Concordia University

Lab 1: Additional MATLAB features, Properties of Signals and Systems, Convolution and System Response

ELEC 342
Lab Section: UL-X

Discrete Time Signals and Systems

Penoelo Thibeaud
40212017

Lab Instructor: Razieh Abdolahi

Performed on : September 17, 2024
Submitted on : October 1, 2024

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

Objectives –

The objective of this lab was to introduce students to MATLAB functionalities and to explore the properties of discrete-time signals and systems. The lab was divided into two main parts:

Part I: Focused on understanding the use of MATLAB programming constructs such as loops, conditional statements, and array processing. This part also included verifying properties of signals and systems like linearity, even or odd properties through MATLAB.

Part II: Aimed at analyzing a system's response using convolution and exploring the system properties through mathematical and experimental approaches. The MATLAB convolution function was used to determine the output response of a system based on a given input signal, and the results were compared against theoretical expectations.

Theory -

In this lab, the theoretical foundation centers around the analysis of discrete-time signals and systems, specifically their properties and responses. The key theoretical concepts covered include:

1. **MATLAB Programming Constructs demonstrated to validate Signal properties:**
   - For loops
   - Conditional statements (if, else)
   - Array manipulation techniques
   - MATLAB's conv function

2. **Properties of Signals:**
   - Linearity: A system is considered linear if it adheres to the principle of superposition.
     - Let $x_1[n] \rightarrow y_1[n]$ and $x_2[n] \rightarrow y_2[n]$ and $x_3[n] \rightarrow y_3[n]$
     - If $x_1[n] + x_2[n] = x_3[n]$, then $y_1[n] + y_2[n] = y_3[n]$
     - If $ax_1[n]$, then $ay_1[n]$
   - Even Signals if $x[n] = x[-n]$
   - Odd Signals if $x[n] = -x[-n]$

3. **System convolution Response Analysis:**
   - Convolution operation is fundamental in determining a system's response to a given input.
     - $y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k]$

4. **Time-Invariance:**
   - A system is considered time-invariant if its behavior and characteristics do not change over time. If a time-shifted input results in a time-shifted output, then the system is time-invariant.

Tasks/Results/Discussion

Part I: MATLAB Programming Features and Signal Properties

Task for loop:

1. **Compute and display the MATLAB script from the example 1.**
   - The output display is 55, since the summation y = $\sum_{i=1}^{10} i$ = 55.

2. **Compute and display the MATLAB script from the example 2.**
   - The output display is 55, since the summation y = $\sum_{i=1}^{5} i^2$ = 55.

3. **Compute and display the MATLAB script from the example 3.**
   - The output display is 55, since the summation y = $\sum_{i=1}^{5} i^2$ = 55.

Task for if Statement:

1. **Read example 1.**
   - It displays a basic if statement and the explain the relational operators for conditional statements in MATLAB.

2. **Compute and display the MATLAB script from the example 2.**
   - It introduces the concept of user input and code flow changing based on user interaction.

3. **Compute and display the MATLAB script from the example 3.**
   - It introduces the "else" branch to control the flow when the contrition is negated.

4. **Compute and display the MATLAB script from the example 4.**
   - It introduces some good programing practice to avoid round off error.

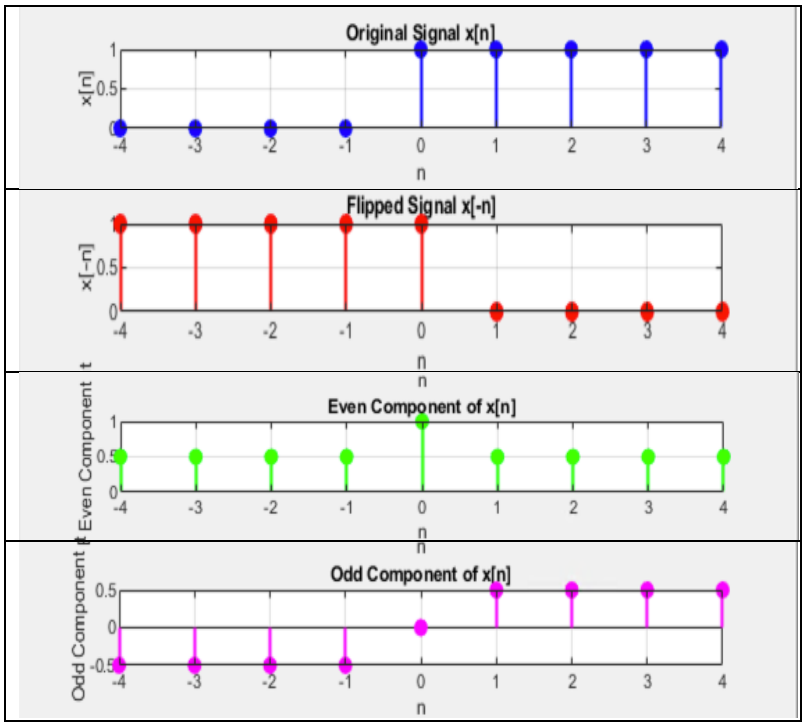
Task for Linearity:

1. **Compute and display the MATLAB script from the example 1.**
   - It confirmed the linearity of $x_1[n]$ and $x_2[n]$the example using MATLAB and display the string "Outputs are consistent with a linear system".

Task for Even-Odd signals:

1. **Compute and display the MATLAB script from the example 2.**
   - It demonstrate the notion of even or odd signal by finding the even and odd component of a signal.

Table 1Even and odd Component seperation



- The original signal $x[n]$ is in blue, The flipped signal $x[-n]$ is in red. The even component $x_e[n]$ is in green and the odd component $x_o[n]$ is in purple.
- The even part is found by using the equation $x_e[n] = \frac{x[n]+x[-n]}{2}$.
- The odd part is found by using the equation $x_o[n] = \frac{x[n]-x[-n]}{2}$.

Tips for Saving plot:

1. **Save file using print**
   - First argument specify the file type (eg. -dpsc)
   - Second argument specify the file name and extension (eg. even_odd_components.ps). The extension has to mathc the file type.

Part II: System Response and Convolution

This part of the lab demonstrate the use of convolution to analyze system responses and provide a practical means to validate theoretical system properties.Here are the main goals:

- Determine the System Response Using Direct Computation.
- Compute the System Response Using Convolution.
- Compare the Convolution-Based Response with the Direct Computation Response

Questions –

Part I: MATLAB Programming Features and Signal Properties

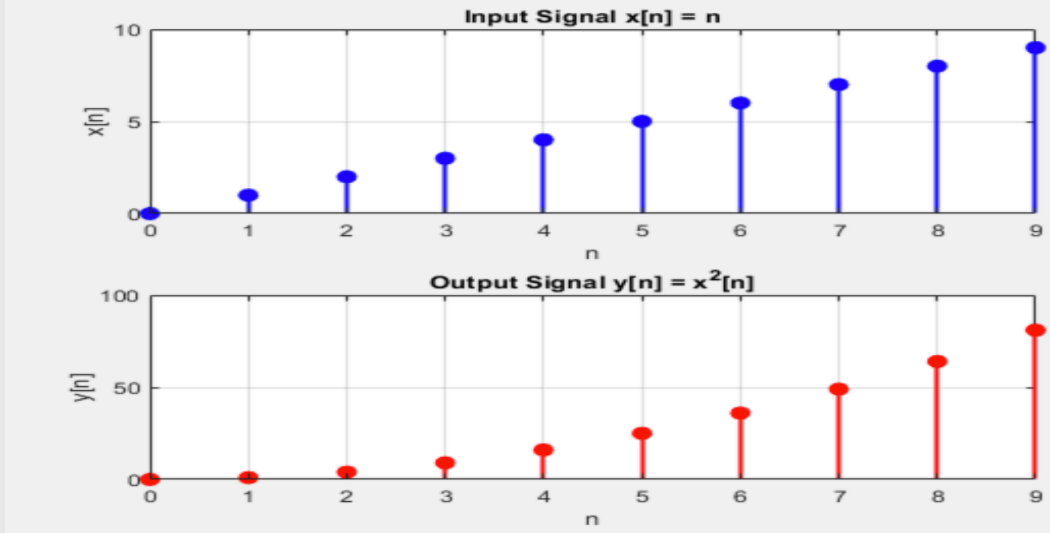
Question 1:

1. **Plot the input signal $x[n] = n$ and the output signal $y[n] = x^2[n]$ over the interval n = 0, 1,2, 3, 4, 5, 6, 7, 8, 9. Compute the total energy in the signal $x[n]$ and $y[n]$ (**Hint: the total energy in a signal is equal to the sum of the squares of all the values contained in the signal)**. Use the disp command to display the two energies.**

Table 2 Code and Output question 1-A

```
% Define the interval, input and output signal
n = 0:9;
x = n;
y = x.^2;

% Calculate the total energy of signals x[n] and y[n]
energy_x = sum(x.^2);
energy_y = sum(y.^2);

% Display the total energies
disp(['Total energy of the input signal x[n]: ', num2str(energy_x)]);
disp(['Total energy of the output signal y[n]: ', num2str(energy_y)]);

% Plot the input signal x[n]
figure;
subplot(2, 1, 1);
stem(n, x, 'filled', 'LineWidth', 1.5, 'Color', 'b');
title('Input Signal x[n] = n');
xlabel('n');
ylabel('x[n]');
grid on;

% Plot the output signal y[n]
subplot(2, 1, 2);
stem(n, y, 'filled', 'LineWidth', 1.5, 'Color', 'r');
title('Output Signal y[n] = x^2[n]');
xlabel('n');
ylabel('y[n]');
grid on;
```

```
Total energy of the input signal x[n]: 285
Total energy of the output signal y[n]: 15333
```



- The energy value is larger for the output signal as expected ,since the input value is squared twice.

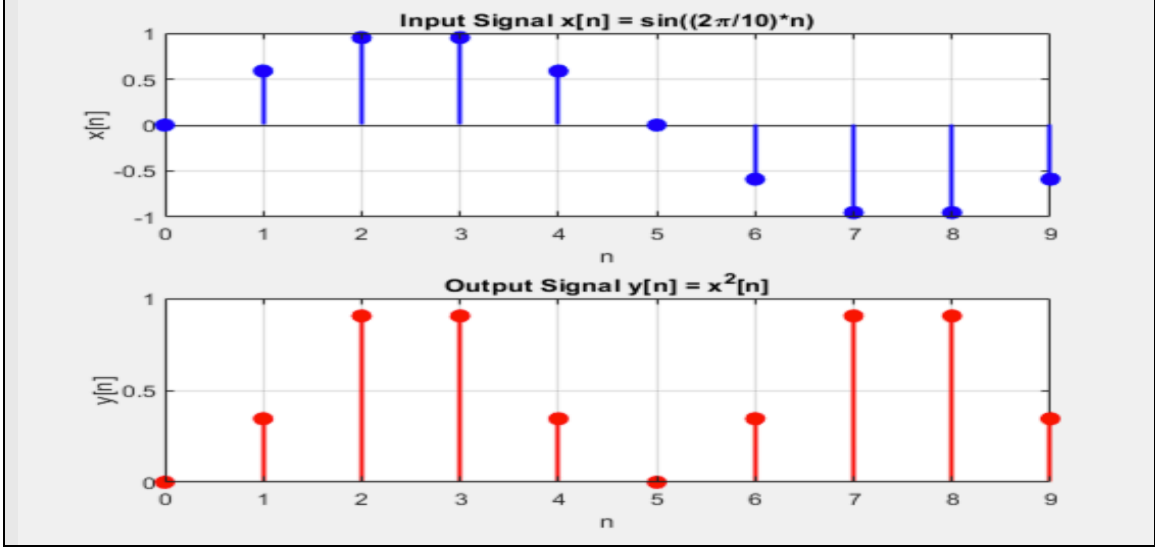
2. **Repeat part (a) using the input signal $x[n] = sin\left(\left(\frac{2\pi}{10}\right)n\right)$, n = 0, 1, 2, …, 9. Use the MATLAB function sin to compute the values of the input signal over the specified interval. Use the help sin facility to learn how to use the sin function.**

Table 3 Code and Output question 1-B

```
% Define the interval, input and output signal
n = 0:9;
x = sin((2*pi/10) * n);
y = x.^2;

% Calculate the total energy of the input signal x[n]
energy_x = sum(x.^2);
energy_y = sum(y.^2);

% Display the total energies
disp(['Total energy of the input signal x[n]: ', num2str(energy_x)]);
disp(['Total energy of the output signal y[n]: ', num2str(energy_y)]);

% Plot the input signal x[n]
```

```
figure;
subplot(2, 1, 1);
stem(n, x, 'filled', 'LineWidth', 1.5, 'Color', 'b');
title('Input Signal x[n] = sin((2\pi/10)*n)');
xlabel('n');
ylabel('x[n]');
grid on;

% Plot the output signal y[n]
subplot(2, 1, 2);
stem(n, y, 'filled', 'LineWidth', 1.5, 'Color', 'r');
title('Output Signal y[n] = x^2[n]');
xlabel('n');
ylabel('y[n]');
grid on;
```

```
Total energy of the input signal x[n]: 5
Total energy of the output signal y[n]: 3.75
```



- The sinusoid x[n] has an maximum amplitude of 1 so squaring all the value will result in a positive signal y[n] with an amplitude smaller or equal to the x[n]. This also explain why the energy of the signal y[n] is smaller then the energy of x[n].

Question 2:

A. Determine whether the discrete time system which has an output $y[n] = 2x[n]$ over the interval 0 ≤ n ≤ 10 is linear or not by determining the response $y_1[n]$ to the input signal $x_1[n] = sin\left(\left(\frac{2\pi}{10}\right)n\right)$ and the response $y_2[n]$ to the input signal $x_2[n] = cos\left(\left(\frac{2\pi}{10}\right)n\right)$.

Determine the response $y_3[n]$ to the input signal $x_3[n] = x_1[n] + x_2[n]$ and compare it with $y_4[n] = y_1[n] + y_2[n]$. Plot (using stem) **in one graph all the input signals and their corresponding output signals. Use the disp command to output whether the system has 'outputs consistent with a linear system' or 'not linear'.**

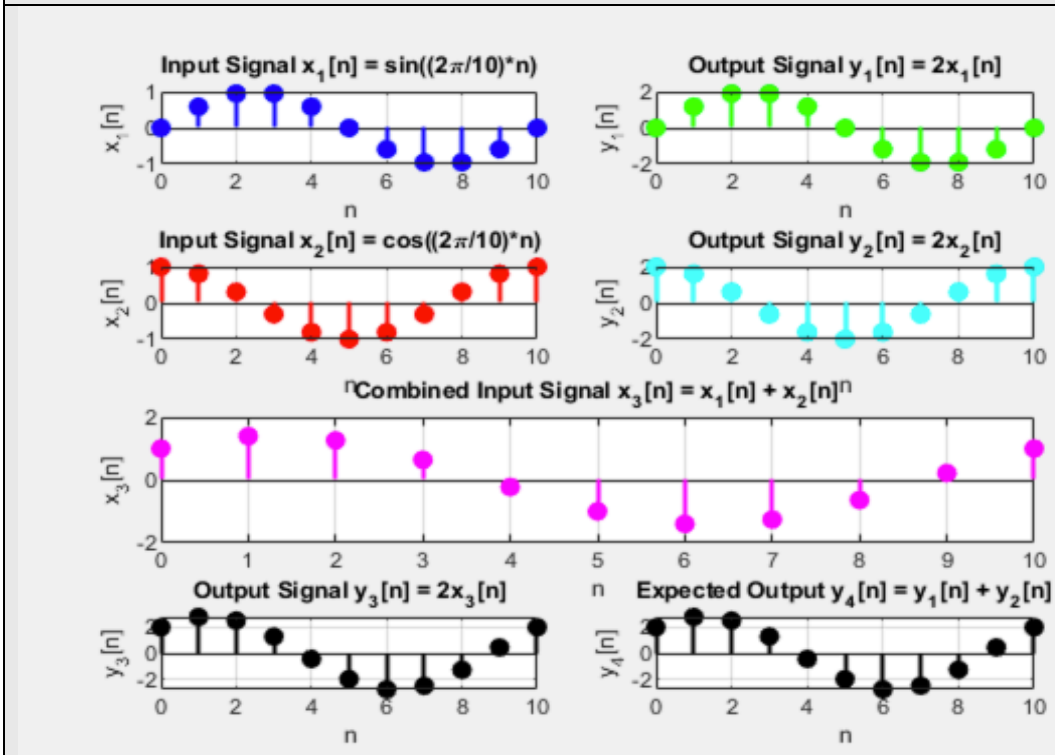
Table 4 Code and Output question 2-A

```
% Define the interval and input signals
n = 0:10;
x1 = sin((2*pi/10) * n); % Input signal x1[n]
x2 = cos((2*pi/10) * n); % Input signal x2[n]

% Compute the output responses y1[n] and y2[n]
y1 = 2 * x1;
y2 = 2 * x2;

% Define the combined input signal x3[n]
x3 = x1 + x2;

% Compute the response y3[n] and y4[n]
y3 = 2 * x3;
y4 = y1 + y2;

% Plot the input and output signals
figure;

% Plot input signals x1[n], x2[n], and x3[n]
subplot(4, 2, 1);
stem(n, x1, 'filled', 'LineWidth', 1.5, 'Color', 'b');
title('Input Signal x_1[n] = sin((2\pi/10)*n)');
xlabel('n');
ylabel('x_1[n]');
grid on;

subplot(4, 2, 3);
stem(n, x2, 'filled', 'LineWidth', 1.5, 'Color', 'r');
```

```
title('Input Signal x_2[n] = cos((2\pi/10)*n)');
xlabel('n');
ylabel('x_2[n]');
grid on;

subplot(4, 2, [5, 6]);
stem(n, x3, 'filled', 'LineWidth', 1.5, 'Color', 'm');
title('Combined Input Signal x_3[n] = x_1[n] + x_2[n]');
xlabel('n');
ylabel('x_3[n]');
grid on;

% Plot output signals y1[n], y2[n], y3[n] and y4[n]
subplot(4, 2, 2);
stem(n, y1, 'filled', 'LineWidth', 1.5, 'Color', 'g');
title('Output Signal y_1[n] = 2x_1[n]');
xlabel('n');
ylabel('y_1[n]');
grid on;

subplot(4, 2, 4);
stem(n, y2, 'filled', 'LineWidth', 1.5, 'Color', 'c');
title('Output Signal y_2[n] = 2x_2[n]');
xlabel('n');
ylabel('y_2[n]');
grid on;

subplot(4, 2, 7);
stem(n, y3, 'filled', 'LineWidth', 1.5, 'Color', 'k');
title('Output Signal y_3[n] = 2x_3[n]');
xlabel('n');
ylabel('y_3[n]');
grid on;

subplot(4, 2, 8);
stem(n, y4, 'filled', 'LineWidth', 1.5, 'Color', 'k');
title('Expected Output y_4[n] = y_1[n] + y_2[n]');
xlabel('n');
ylabel('y_4[n]');
grid on;

% Compare y3[n] with y4[n] to check linearity
if isequal(y3, y4)
   disp('The system has outputs consistent with a linear system.');
```

```
else
  disp('The system is not linear.');
end
```

```
The system has outputs consistent with a linear system.
```



- The system was confirm to be linear,because it respect the superposition principle.

B. Design an experiment to test whether the systems: (Note: *Student must answer this question in their lab report.*)

i. $y[n] = x2[n]$

ii. $y[n] = 2x[n] + 5\delta[n]$

are linear and time-invariant. Use the input data x[n] = [0,1]. Next, using a larger set of values, for your choice of the input data, repeat the experiment and analyze and interpret the results obtained with this new data set. Do the new results validate or invalidate the original results obtained with x[n] = [0,1] as choice of input data? Explain how a choice of data used may impact the results obtained.

Table 5 Code and Output question 2-B-I

```
% System 1 Analysis
x1 = [0, 1];
x2 = 2 * x1;
x3 = x1 + x2;
y1 = x1 .^ 2;
y2 = x2 .^ 2;
y3 = x3 .^ 2;
y4 = y1 + y2;

% Check for Linearity
disp("System 1 Part 1 ([0,1]): ");
if (max(abs(y3 - y4)) < 0.001)
  disp("Linear");
else
  disp("Not Linear");
end

% Check for Time Invariance
x_shifted = [0, x1(1:end-1)];
y_shifted = x_shifted .^ 2;
y_shifted_expected = [0, y1(1:end-1)];

if (max(abs(y_shifted - y_shifted_expected)) < 0.001)
  disp("Time Invariant");
else
  disp("Time Variant");
end

% System 1 Analysis for n = [0:10]
n = 0:10;
x1_n = 3*n.^2 - 6*n;
x2_n = 2 * x1_n;
x3_n = x1_n + x2_n;
y1_n = x1_n .^ 2;
y2_n = x2_n .^ 2;
y3_n = x3_n .^ 2;
y4_n = y1_n + y2_n;

% Check for Linearity with n = [0:10]
disp("System 1 Part 2 ([0,10]): ");
if (max(abs(y3_n - y4_n)) < 0.001)
```

```
  disp("Linear");
else
  disp("Not Linear");
end

% Time Invariance for n = [0:10]
x_shifted_n = [0, x1_n(1:end-1)];
y_shifted_n = x_shifted_n .^ 2;
y_shifted_n_expected = [0, y1_n(1:end-1)];

if (max(abs(y_shifted_n - y_shifted_n_expected)) < 0.001)
  disp("Time Invariant");
else
  disp("Time Variant");
end
```

```
System 1 Part 1 ([0,1]):
Not Linear
Time Invariant
System 1 Part 2 ([0,10]):
Not Linear
Time Invariant
```

- The system was confirm to be non linear, because it does not respect the superposition principle.

Table 6 Code and Output question 2-B-II

```
% Define System 2 Part 1 ([0, 1])
x1 = [0, 1];
x2 = 2 * x1;
x3 = x1 + x2;
impulse_pos = 1;
dirac1 = zeros(size(x1));
dirac1(impulse_pos) = 1;

% Define Output Signals
y1 = 2 .* x1 + 5 * dirac1;
y2 = 2 .* x2 + 5 * dirac1;
y3 = 2 .* x3 + 5 * dirac1;
y4 = y1 + y2;
```

```
% Display Results for Linearity Check (Part 1)
disp("System II Part 1 ([0,1]): ");
if max(abs(y3 - y4)) < 0.001
  disp("Linear");
else
  disp("Not Linear");
end

% Time Invariance Check (Part 1)
x_shifted = [0, x1(1:end-1)];
dirac_shifted = [0, dirac1(1:end-1)];
y_shifted = 2 .* x_shifted + 5 * dirac_shifted;
y_shifted_expected = [0, y1(1:end-1)];

if max(abs(y_shifted - y_shifted_expected)) < 0.001
  disp("Time Invariant");
else
  disp("Time Variant");
end

% Define System 2 Part 2 ([0, 10])
n = 0:10;
x1_n = 2 * n .^ 2 + 4 * n;
x2_n = 2 * x1_n;
x3_n = x1_n + x2_n;

% Define Impulse for Larger Range
dirac_n = zeros(size(n));
dirac_n(1) = 1;

% Define Output Signals for Larger Set
y1_n = 2 .* x1_n + 5 * dirac_n;
y2_n = 2 .* x2_n + 5 * dirac_n;
y3_n = 2 .* x3_n + 5 * dirac_n;
y4_n = y1_n + y2_n;

% Display Results for Linearity Check (Part 2)
disp("System II Part 2 ([0,10]): ");
if max(abs(y3_n - y4_n)) < 0.001
  disp("Linear");
else
  disp("Not Linear");
end
```

```
% Time Invariance Check (Part 2)
x_shifted_n = [0, x1_n(1:end-1)];
dirac_shifted_n = [0, dirac_n(1:end-1)];
y_shifted_n = 2 .* x_shifted_n + 5 * dirac_shifted_n;
y_shifted_n_expected = [0, y1_n(1:end-1)];

if max(abs(y_shifted_n - y_shifted_n_expected)) < 0.001
  disp("Time Invariant");
else
  disp("Time Variant");
end
```

```
System II Part 1 ([0,1]):
Not Linear
Time Invariant
System II Part 2 ([0,10]):
Not Linear
Time Invariant
```

- In both cases the results validate the original results. This confirm that the systems are not linear and not time invariant regardless of the input choice.

Question 3:

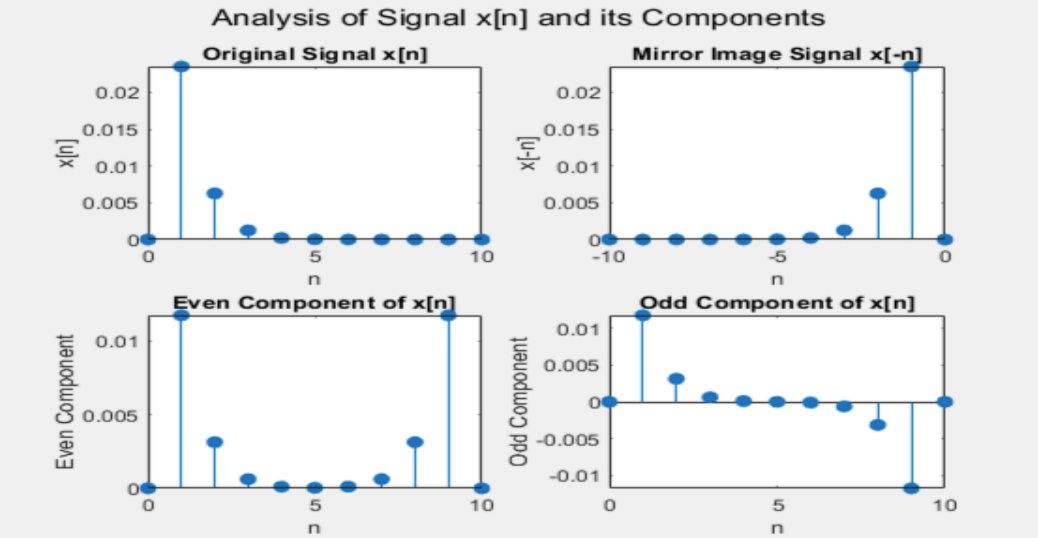
A. Plot the following signal $x[n]$, its mirror image $x[-n]$, and it's even and odd components:

i. $x[n] = e^{-2|n|} sin\left(\left(\frac{2\pi}{36}\right)n\right)$, **0 ≤ n ≤ 10**

ii. **Use the MATLAB functions exp and abs .**

```
% Define the range of n
n = 0:10;

% Compute the given signal x[n]
x_n = exp(-2 * abs(n)) .* sin((2 * pi / 36) * n);

% Define the mirror image x[-n]
n_mirror = -fliplr(n);   % Create a flipped version of n to represent -n
x_mirror = fliplr(x_n);  % Mirror image of x[n] is simply the flipped x[n]

% Calculate even and odd components of the signal
x_even = 0.5 * (x_n + x_mirror); % Even component
x_odd = 0.5 * (x_n - x_mirror);  % Odd component

% Plot the original signal x[n]
subplot(2, 2, 1);
stem(n, x_n, 'filled');
title('Original Signal x[n]');
xlabel('n');
ylabel('x[n]');

% Plot the mirror image x[-n]
subplot(2, 2, 2);
stem(n_mirror, x_mirror, 'filled');
title('Mirror Image Signal x[-n]');
xlabel('n');
ylabel('x[-n]');

% Plot the even component
subplot(2, 2, 3);
stem(n, x_even, 'filled');
title('Even Component of x[n]');
xlabel('n');
ylabel('Even Component');

% Plot the odd component
subplot(2, 2, 4);
```

```
stem(n, x_odd, 'filled');
title('Odd Component of x[n]');
xlabel('n');
ylabel('Odd Component');

% Display the plots
sgtitle('Analysis of Signal x[n] and its Components');
```

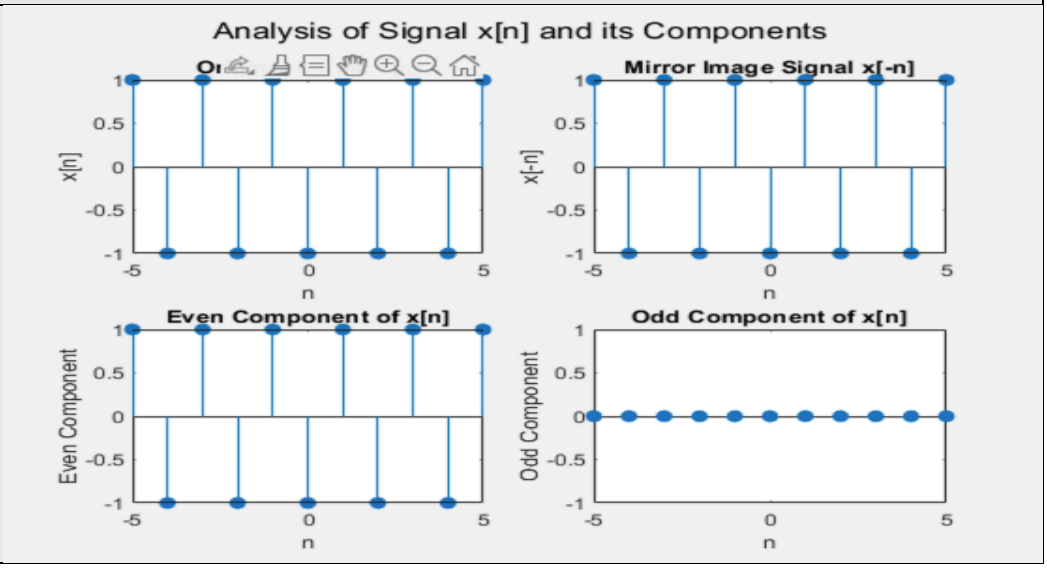


- The even and odd component were recreated.

B. Repeat part (a) for the signal: $x[n] = (-1)n-1$, $-5 \leq n \leq 5$

```
% Define the range of n
n = -5:5;

% Compute the given signal x[n]
x_n = (-1).^(n - 1);

% Define the mirror image x[-n]
n_mirror = -fliplr(n);    % Create a flipped version of n to represent -n
x_mirror = fliplr(x_n);   % Mirror image of x[n] is simply the flipped x[n]

% Calculate even and odd components of the signal
x_even = 0.5 * (x_n + x_mirror); % Even component
x_odd = 0.5 * (x_n - x_mirror);  % Odd component

% Plot the original signal x[n]
subplot(2, 2, 1);
```

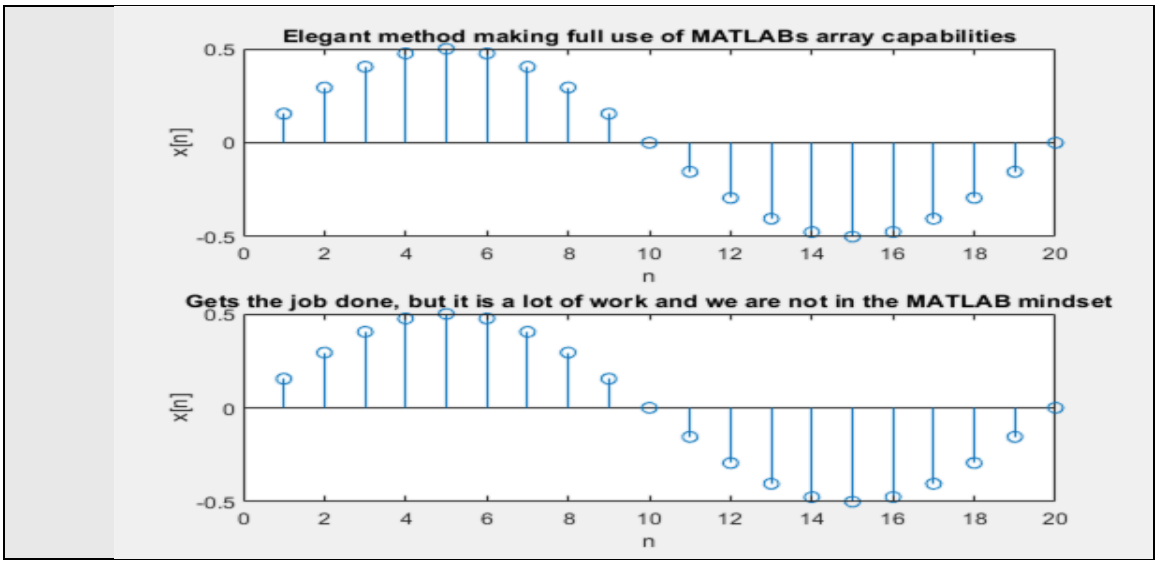
```
stem(n, x_n, 'filled');
title('Original Signal x[n]');
xlabel('n');
ylabel('x[n]');

% Plot the mirror image x[-n]
subplot(2, 2, 2);
stem(n_mirror, x_mirror, 'filled');
title('Mirror Image Signal x[-n]');
xlabel('n');
ylabel('x[-n]');

% Plot the even component
subplot(2, 2, 3);
stem(n, x_even, 'filled');
title('Even Component of x[n]');
xlabel('n');
ylabel('Even Component');

% Plot the odd component
subplot(2, 2, 4);
stem(n, x_odd, 'filled');
title('Odd Component of x[n]');
xlabel('n');
ylabel('Odd Component');

% Display the plots
sgtitle('Analysis of Signal x[n] and its Components');
```



- This signal was an even signal since the decomposition show that the odd component is zero for all the values.

C. Compare and contrast the two methods used to generate the two MATLAB arrays x1 and x2 in the following MATLAB code:

```
% T. Obuchowicz
%Fri Apr 27 16:03:27 EDT 2012

clear
n = [1 : 20 ]

x1 = sin((2*pi/40) * n) .* cos((2*pi/40) * n)

for index = 1 : 20

% Note: In MATLAB, no need to pre-allocate the array,
% unlike C++ and other high-level programming languages.

   x2(index) = sin((2*pi/40) * index) * cos((2*pi/40) * index)

end

subplot(2,1,1)
stem(n, x1)
title('Elegant method making full use of MATLABs array capabilities')
xlabel('n')
ylabel('x[n]')

subplot(2,1,2)
stem(n, x2)
title('Gets the job done, but it is a lot of work and we are not in the MATLAB
mindset')
xlabel('n')
ylabel('x[n]')
```



- They will both produce the same outputs, but there is a difference in the way they handle data. The first method utilize the built in MATLAB functionality and can be written in a single line. The second method give more transparency on what is happening, but is also less efficient.

Part II: System Response and Convolution

Question 1:

For this system, compute the response $y[n]$ over the interval $0 \leq n \leq 9$ using the input signal $x[n]$ given in Figure 4.

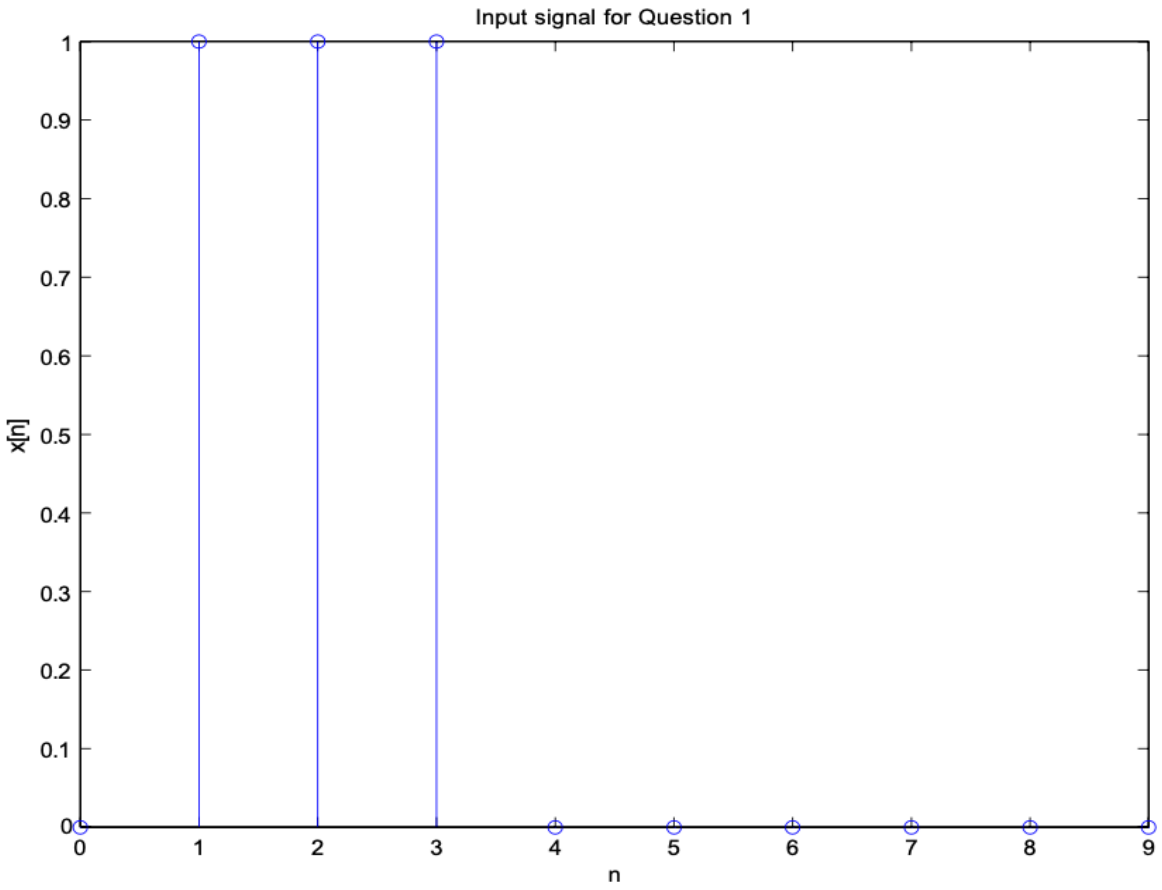
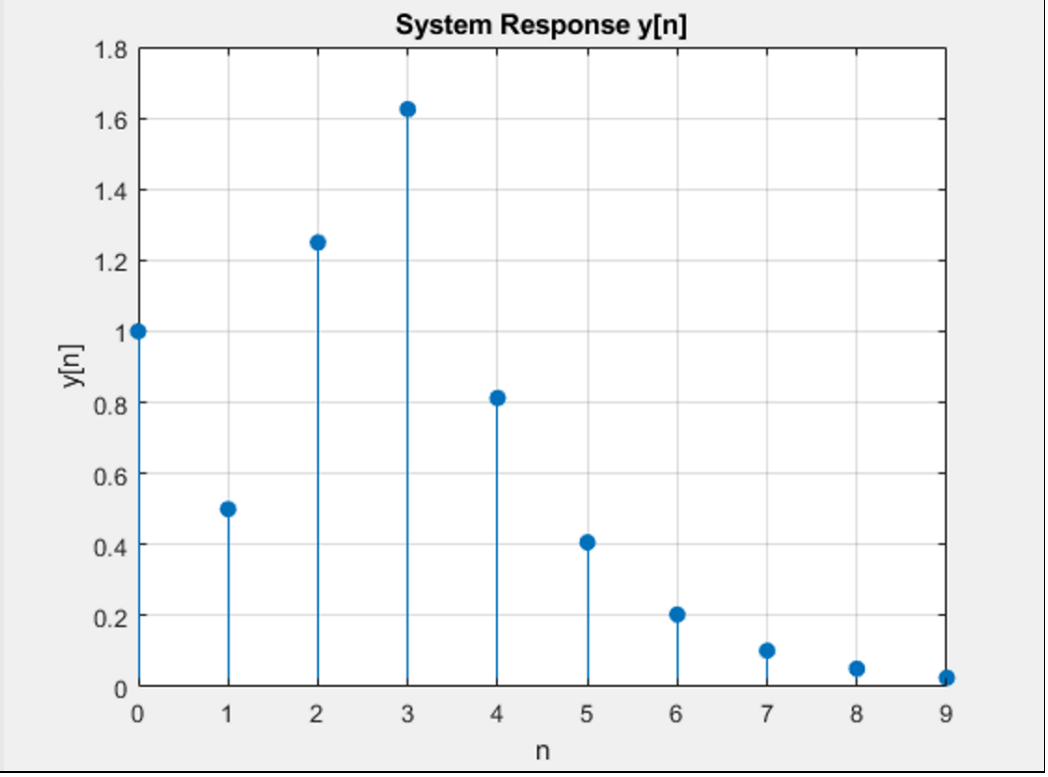


Figure 4: Input signal to be used to determine system response.

Plot the response using the stem function.

```
% Define the input signal x[n]
n = 0:9;
x = zeros(1, 10);
x(1) = 1;
x(3) = 1;
x(4) = 1;

% Define the impulse response h[n] for the assumed system
```

```
h = (0.5) .^ (0:9);

% Compute the system response using convolution
y = conv(x, h);

% Truncate y[n] to the interval
y = y(1:10);

% Plot the response using stem
figure;
stem(n, y, 'filled');
title('System Response y[n]');
xlabel('n');
ylabel('y[n]');
grid on;
```



Question 2:

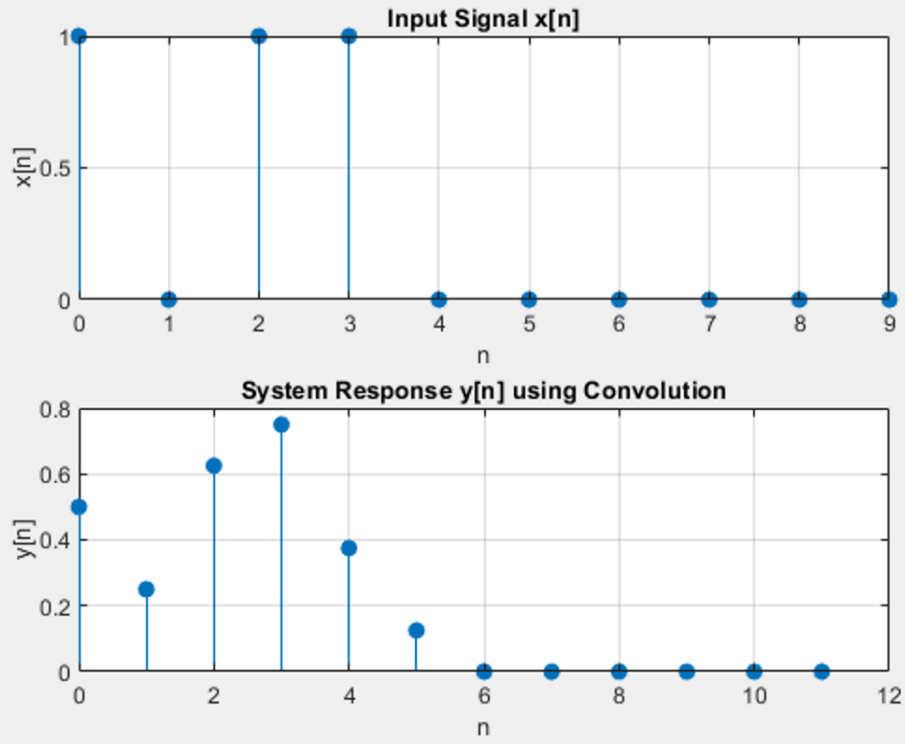
MATLAB contains a built-in function called conv which performs the convolution of two vectors:

```
>> help conv
  conv Convolution and polynomial multiplication.
       C = conv(A, B) convolves vectors A and B. The resulting vec-
tor is
     length MAX([LENGTH(A)+LENGTH(B)-1,LENGTH(A),LENGTH(B)]). If A
and B are
vectors of polynomial coefficients, convolving them is equivalent
to multiplying the two polynomials.
```

Compute the system response (using the input signal $x[n]$ given in Question (1)) by convolving $H[n]$ and $x[n]$ using the MATLAB conv function. Plot the response using stem. Compare this response with that obtained in Question 1. Explain any differences between the two outputs.

```
% Define the input signal x[n]
n = 0:9;
x = zeros(1, 10);
x(1) = 1;
x(3) = 1;
x(4) = 1;

% Define the impulse response h[n]
h = [0.5, 0.25, 0.125];

% Compute the system response using convolution
y = conv(x, h);

% Truncate the output to the interval 0 <= n <= 9
y_truncated = y(1:10);

% Plot the original input signal x[n]
figure;
subplot(2, 1, 1);
stem(n, x, 'filled');
title('Input Signal x[n]');
xlabel('n');
ylabel('x[n]');
grid on;
```

```
% Plot the truncated system response y[n]
subplot(2, 1, 2);
stem(n, y_truncated, 'filled');
title('System Response y[n] using Convolution');
xlabel('n');
ylabel('y[n]');
grid on;
```



- The impulse response $h[n]$ is set correctly, the convolution result matched the plot from Question 1.

Question 3:

(Note: *Student must answer all Question 3 in their lab report.*)

Sometimes a system that we are interested in is a black box, i.e., we can see what goes into the system and what comes out of the system, but we cannot see what is inside the system.
In such a case we can try to find out some things about the system by choosing inputs to put in and then observing what comes out.
We could also use this same process of putting inputs in and observing the output to determine the system entirely. When we do this, it is called system identification.
Here you are given a system called Sys1. Note, Sys1.p is MATLAB P-code file, available at this link.

y = Sys1(x)

Inputs to the function is the input vector x. The vector y is a vector which is the corresponding output signal. Note that y may not be the same length as x.

A. **Is the system Linear?**
   - **To show that a system is linear we must show it in general. To show it is not linear we must provide a counter example. Here we will give inputs to the system and get the outputs. We can do this as many times as we like. Write an hypothesis that you will design an experiment to test. Comment on the possible outcomes of your experiment.**
   - **Design an experiment to test linearity. Discuss why you chose particular inputs to put into the system and what you are expecting to learn by observing the outputs. This answer should be 0.5 to 1 page long.**
   - **Describe what you observed when you put your inputs into the system.**
   - **What did you conclude from your experiments? Was the system linear or not? Can you state this conclusion with certainty?**

```
% Define test input signals
x1 = [1, 2, 3, 4, 5];
x2 = [5, 4, 3, 2, 1];

% Define a scalar constant
a = 2;

% Compute the system response for each input using Sys1
y1 = Sys1(x1);
y2 = Sys1(x2);

% Check Superposition Property
```

```
x_sum = x1 + x2;
y_sum = Sys1(x_sum);
y_sum_expected = y1 + y2;

% Display results for Superposition
disp('Checking Superposition Property:');
disp(['y_sum (Sys1(x1 + x2)) = ', num2str(y_sum)]);
disp(['y_sum_expected (y1 + y2) = ', num2str(y_sum_expected)]);

if isequal(y_sum, y_sum_expected)
  disp('The system satisfies the Superposition Property .');
else
  disp('The system does NOT satisfy the Superposition Property .');
end

% Check Homogeneity Property
x_scaled = a * x1;
y_scaled = Sys1(x_scaled);
y1_scaled_expected = a * y1;

% Display results for Homogeneity
disp('Checking Homogeneity Property:');
disp(['y_scaled (Sys1(a * x1)) = ', num2str(y_scaled)]);
disp(['y1_scaled_expected (a * y1) = ', num2str(y1_scaled_expected)]);

if isequal(y_scaled, y1_scaled_expected)
  disp('The system satisfies the Homogeneity Property .');
else
  disp('The system does NOT satisfy the Homogeneity Property .');
end
```

```
Checking Superposition Property:
y_sum (Sys1(x1 + x2)) = 36  48  48  48  48  12   0   0   0   0   0   0   0   0   0
y_sum_expected (y1 + y2) = 26  32  30  32  38  12   0   0   0   0   0   0   0   0   0
The system does NOT satisfy the Superposition Property .
Checking Homogeneity Property:
y_scaled (Sys1(a * x1)) = 4   20   44   76  116   20    0    0    0    0    0    0    0    0    0
y1_scaled_expected (a * y1) = 2  12  26  44  66  20   0   0   0   0   0   0   0   0   0
The system does NOT satisfy the Homogeneity Property .
```

- Both properties (superposition and homogeneity) are not satified, so the sys1.p is not linear.

B. Is the system Time Invariant?

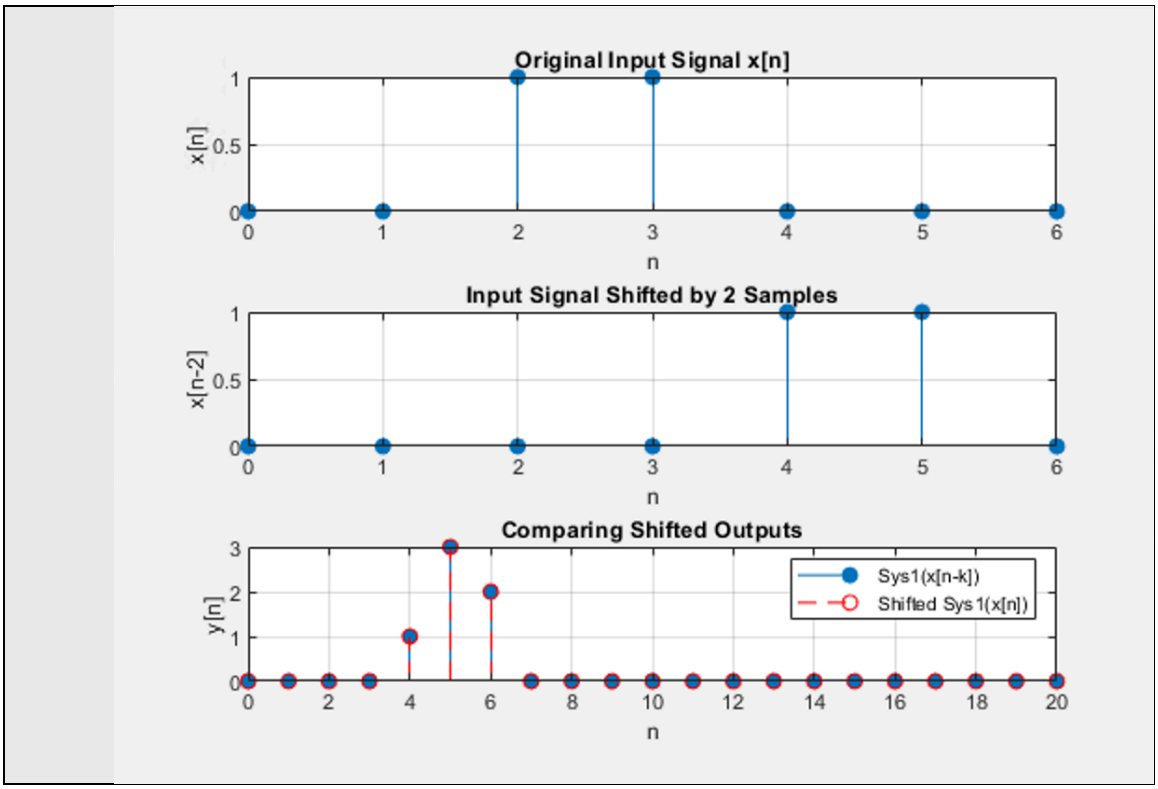
- **To show that a system is time invariant we must show it in general. To show it is not time invariant or time varying we must provide a counter example. Here we will give inputs to the system and get the outputs. We can do this as many times as we like. Write an hypothesis that you will design an experiment to test. Comment on the possible outcomes of your experiment.**
- **Design an experiment to test time invariance. Discuss why you chose particular inputs to put into the system and what you are expecting to learn by observing the outputs. This answer should be 0.5 to 1 page long.**
- **Describe what you observed when you put your inputs into the system.**
- **What did you conclude from your experiments? Was the system time invariant or not? Can you state this conclusion with certainty?**

```
% Define the original input signal x[n]
x = [0, 0, 1, 1, 0, 0, 0];
n = 0:length(x)-1;

% Define the time shift value
k = 2;

% Compute the original system response
y = Sys1(x);

% Shift the input signal by k (left shift)
x_shifted = [zeros(1, k), x(1:end-k)];

% Compute the system response to the shifted input signal
y_shifted_input = Sys1(x_shifted);

% Shift the original system response by k (left shift)
y_shifted_output = [zeros(1, k), y(1:end-k)];

% Display results
disp('Original Input Signal x[n]:');
disp(x);

disp(['Input Signal Shifted by ', num2str(k), ' samples:']);
disp(x_shifted);

disp('Original System Response y[n]:');
disp(y);

disp(['System Response to Shifted Input y_shifted_input:']);
```

```
disp(y_shifted_input);

disp(['Original System Response Shifted by ', num2str(k), ' samples:']);
disp(y_shifted_output);

% Compare the results to check for time invariance
if isequal(y_shifted_input, y_shifted_output)
   disp('The system is Time Invariant.');
else
   disp('The system is Time Variant.');
end

% Plot the results for visual confirmation
figure;
subplot(3,1,1);
stem(n, x, 'filled');
title('Original Input Signal x[n]');
xlabel('n');
ylabel('x[n]');
grid on;

subplot(3,1,2);
stem(n, x_shifted, 'filled');
title(['Input Signal Shifted by ', num2str(k), ' Samples']);
xlabel('n');
ylabel(['x[n-', num2str(k), ']']);
grid on;

subplot(3,1,3);
stem(0:length(y_shifted_input)-1, y_shifted_input, 'filled');
hold on;
stem(0:length(y_shifted_output)-1, y_shifted_output, 'r--');
title('Comparing Shifted Outputs');
xlabel('n');
ylabel('y[n]');
legend('Sys1(x[n-k])', 'Shifted Sys1(x[n])');
grid on;
```

```
Original Input Signal x[n]:
   0    0    1    1    0    0    0

Input Signal Shifted by 2 samples:
```

```
    0   0   0   0   1   1   0

Original System Response y[n]:
 Columns 1 through 16

    0   0   1   3   2   0   0   0   0   0   0   0   0   0   0   0

 Columns 17 through 21

    0   0   0   0   0

System Response to Shifted Input y_shifted_input:
 Columns 1 through 16

    0   0   0   0   1   3   2   0   0   0   0   0   0   0   0   0

 Columns 17 through 21

    0   0   0   0   0

Original System Response Shifted by 2 samples:
 Columns 1 through 16

    0   0   0   0   1   3   2   0   0   0   0   0   0   0   0   0

 Columns 17 through 21

    0   0   0   0   0

The system is Time Invariant.
```



- The systems are time invariant , since the behavior of the system remains consistent even with a shift.

Conclusions –

This lab explored various MATLAB functionalities and applied them to analyze discrete-time signals and systems, focusing on properties such as linearity, even-odd symmetry and time invariance. The experiment was divided into two main parts, each addressing distinct aspects of signal and system analysis. Part I involved using MATLAB programming constructs such as loops, conditionals and array manipulation to verify signal properties. The results confirmed the theoretical expectations, demonstrating that MATLAB is a powerful tool for validating linearity and symmetry properties in signals. Part II emphasized system response analysis using both direct computation and convolution methods. The experiment verified that convolution accurately models the response of linear time-invariant (LTI) systems and is equivalent to solving the system's governing difference equations. This section also included experiments to validate the time invariance and linearity properties of different systems.