**Department of Electrical Engineering**

|  |  |
| --- | --- |
| **Faculty Member: Mr. Kalimullah** | **Dated: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
|  |  |
| **Course/Section: BEE8D** | **Semester: Spring 2019** |
|  |  |

**EE-330 Digital Signal Processing**

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4** | | **PLO5** | **PLO8** | **PLO9** |
| **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** |
| **Muhammad Haris Rehman** | **182797** |  |  |  |  |  |
| **Muhammad Anus Siddiqui** | **174308** |  |  |  |  |  |
| **Muhammad Hasnain Afridi** | **192414** |  |  |  |  |  |
|  |  |  |  |  |  |  |

**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
* Periodisation of a finite duration signal using convolution operator

**Lab Instructions**

* The students should perform and demonstrate each lab task separately for step-wise evaluation (please ensure that course instructor/lab engineer has signed each step after ascertaining its functional verification)
* Each group shall be submitted 1 Lab report on LMS within 6 day after each lab is conducted .Only lab report submitted on LMS will be graded no email will be accepted via email.
* Students are however encouraged to practice on their own in spare time for enhancing 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 = 2 : 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(1,4)];

[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.

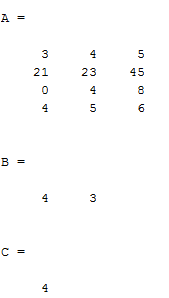
**Code**

A = [3 4 5; 21 23 45; 0 4 8;4 5 6 ]%declaring a matrix

B=size(A)%using size command which returns no of row and columns respectively

C=length(A)% returns largest dimension

**Screenshot**



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

yy = xx;

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

**code**

xx = [zeros(1,3), linspace(0,1,5), ones(1,4)] % declaring a matrix

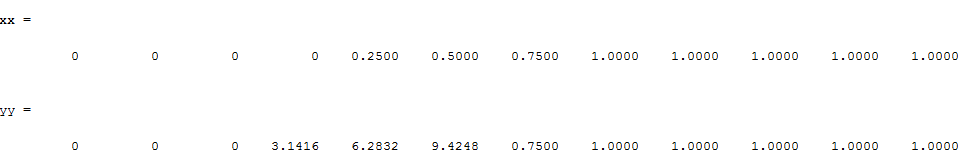
[s1 s2] = size(xx); % returning no of rows and columns

s3 = length(xx); % returning largest dimension

yy = xx % equating matrix

yy(4:6) = pi\*(1:3) % assigning new values in yy from 4 to 6

**screenshot**



## 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)’)

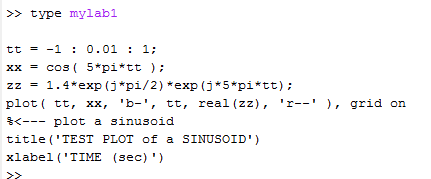
Now go to Command Window and type

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

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

% mylab1.m to the screen

**Screenshot**



## 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.



x = [2 3 -1 2 -3];

n = 0:length(x)-1;

[x\_e, x\_o, nl] = fun(n, x);

subplot(211), stem(nl, x\_e), grid

subplot(212), stem(nl, x\_o), grid

**Function:**

function [x\_e, x\_o, n1] = fun(n, x)

tl = length(n) - 1;

if n(length(n) < -n(1))

tl = -n(1);

end

n1 = -tl:tl;

xn = n1;

for i = 1:length(xn)

xn(i) = 0; % x[all n] = 0

end

x\_n = xn;

for i = n(1)+1:n(length(n))+1

xn(i+tl) = x(i);

end

x\_n = flip(xn);

for i = 1:length(xn)

x\_e = 0.5\*(xn + x\_n);

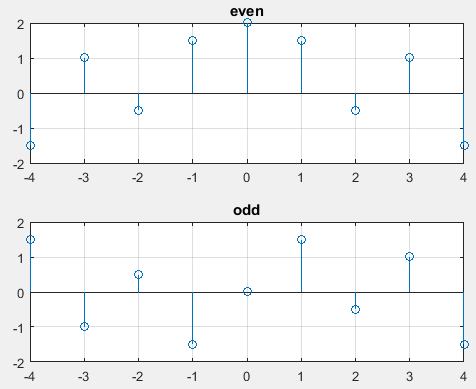
end

for i = 1:length(x\_n)

x\_o = 0.5\*(xn - x\_n);

end

end



**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***.

function[y]=firstorder(a,x,y1)% declaring function

y=zeros(1,length(x));

for i=1:length(x)% intilizing loop

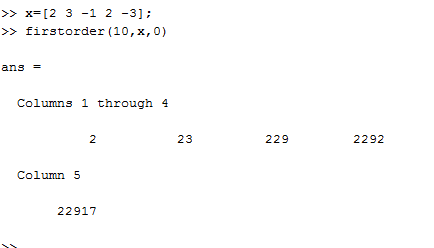
y(i)=a\*y1+x(i);

y1=y(i);

end

end

**screenshot**

****

**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).

function [ y,index\_y ] = myconv( x,h,index\_x,index\_h )

xl=length(x);

hl=length(h);

yl=xl+hl-1;

yz=zeros(hl,yl);

fori=1:hl

yz(i,i:(xl+i-1))= h(i)\*x;

end

y=sum(yz,1);

y\_start=index\_x(1) +index\_h(1);

y\_end= y\_start + yl-1;

index\_y= y\_start:y\_end;

subplot(3,1,1), stem(index\_x,x),title('Input Signal');

subplot(3,1,2), stem(index\_h,h),title('Impulse Response');

subplot(3,1,3), stem(index\_y,y),title('Output Signal');

end

1. 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.



>> x=[0 1 2 4 1 1];

>> index\_x=[0 1 2 3 4 5];

>> h=[1 2 1 2];

>> index\_h=[0 1 2 3];

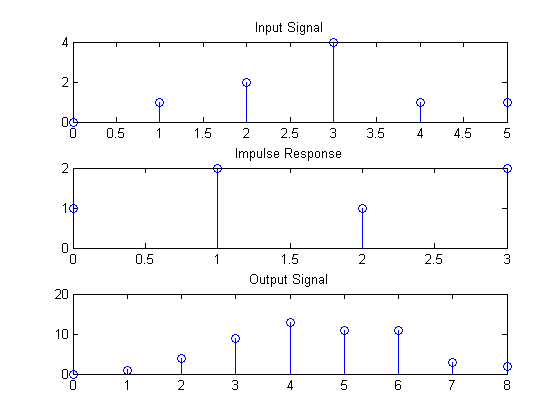
>> myconv(x,h,index\_x,index\_h);

>> y =

0 1 4 9 13 11 11 3 2

>> index\_y =

0 1 2 3 4 5 6 7 8



The resulting output and the index array is the same as that calculated by manual method.

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.

>> x=[0 1 2 4 1 1];

>> h=[1 2 1 2];

>> conv(x,h)

ans =

0 1 4 9 13 11 11 3 2

The convolution calculated by the “**myconv**” is the same as that calculated by the “**conv**” function.

1. 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.

>> x=[0 1 2 4 1 1];

>> index\_x=[-1 0 1 2 3 4];

>> h=[1 2 1 2];

>> index\_h=[-2 -1 0 1];

>> myconv(x,h,index\_x,index\_h);

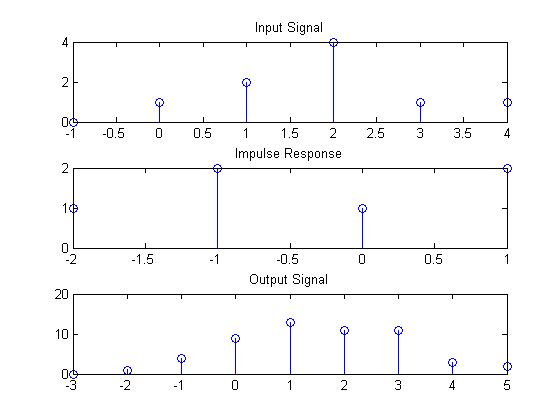
>> y =

0 1 4 9 13 11 11 3 2

>> index\_y =

-3 -2 -1 0 1 2 3 4 5

The above results are accurate and the output signal plot with proper time axis is given below:

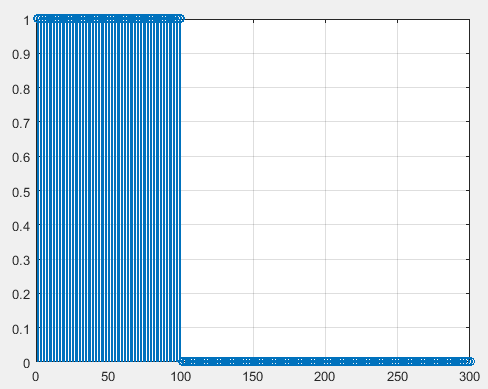


**d)** **Convolution of signal with an impulse train**

1. Define ***x[n]*** as a very simple signal starting at ***n=0*** and containing a sequence of ones in its first 100 samples and a sequence of zeros in its next 200 samples. You may like to use the MATLAB functions ***zeros()*** and ***ones()***.
2. Plot this discrete time signal (use ***stem***) as a function of the timing variable ***n***.

xx = [zeros(1,100) ones(1,200)];

plot(xx), grid



Now consider an impulse train defined mathematically as:

**,**

where ***N*** represents the period of this impulse train. A particular example of this train is shown below for ***N = 3*** and for ***k =-3:3***.

1. Write a function that lets the user generate an impulse train for a given ***N*** and given range of ***k***, i.e. the function takes at its input ***N*** and range of ***k*** (or one positive value and define within your function array from ***–k:k*** to get the whole range) and outputs the impulse train in an array along with a timing index.
2. Use the function that you just defined to generate an impulse train with ***N = 300*** and ***k =-2:2.*** Plot the impulse train as a function of time index that would be generated.

N = 300;

k = 2; % given in question

y = zeros(1, k\*N+k\*N+1); % total length of train

mid = (length(y) + 1) / 2; % mid point of train

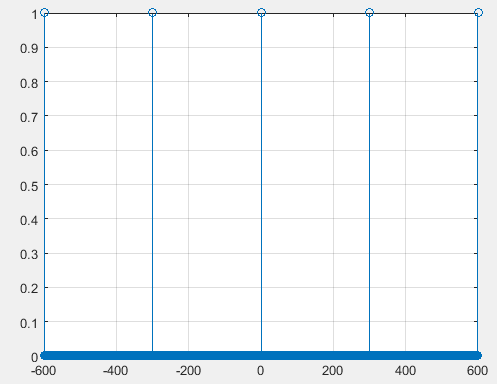
y(mid:N:end) = 1; % ones on right side of train

for i = mid:-N:1

y(i) = 1; % ones on left side of train

end

stem(-N\*k:N\*k , y), grid



1. Now comes the interesting part, convolve the signal generated in **(i)** with the impulse train you generated in **(iv)**.Plot the output of the convolution. How’s this output related to the original input you plotted in **(i)**?

x = [ones(1,100) zeros(1,200)];

N = 300;

k = 2; % given in question

y = zeros(1, k\*N+k\*N+1); % total length of train

mid = (length(y) + 1) / 2; % mid point of train

y(mid:N:end) = 1; % ones on right side of train

for i = mid:-N:1

y(i) = 1; % ones on left side of train

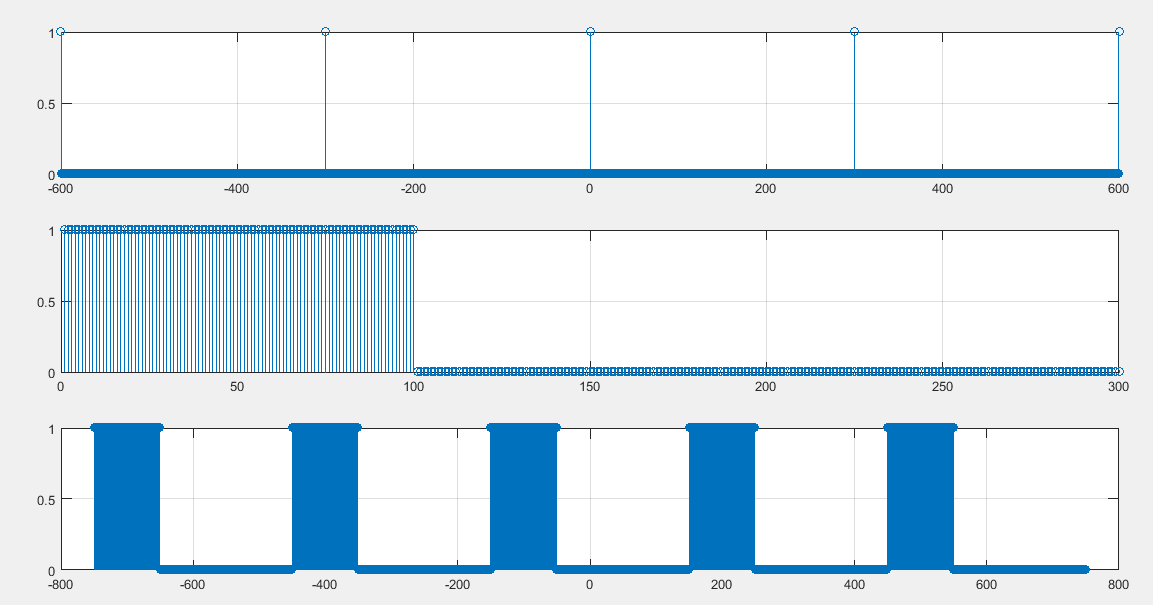
end

R = conv(x, y);

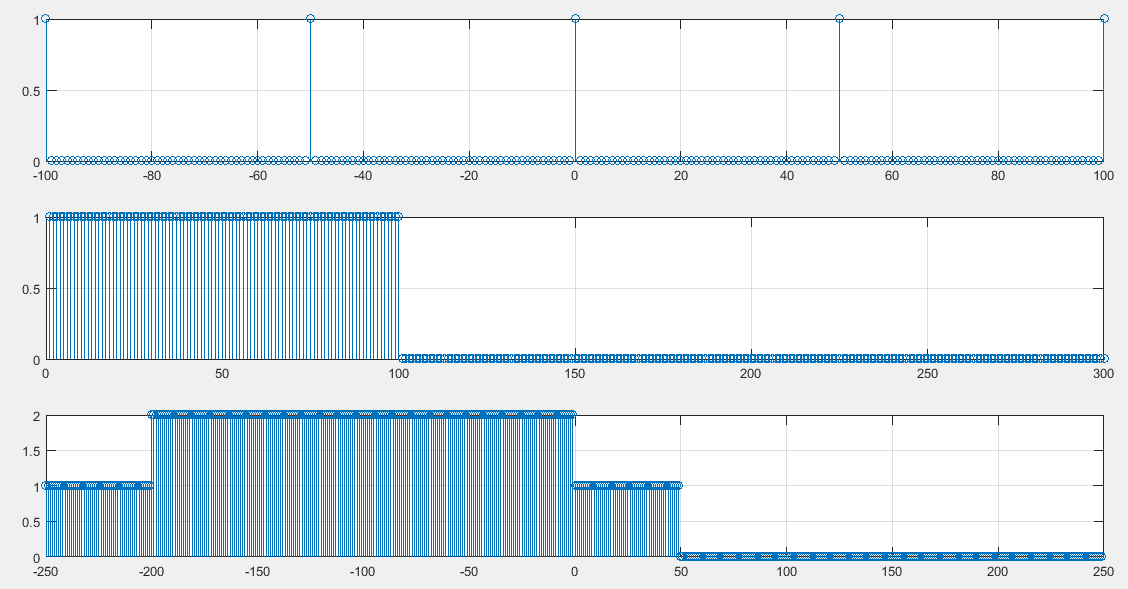
subplot(311), stem(-N\*k:N\*k , y), grid

subplot(312),stem(x), grid

subplot(313),stem(-k\*N-300/2:k\*N+300/2-1, R), grid



1. Repeat **(iv)** and **(v)** with **N = 50** for the same input as in **(i)**. Plot the corresponding outputs. Could you still establish a similar relation as in **(v)**.



1. Based on the results of **(v)** & **(vi)**, comment how could you obtain the periodic version of a time limited input signal and if there’s any constraint on the value of ***N*** with reference to the input signal.

***Ans:*** *For the complete periodic version, the length of the function must be less than or equal to the Period of impulse train. Otherwise, it will overlap.*

1. Load a sample audio clip (.wav file) in MATLAB and generate its periodic version (repetition) using the process above without causing any overlap on the original audio content.

[x,Fs] = audioread('z.wav');

tr = 1:length(x)\*4+1;

tr(1:end) = 0;

mid = (length(tr) + 1) / 2;

tr(mid:length(x):end) = 1;

for i = mid:-length(x):1

tr(i) = 1;

end

y = conv(tr, x(1:length(x)));

sound(y, Fs)

