal to
                                     % the slope of our linear amplitude
>> A = [astart : da : aend];
>> plot(t, A .* sin(w*t));           % Be sure to perform point-wise multiplication
```

Q4c. Amplitude Effects: Do the above exercise in your MATLAB® command window.

g **M-file scripts and programming constructs**: Rather than entering commands one at a time at the MATLAB® prompt, MATLAB® can read in commands from a script file. These script files are known as M-files and will be used in most of these labs. They provide a simple way to execute a group of statements with one click or command. These script files also facilitate the use of MATLAB's basic programming constructs (loops, conditional statements, etc.). These constructs are similar to a typical general purpose programming language. Occasionally, they will be necessary for your labs. One of the most common constructs is the 'for' loop which will repeat a series of statements for each element in a vector. As an example, let us use a 'for' loop to calculate the derivative of a sine function. Open the MATLAB® editor by clicking File..New..M-file. Save the file as ee301_forloop.m. Add the directory that you saved the script to the MATLAB® path (click File..Set Path and add the folder where you saved your script). When done entering the following script (you do not need to enter the comments after a statement), run the script by clicking the Run button on the Editor window or typing the name of the file (without the .m extension) at the MATLAB® command prompt (e.g. >> ee301_forloop)

```
% ee301_forloop.m

fs = 8000;  dt = 1/fs;
t = [0 : dt : 2];
X = sin(w*t);
for i = i:length(X)-1                % iterate once for all but the last elements of X
  deriv(i) = (X(i+1) - X(i)) / dt;   % derivative = slope between two adjacent points
end
deriv(length(X)) = deriv(length(X)-1);%set the last value of the derivative equal
                                     % to the second to last value
plot(t,deriv);                       % Ensure that this resembles a cosine function
                                     %  and note its amplitude
```

Q5. Loops

We will examine the effect of adding several sine waves with frequencies equal to multiples of a fundamental frequency (i.e. f, 3*f, 5*f, ...).

1 Create an M-file named ee301_lab1.m. The following lines have been completed for you. Copy and paste them in your M-file.

```
fs = 8000; dt = 1/fs;
t = [0 : 1/fs : 3];          % We'll sample for 3 seconds
f = 1;                       % f = 1 Hz is the fundamental frequency
X1 = sin(2*pi*f*t);
X3 = sin(2*pi*3*f*t)/3;
X5 = sin(2*pi*5*f*t)/5;
X7 = sin(2*pi*7*f*t)/7;
```

The first two generate a vector, t, representing time from 0 to 3 seconds, sampled at 8000 samples per second (i.e. a stepsize of 1/8000). The next two lines generates a sine wave of with fundamental frequency f = 1 Hz ($\omega = 2\pi f$) from time 0 to 3 seconds. Finally the next 3 lines generate 3 more sine waves with odd multiples of the fundamental frequency (f = 3Hz, 5Hz, and 7 Hz) and whose amplitudes are divided by those same multiples.

2 Plot the sine wave of frequency f = 1Hz. Add suitable labels and title.

3 Now add all four sine waves together to create a composite signal, Z and plot Z vs. t.

4 The question we want to answer is what would a composite signal look like if we added more and more sine waves of odd multiple frequencies (i.e. f = 9, 11, 13 Hz, etc.). Take a second and try to make an educated guess as to what you think this signal would look like. Now, compute the actual result by writing a simple for loop to calculate this composite signal formed from frequencies 1,3, 5,...29,31 Hz. Use the lines below to get you started...and remember to divide each sine wave's amplitude by its frequency.

```
Z1 = zeros(1,length(t));
for f = 1:2:31               % f will take on values 1,3,5,…,31 during the loop
  your code here
end
```

5 Now, plot the resulting signal (should be Z1) vs. t. Was your guess correct? You should see a plot that begins to resemble a square wave (high level followed by a low level). In fact, to create a perfect square wave would require summing infinitely many sine waves.

h **The Heaviside step function**

We can plot the unit step function using the command heaviside(). For example replicate the following code in your Matlab command window:

```
>>t = -5:0.01:10;
>>u = heaviside(t);
>>plot(t,u)
>>ylim([-0.3 1.3])
```

Notice that the command ylim() set the limits of y on the plot.

Q6. Unit step function: Plot the continuous-time signal x(t) = u(t+1)-u(t-2)+u(t-4), using the command heaviside() and setting the max limit of y on the plot to 1.1 and min limit of y on the plot to -0.1. Make sure that you can clearly see the entire signal on your plot.

Q7. Even and Odd functions: Consider the signal $x(t) = te^{-t}$, $0 \le t \le 5$. Plot the signal x(t), the even decomposition $x_e(t)$ of x(t), the odd decomposition $x_o(t)$ of x(t) and the signal $y(t) = x_e(t) + x_o(t)$. Use the subplot() command to plot all four plots at once.

i **Numerical Integration**

There are cases where calculating an integral by hand is not an easy task (even for mathematicians). The signals we encounter in practical systems are often described in terms of complicated mathematical expressions whose integrals are very hard (or even impossible) to compute. Fortunately, there is a way to calculate any definite integral (up to desired accuracy) using MATLAB. This approach is called *numerical integration*.

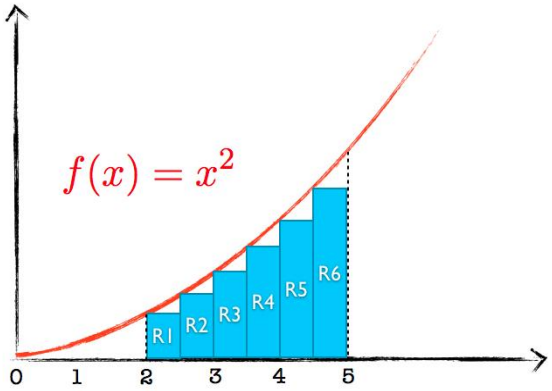
Say you want to calculate:

$$A = \int_2^5 x^2 dx$$

Of course, x^2 is not a "complicated mathematical expression" but we are going to use it here as an example. It is easy to calculate by hand that:

$$A = \left[\frac{x^3}{3}\right]_2^5 = \frac{5^3}{3} - \frac{2^3}{3} = 39$$

Recall that an integral calculates the area below a curve. The following figure shows the basic idea behind numerical integration. A simple way to approximate the area below $f(x) = x^2$ from 2 to 5 is to sum the six rectangular areas shown in blue.



Lets calculate: The area of a rectangle is the length of its base times its height. Notice that we took the base of all rectangles equal to Δx = 0.5 and hence we only need to calculate their heights. But the height of a rectangle can be given directly from the function f. The height of R1 is f(2), the height of R2 is f(2.5) the height of R3 is f(3) etc. So the only thing we need is to sample the function f from 2 to 5 in intervals of 0.5 (in matlab notation this is [2:0.5:4.5]).

The area is:

$$\begin{aligned} A &\simeq \Delta x \cdot f(2) + \Delta x \cdot f(2.5) + \Delta x \cdot f(3) + \Delta x \cdot f(3.5) + \Delta x \cdot f(4) + \Delta x \cdot f(4.5) \\ &= 0.5 \cdot \left(2^2 + 2.5^2 + 3^2 + 3.5^2 + 4^2 + 4.5^2\right) \\ &= 0.5 \cdot 67.75 = 33.875 \end{aligned}$$

Comparing this with the exact result calculated above (A=39), we see that there is room for improvement. The reason our approximation is not very good is that we didn't account for the "empty space" between the curve and the rectangles (See figure). There is a way, however, to reduce that empty space and increase our accuracy by taking smaller and smaller intervals Δx. In fact, as Δx->0 the above method will converge to the exact result.

Here is the code for the numerical integration of x^2:

```
Tstart=2;
Tend=5;
Nr=6;                          %number of rectangles
Dx=(Tend-Tstart)/(Nr);
t=Tstart:Dx:Tend-Dx;           %sampling points
fs=t.^2;                       %our function evaluated at points t
A=sum(Dx*fs)                   % Dx*fs(1) + Dx*fs(2) + ...
```

Q8. Numerical Integration: Run the code given in the example. "A" should be equal to 33.875 as calculated above. The relative absolute error is in this case equal to: E(6) = |39 - 33.875| = 5.125. Increase the number of rectangles to 10. Is the number closer to 39? What is E(10)? Calculate and plot E(N) for N=10 to 100. (Hint: E should be a vector of the appropriate length). What do you observe? How many rectangles are required to achieve a relative error less than 0.5?
(Hint: if 'v' is a vector, p = find(v<c) returns to 'p' the indices i of v such that v(i)<c. E.g, if v=[1,5,6,2,5] then p=find(v<5) will be [1,4])

★ Extra credit

1) Calculate the energy of the following signal:

$$f(t) = \begin{cases} \log(t+1)\sin\left(t^2/3\right)e^{-\cos\left(2\pi\sqrt{t}\right)}, & 0 \le t \le 10 \\ 0, & otherwise \end{cases}$$

2) Plot f(t) together with a pulse signal (i.e, a signal p(t) that is constant for all t in [0,10] and zero everywhere else) that has the same energy.

5 Review

1 None.

6 Lab Report

1 The first page of your Lab report should be a cover sheet with your name, USC ID and Lab #. Please note that all reports should be typed.
2 Answer all the questions which were asked in the lab report. Kindly display the code lines you executed to arrive at your answer along with figures to support them. Please give written explanation or put comment lines where necessary. Please note that each figure should have proper labels for the x and y axis and should have a suitable title.
3 Answer the review questions.
4 Submit a printout of your completed M-file documenting all the lab exercises.