**Department of Electrical Engineering**

|  |  |
| --- | --- |
| **Faculty Member: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Dated: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
|  |  |
| **Course/Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Semester: Fall 2020** |
|  |  |

**EE-330 Digital Signal Processing**

**Lab1: MATLAB REVIEW-Signals & Systems Fundamentals**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4-CLO4** | | **PLO5-CLO5** | **PLO8-CLO6** | **PLO9-CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Team Work** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Lab1: MATLAB REVIEW-Signals & Systems Fundamentals**

**Objectives**

The purpose of this lab is to review the fundamentals of signals and systems with MATLAB, particularly:

* Signal transformations (shifting, inversion, scaling)
* Even and Odd parts of a signal
* Convolution operator-the basic property of Linear Time Invariant (LTI) Systems

**Lab Instructions**

* The students should perform and demonstrate each lab task separately for step-wise evaluation Each group shall submit one lab report on LMS within 5 days after the lab is conducted. Lab report submitted via email will not be graded.
* Students are encouraged to practice on their own in spare time to enhance their skills.

**Lab Report Instructions**

All questions should be answered precisely to get maximum credit. Lab report must ensure following items:

* Lab objectives
* MATLAB codes
* Results (graphs/tables) duly commented and discussed.
* Conclusion

## Matrices/vectors in MATLAB

(a) Make sure that you understand the **colon** notation. In particular, explain in words what the following MATLAB code will produce

jkl = 0 : 6;

jkl = 4 : 4 : 17;

jkl = 99 : -1 : 88;

ttt = 2 : (1/9) : 4;

tpi = pi \* [ 0:0.1:2 ];

(b) Extracting and/or inserting numbers into a vector is very easy to do. Consider the following definition of xx:

xx = [zeros(1,3), linspace(0,1,5), ones(1,5)];

[s1 s2] = size(xx);

s3 = length(xx);

Explain the results echoed from the last four lines of the above code.

What’s the difference between a length and a size statement for a matrix? To test this define a matrix X with arbitrary inputs, having multiple rows and columns and test the output of length() and size() function on it.

(c) Assigning selective values in a matrix differently. Comment on the result of the following assignments:

yy = xx;

yy(4:6) = pi\*(1:3);

## Creating a M-file

Go to File > New > M–file. MATLAB editor will open up. Enter the following code in the editor and then save the file as Namelab1.m

tt = -1 : 0.01 : 1;

xx = cos( 5\*pi\*tt );

zz = 1.4\*exp(j\*pi/2)\*exp(j\*5\*pi\*tt);

plot( tt, xx, ’b-’, tt, real(zz), ’r--’ ), grid on

%<--- plot a sinusoid

title(’TEST PLOT of a SINUSOID’)

xlabel(’TIME (sec)’)

Now go to Command Window and type

mylab1 %<---will run the commands in the file

type mylab1 %<---will type out the contents of

% mylab1.m to the screen

## Functions-Key to Efficient Coding

It is often convenient to define functions so that they may used at multiple instances and with different inputs. Functions are a special type of M-file that can accept inputs (matrices and vectors) and may return outputs. The keyword ***function*** must appear as the first word in the M-file that defines the function, and the first line of the M-file defines how the function will pass input and output arguments. The file extension must be lower case “***m***” as in my ***func.m***. The following function has a few mistakes. Before looking at the correct one below, try to find these mistakes (there are at least three):

*Matlab mfile [xx,tt] = badcos(ff,dur)*

*%BADCOS Function to generate a cosine wave*

*% xx = badcos(ff,dur)*

*% ff = desired frequency in Hz*

*% dur = duration of the waveform in seconds*

*tt = 0:1/(100\*ff):dur; %-- gives 100 samples per period*

*badcos = cos(2\*pi\*freeq\*tt);*

*The corrected function should look something like:*

*function [xx,tt] = goodcos(ff,dur)*

*tt = 0:1/(100\*ff):dur; %-- gives 100 samples per period*

*xx = cos(2\*pi\*ff\*tt);*

Notice the word “function” in the first line. Also, “freeq” has not been defined before being used. Finally, the function has “xx” as an output and hence “xx” should appear in the left-hand side of at least one assignment line within the function body. The function name is *not* used to hold values produced in the function.

## Review of Basic Signals and Systems

**a) Even and odd parts of a signal:**

Any signal ***x[n]*** can be decomposed into its even part and odd parts as:

Write a simple MATLAB code (in the form of a function) that allows you to decompose a signal into its even and odd parts.

***Note:*** The function takes two inputs ***n***, the timing index and ***x*** the values of the signal at the designated time instants. The function outputs include the two sub-functions, x\_e and x\_o along with the timing index.

Test your function on the following signal ***x[n]*** and compute its even and odd parts.

**b) First order Difference equation:**

Recall that one way of defining the LTI systems is through the difference equations that relate the input ***x[n]*** to the output ***y[n]***.

Consider the first order system defined by the difference equation as follows (we’ll review the discussion on how determination of order for a difference equation later):

***y[n] = a. y*[*n-1*] *+ x*[*n*]**

Write a function ***y = diffeqn* (*a, x, y[-1]*)** which computes the output ***y[n]*** of the system determined by the given equation. The vectors ***x[n]*** contains the signal as defined in the upper part and ***y[n] = 0 for n < 1***.

**c)** **Convolution of signals**

Recall that one the most convenient ways to represent an LTI system is through its impulse response ***h[n]***. Once the impulse response of a system is known, the output (response) of the system to any given input can be computed using the convolution operator as:

****

The convolution essentially involves two operations: flipping either the input signal or the impulse response (as in above equation) and then sliding the flipped signal.

1. Write your own convolution function, ***myconv.m*** that computes the convolution between the two signals (or the output of passing an input signal through a system). Designate all the necessary inputs for your function, considering that the input signal and the impulse response may start at some ‘***n***’ that is negative. The function output is obviously the system output along with the timing index for the output ***n1***, which must be set manually. Your function should work on any general signal and the impulse response (of finite length).
2. Test your function on the signal and the impulse response provided in the figures below and verify the correctness of your function through a comparison of manual computation of the convolution for the given signal and a plot of your function’s output.



1. MATLAB has a built-in function ‘***conv***’ that performs the same operation. Compare the results of part (ii) with the ***conv*** function of MATLAB.
2. Consider now that ***x[n]*** starts from ***n = -1*** and ***h[n]*** starts from ***-2***. What will be the result of convolution then? Plot the corresponding output signal using the stem command and proper timing axis.