National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering and Computer Science

Faculty Member: Dr. Ahmad Salman **Dated:** 06/02/2023

Semester: 6th **Section:** BEE 12C

EE-330: Digital Signal Processing

Lab 1: MATLAB REVIEW- Signals and Systems Fundamentals

Group Members

Name	Reg. No	Lab Report Marks	Viva Marks	Total
		10 Marks	5 Marks	15 Marks
M. Ahmed Mohsin	333060			
Hassan Rizwan	335753			
Syeda Fatima Zahra	334379			

1 Table of Contents

Signals and Systems Fundamentals

1.1 Objectives

The purpose of the lab is as follows:

- Reviewing fundamentals of signals and systems using MATLAB
- Performing signal transformations
- Isolating even and off parts of a signal
- Implementing convolution

1.2 Introduction

In this lab reviews various basic functions of MATLAB used in signals and system analysis, particularly the colon notation, creating functions and how to implement convolution. This was an introductory lab to familiarize students with matrices, functions and loops in MATLAB.

1.3 Software

MATLAB R2022b

Lab Exercises

2.1. 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;

Answer: This code will produce a vector jkl starting from 0 till 6, with an increment of 1 as the default step size is 1 in MATLAB.

jkl= [0 1 2 3 4 5 6]

jkl = 4 : 4 : 17;

Answer: This code will produce a vector jkl starting from 4 and increments by 4 uptil 17.

jkl= [4 8 12 16]

jkl = 99 : -1 : 88;

Answer: This code will produce a vector jkl starting from 99 and decrementing by 1 uptil 88.

jkl=[99 98 97 96 95 94 93 92 91 90 89 88]

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

Answer: This code will produce a vector starting from 2 incrementing by 1/9 till 4.

ttt = [2.0000 2.1111 2.2222 2.3333 2.4444 2.5556 2.6667 2.7778 2.8889 3.0000 3.1111 3.2222 3.3333 3.4444 3.5556 3.6667 3.7778 3.8889 4.0000]

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

Answer: This code will produce a vector tpi created by a vector starting from 0 and incremented by 0.1 until 2, multiplied by pi.

tpi= [0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850 2.1991 2.5133 2.8274 3.1416 3.4558 3.7699 4.0841 4.3982 4.7124 5.0265 5.3407 5.6549 5.9690 6.2832]

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?

Answer: The above lines of code do the following:

Create a vector xx containing zeros in the first 3 columns (zeros(1,3)), followed by 5 equally spaced vectors starting from 0 uptil 1 (linspace(0,1,5)), followed by 5 columns of 1 (ones(1,5)).

xx = [0 0 0 0 0.2500 0.5000 0.7500 1.0000 1.0000 1.0000

1.0000 1.0000 1.0000]

The next line stores the size of xx in two variables s1 and s2. The size function returns the size of the vector xx in the format rows x columns. The code stores the value of number of rows in s1 and number of columns in s2.

s1=1 s2=13

The next line stores the value of length of xx in s3. The function length returns the total number of elements in the vector xx.

s3=13

The length statement returns a single value whereas the size statement returns a vector of the type [numofrows numofcolumns].

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

```
yy = xx;
```

Answer: This line of code stores all the values of the vector xx in the vector yy.

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

Answer: This line of code assigns the 4^{th} uptil the 6^{th} element of yy values returned by the vector pi*[1 2 3].

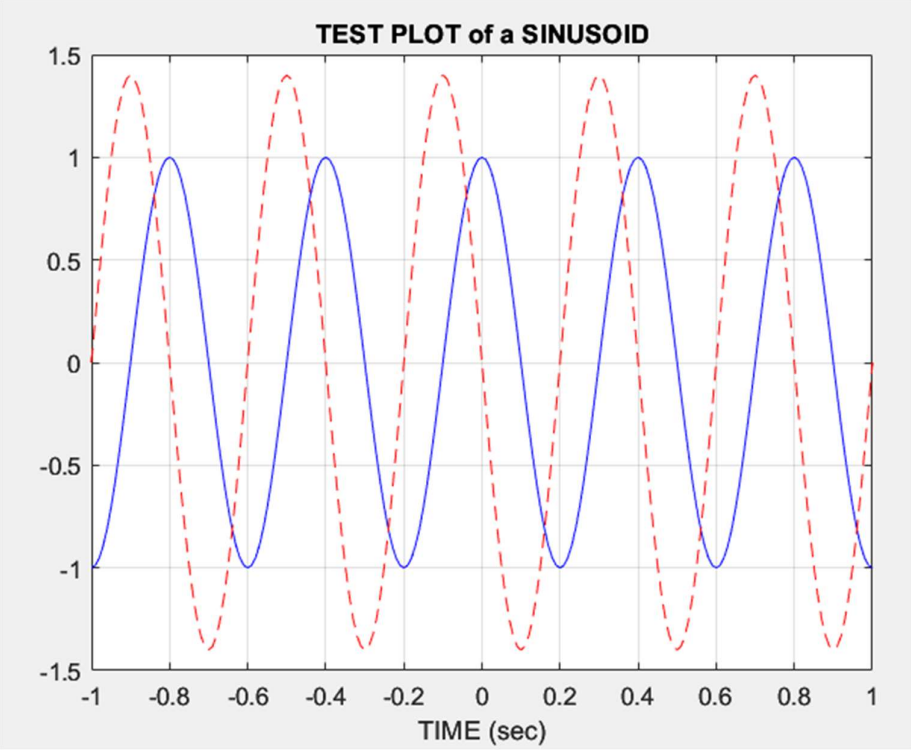
2.2. Creating an M-file

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

Code:

```
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)')
```

Output:



2.3. Functions

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);*

Faults:

- You can't call the function inside the function definition, i.e. *xx = badcos(ff,dur).*
- Freeq is not a variable that exists for the function, i.e. *badcos = cos(2*pi*freeq*tt).*
- You cant use the function name as an assignment, i.e. *badcos = cos(2*pi*freeq*tt.*
- The function should return the variable xx atleast once inside the declaration.

Corrected function:

```
function [xx,tt] = goodcos(ff,dur)
tt = 0:1/(100*ff):dur; %-- gives 100 samples per period
xx = cos(2*pi*ff*tt);
end
```

2.4. 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:

$$x_e(n) = \frac{1}{2}[x(n) + x(-n)]$$
$$x_0(n) = \frac{1}{2}[x(n) - x(-n)]$$

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.

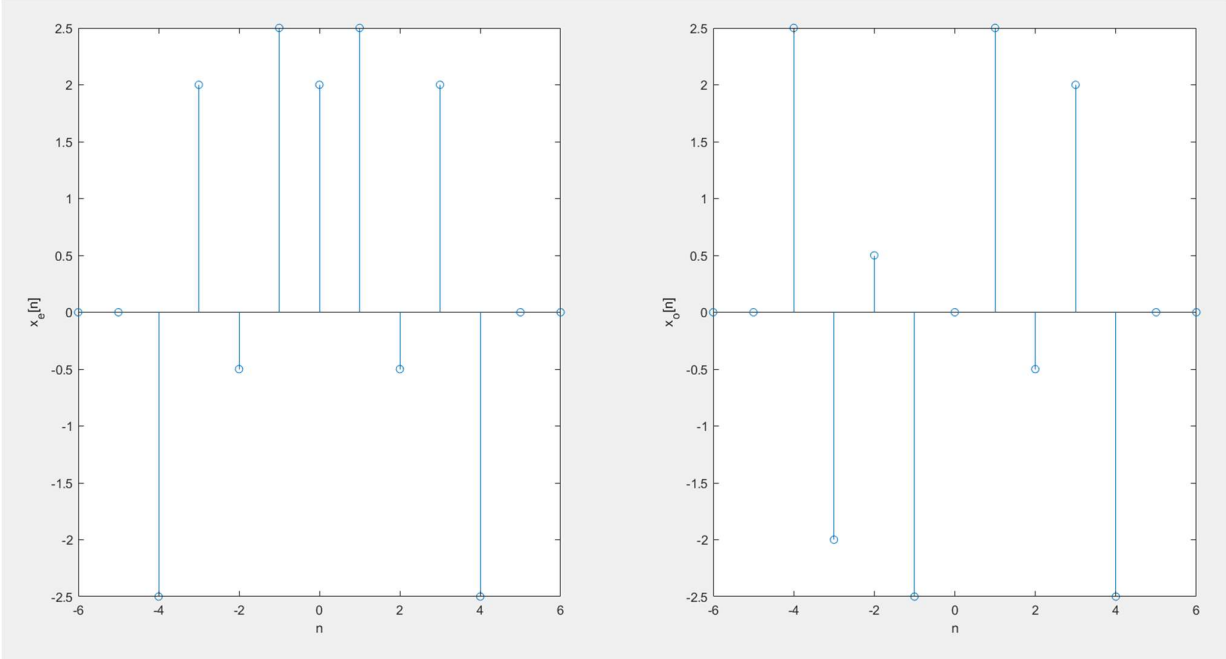
$$x[n] = \begin{cases} 2 & n = 0 \\ 5 & n = 1 \\ -1 & n = 2 \\ 4 & n = 3 \\ -5 & n = 4 \\ 0 & \text{elsewhere} \end{cases}$$

Code:

```
function [x_e, x_o] = Even_Odd(n, x)
    x_ = fliplr(x); % Used to invert in the time domain

    if (sum(ismember(n, 0)) ~= 0)
        idx_zero = find(n == 0);
        LeftPadding = zeros(1, (length(n) + 1) - (2 * idx_zero)) % Padding
zeros on left
        RightPadding = zeros(1, (2 * idx_zero) - (length(n) + 1)) % Padding
zeros on right
    else
        if (n(1) > 0)
            LeftPadding = zeros(1, length(0:n(1)) + 1)
            RightPadding = []
        else
            LeftPadding = []
            RightPadding = zeros(1, length(n(length(n)):0) + 1)
        end
    end

    x = [LeftPadding, x, RightPadding]
    x_ = [RightPadding, x_, LeftPadding]

    x_e = (1 / 2) * (x + x_) % Even signal Extracted
    x_o = (1 / 2) * (x - x_) % Odd Signal Extracted


n = [0:1:6]; % Indices for timing input
x = zeros(1,length(n));  % X is the INPUT signal
shift = 1 - n(1);
x(shift:shift+4) = [2 5 -1 4 -5 ];
[e, o] = decompose(n,x)
n_out = (-6:6);
subplot(121)
stem(n_out,e); % plotting the even signal
xlabel('n');
ylabel('x_e[n]');
subplot(122)
stem(n_out,o); % plotting the odd signal
xlabel('n');
```

```
ylabel('x_o[n]');
end
```

OUTPUT:



b. First order Difference equation:

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

$$y[n] = a.\, y[n-1] + x[n]$$

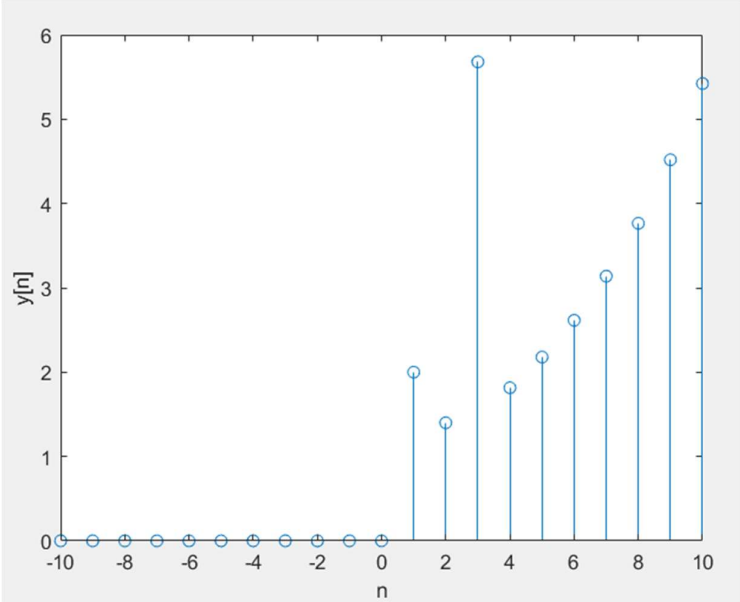
Write a function ***y = diffeqn (a, x, y[-1])*** which computes the output ***y[n]*** of the system determined by the given equation. The vectors ***x[n]*** contains the signal as defined in the upper part and ***y[n] = 0 for n < 1***.

Code:

```
function y = diffeqn(a,x,y_init)
shift = 11; % Taking a set number of shifts
Samples = length(x); % Accounting the length for the samples
y = zeros(1,Samples); % Creating a matrix for samples
y(shift + 1) = y_init;

for n =  (shift+2) : shift+10
y(n) = a * y(n-1) + x(n);
end
```

```
n = [-10:10];  % INdexing vector of the INPUT signal
shift = 1 - n(1);
x_n = zeros(1,length(n));
x_n(shift:shift+4) = [2 5 -1 4 -5]; % Taking as the INPUT signal
a = 1.2;  % scalar multiple
y_b = 2;  % Y Initial Value
y = diffeqn(a,x_n,y_b);
stem(n,y) % PLotting the output signal
xlabel('n');
ylabel('y[n]');
end
```

OUTPUT:



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:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

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.

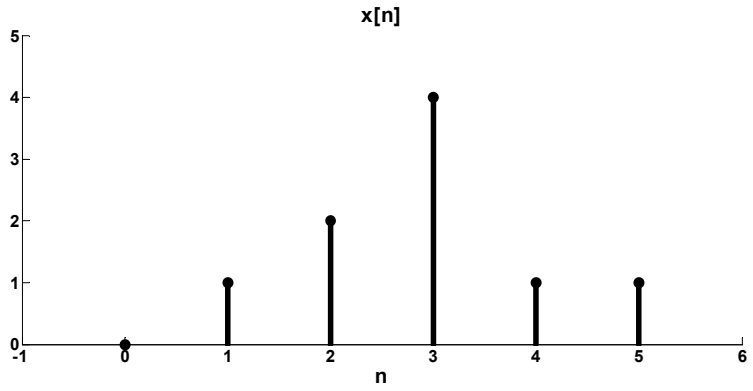
i. Write your own convolution function, ***myconv.m*** that computes the convolution between the two signals (or the output of passing an input signal through a system). Designate all the necessary inputs for your function, considering that the input signal

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).

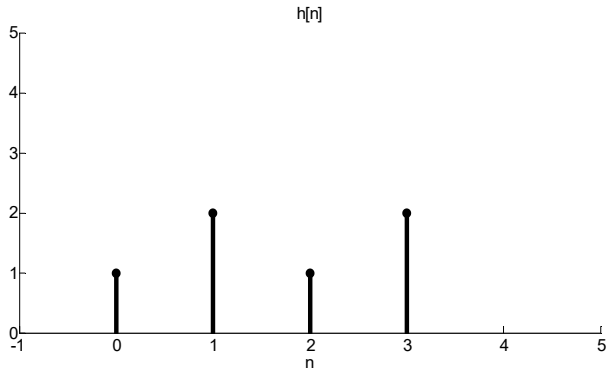
Code:

```
function [n1, y] = myconv(x, h, nx, nh)
% x is our INPUT signal
% h is the IMPULSE response
% nx is the vector index of the INPUT signal
% nh is the index vector for IMPULSE response
n1 = [nx(1) + nh(1) : nx(end) + nh(end)]; % n1 is the INDEX vector for OUTPUT
z = [];
for i=0:length(nx) - 1
   z = [z; zeros(1,i), h, zeros(1, length(nx) - i - 1)];
end
y = x * z;% output signal
x_n = [0 1 2 4 1 1]; % INPUT signal
n1 = [0:5];  % Indexing of the INPUT signal
h_n = [1 2 1 2]; % IMPULSE response
n2 = [0:3]; % Indexing for the IMPULSE response
[n_out,y_n] = myconv(x_n,h_n,n1,n2);
stem(n_out,y_n) % Plotting the OUTPUT signal
xlabel('n');
ylabel('y[n]')
end
```

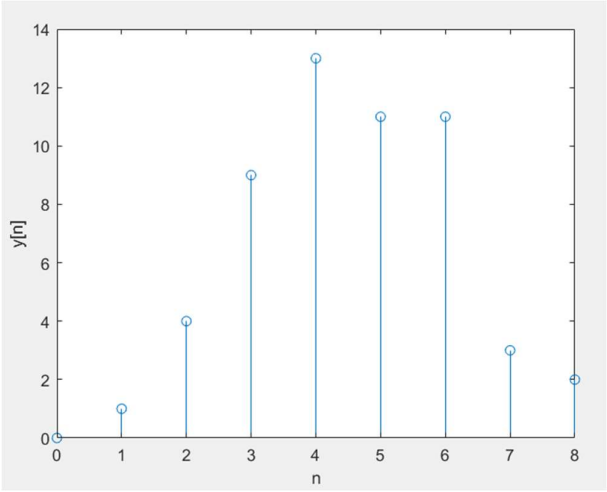
Input:



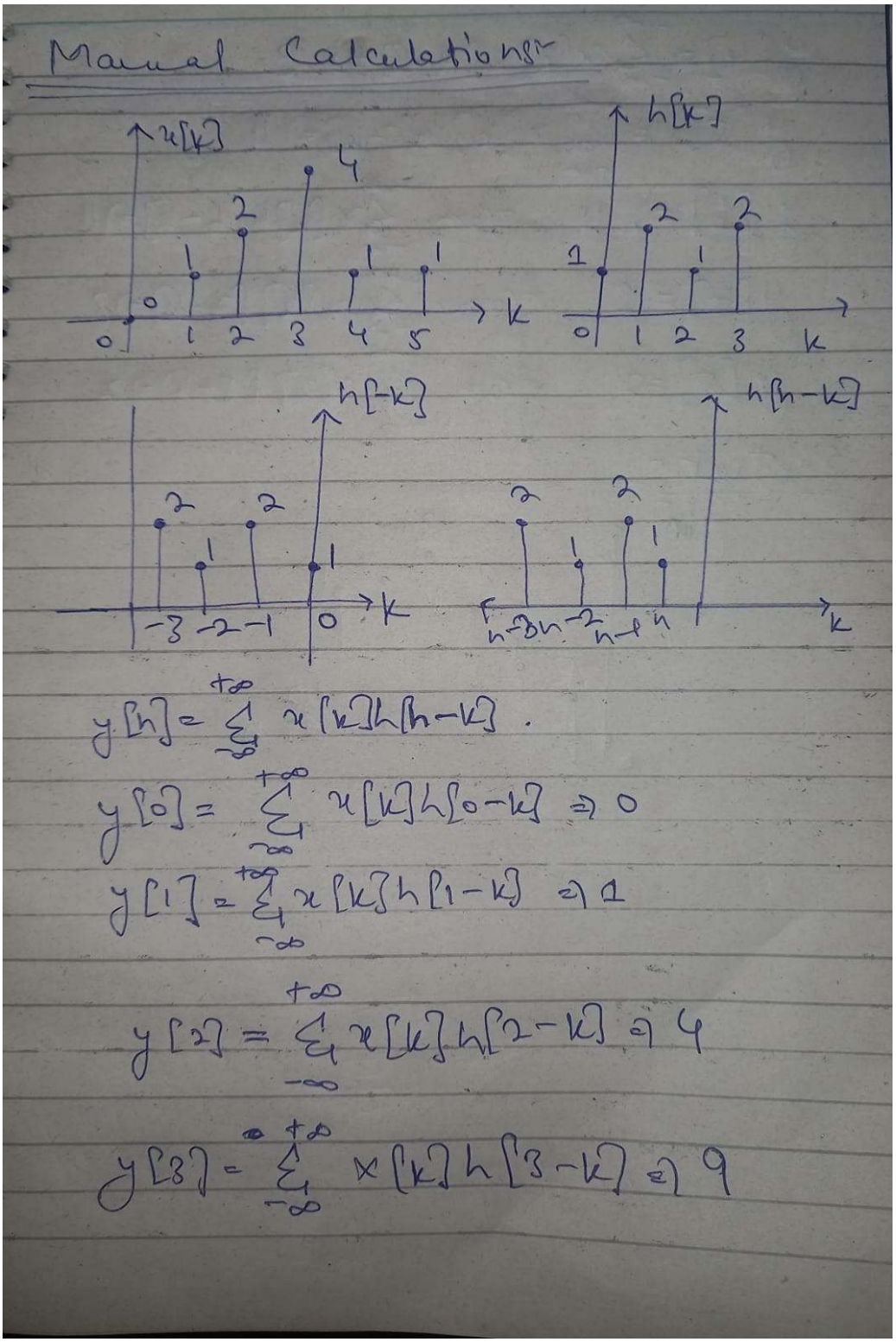
Impulse Response:



Output:



Manual Calculations:


Manual Calculations
x[k]
h[k]
h[-k]
h[n-k]
y[n] = Σ x[k]h[n-k]
y[0] = Σ x[k]h[0-k] ⇒ 0
y[1] = Σ x[k]h[1-k] ⇒ 1
y[2] = Σ x[k]h[2-k] ⇒ 4
y[3] = Σ x[k]h[3-k] ⇒ 9

$y[4] = \sum_{k=-\infty}^{+\infty} x[k]\,h[4-k] \Rightarrow 13$

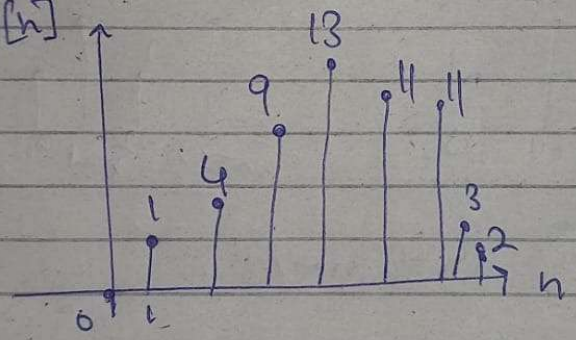
$y[5] = \sum_{k=-\infty}^{+\infty} x[k]\,h[5-k] \Rightarrow 11$

$y[6] \Rightarrow 11$ from $\sum_{k=-\infty}^{+\infty} x[k]\,h[6-k] \Rightarrow 11$

$y[7] = 3$ from $\sum_{k=-\infty}^{+\infty} x[k]\,[7-k] \Rightarrow 3$

$y[8] = 2$ from $\sum_{k=-\infty}^{+\infty} x[k]\,[8-k] \Rightarrow 2$

Output:

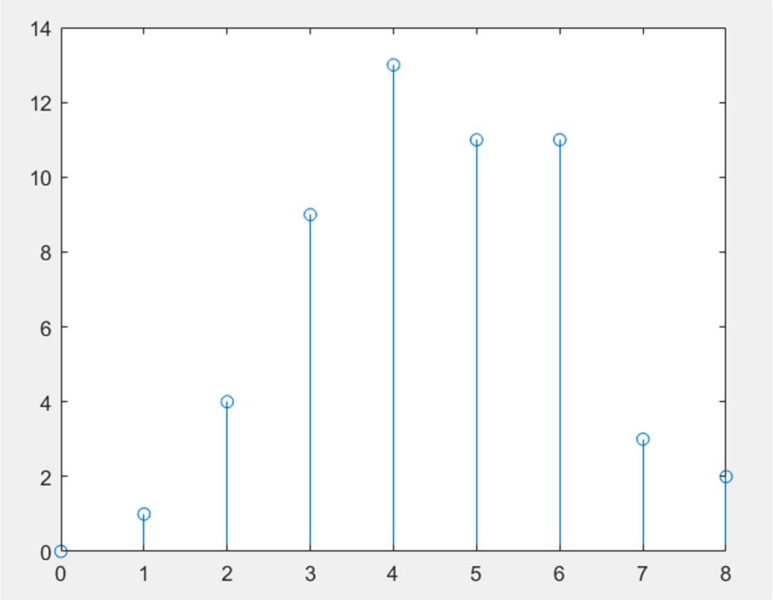


ii. MATLAB has a built-in function '***conv***' that performs the same operation. Compare the results of part (ii) with the ***conv*** function of MATLAB.

Code:

```
% INPUT
x_n = [0 1 2 4 1 1];
% INPUT INDEX VECTOR
n1 = (0:5);
% IMPLUSE RESPONSE
h_n = [1 2 1 2];
% IMPULSE RESPONSE VECTOR INDEX
n2 = [0:3];
% OUTPUT
y_n = conv(x_n,h_n);
n_out = (n1(1) + n2(1):n1(end) + n2(end));
% PLOTTING
stem(n_out,y_n)
```

OUTPUT:



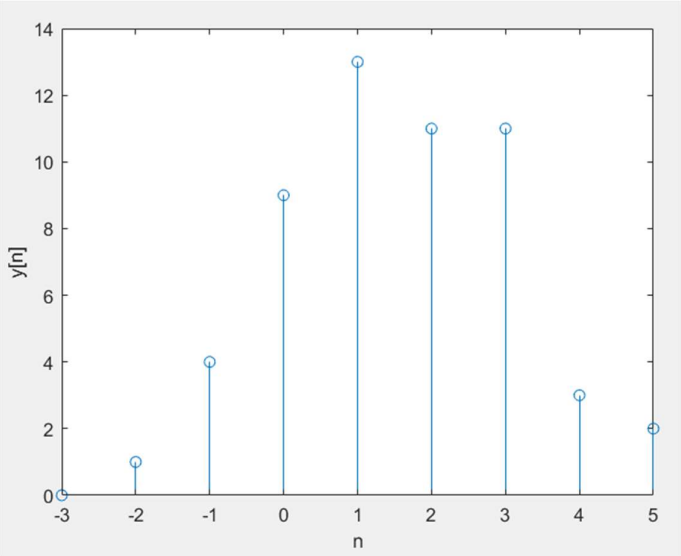
iii. Consider now that ***x[n]*** starts from ***n = -1*** and ***h[n]*** starts from ***-2***. What will be the result of convolution then? Plot the corresponding output signal using the stem command and proper timing axis.

Code:

```
% INPUT SIGNAL
x_n = [0 1 2 4 1 1];
% INPUT INDEX VECTOR
n1 = (-1:4);
% IMPULSE RESPNOSE
h_n = [1 2 1 2];
% IMPULSE INDEX VECTOR
n2 = [-2:1];
```

```
[n_out,y_n] = myconv(x_n,h_n,n1,n2);
% PLOTTING
stem(n_out,y_n)
xlabel('n');
ylabel('y[n]')
```

OUTPUT:



Conclusion

This lab comprehensively covered all the basic MATLAB concepts that were a part of the previous course. The most important functions were reviewed. This includes the difference equation, the decomposition of even and odd parts of a signal, and the convolution operation.

MATLAB certificates of all the group students are present in which this document was found in it.

MathWorks® | *Training Services*

Course Completion Certificate

Hassan Rizwan

has successfully completed **100%** of the self-paced training course

MATLAB Onramp

DIRECTOR, TRAINING SERVICES

12 February 2023

MathWorks® | *Training Services*

Course Completion Certificate

Muhammad Ahmed Mohsin

has successfully completed **100%** of the self-paced training course

MATLAB Onramp

DIRECTOR, TRAINING SERVICES

12 February 2023

MathWorks® | *Training Services*

Course Completion Certificate

Syeda Fatima Zahra

has successfully completed **100%** of the self-paced training course

MATLAB Onramp

DIRECTOR, TRAINING SERVICES

13 February 2023