

PRACTICAL WORK BOOK
For The Course
EE-232 Signals and Systems



For
B.E. Electrical Engineering

Group Members			
Azlaan Ranjha			
Laiba Jabbar			
Degree	DE-43	Syndicate	B

Complied By:

Checked By:

DEPARTMENT OF ELECTRICAL ENGINEERING
College of Electrical & Mechanical Engineering (CEME), NUST-Pakistan

LIST OF EXPERIMENTS

S.NO.	TITLE OF EXPERIMENT
01	Basic MATLAB Commands
02	Plotting and Loops
03	Basic Signal Properties
04	Symbolic Variables and Equations
05	Introduction to Convolution
06	Introduction to Simulink
07	Recording audio signal using Audiorecorder, and Convolution in Simulink
08	Building Graphical User Interface
09	Continuous Time Fourier Series and its Properties
10	Discrete Time Fourier Series and its Properties.
11	Discrete Time Fourier Transform and its Properties
12	Continuous time fourier transform and its properties
13	Implementing Filters, Recording Audio and passing the audio through filters
14	Transfer Functions and their Step and Impulse Response

Experiment No. 1

Objective:

The objective of this lab is to create an understanding of the basic MATLAB commands and familiarize the students with MATLAB environment.

Theoretical Background:

MATLAB stands for **matrix laboratory**. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages including C, C++, Java, etc. The MATLAB application is built around the MATLAB scripting language. Common usage of the MATLAB application involves using the Command Window as an interactive mathematical shell or executing text files containing MATLAB code. This lab deals with an introduction to MATLAB, where students will learn different ways of creating vectors, and to perform various operations on vectors.

Tasks:

The following tasks are to be performed by the students.

Task 1:

The students will learn the different ways of creating vectors in MATLAB in this task.

(a) Generate the following vectors:

$$A = [1 \ 0 \ 4 \ 5 \ 3 \ 9 \ 0 \ 2]$$

$$a = [4 \ 5 \ 0 \ 2 \ 0 \ 0 \ 7 \ 1]$$

Be aware that MATLAB is case sensitive. Vector A and a have different values.

(b) Generate the following vectors:

$$B = [A \ a]$$

$$C = [a, A]$$

Concatenation is the process of joining small matrices to make bigger ones. In fact, you made your first matrix by concatenating its individual elements. The pair of square brackets, [], is the concatenation operator.

(c) Generate the following vectors using function zeros and ones:

$$D = [0 \ 0 \ 0 \dots \ 0] \text{ with fifty 0's.}$$

$$E = [1 \ 1 \ 1 \dots \ 1] \text{ with a hundred 1's.}$$

(d) Generate the following vectors using the colon operator

$$F = [1 \ 2 \ 3 \ 4 \dots \ 30]$$

$G = [25 \ 22 \ 19 \ 16 \ 13 \ 10 \ 7 \ 4 \ 1]$

$H = [0 \ 0.2 \ 0.4 \ 0.6 \dots \ 2.0]$

The colon “:” is one of MATLAB’s most important operators.

Task 2:

Operate with the following vectors to perform tasks (a) to (e):

$V1 = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0]$

$V2 = [0.3 \ 1.2 \ 0.5 \ 2.1 \ 0.1 \ 0.4 \ 3.6 \ 4.2 \ 1.7 \ 0.9]$

$V3 = [4 \ 4 \ 4 \ 4 \ 3 \ 3 \ 2 \ 2 \ 2 \ 1]$

(a) Calculate, respectively, the sum of all the elements in vectors V1, V2, and V3.

(b) How to get the value of the fifth element of each vector? What happens if we execute the command $V1(0)$ and $V1(11)$? Remember if a vector has N elements, their subscripts are from 1 to N.

(c) Generate a new vector V4 from V2, which is composed of the first five elements of V2. Generate a new vector V5 from V2, which is composed of the last five elements of V2.

(d) Derive a new vector V6 from V2, with its 6th element omitted. Derive a new vector V7 from V2, with its 7th element changed to 1.4, and a vector V8 from V2, whose elements are the 1st, 3rd, 5th, 7th, and 9th elements of V2.

(e) What are the results of the following?

- $9-V1$
- $V1*5$
- $V1+V2$
- $V1-V3$
- $V1.*V2$
- $V1*V2$
- $V1.^2$
- $V1.^V3$
- $V1^V3$
- $V1 == V3$
- $V1>6$
- $V1>V3$
- $V3-(V1>2)$
- $(V1>2) \& (V1<6)$
- $(V1>2) | (V1<6)$

- any(V1)
- all(V1)

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

Task 1 Description:

We are creating two vectors in part a of Task 1, '**A**' and '**a**'. Then in part b, we create a vector by the name of '**B**' in which we horizontally concatenate **A** with **a** which means that if they have same rows then elements of **a** are added at the end of elements of **A**. Then in part c, we create a 1x50 vector named **D** that has its all elements as zero. While we also created **E** that is a 1x50 vector with all elements as one. Lastly, we have a vector **F** with elements from 1 to 30, it is a row vector. Vector **G** has elements from 25

to 1 with a spacing of -3 between each element. While Vector **H** has elements from 0 to 2 with a spacing of 0.2 between each element.

Task 2:

Task 2:
Variable Creation:

```
42 V1 = [1:9,0];
43 V2 = [0.3 1.2 0.5 2.1 0.1 0.4 3.6 4.2 1.7 0.9];
44 V3 = [4 4 4 4 3 3 2 2 2 1];
```

Part a:

```
45 Addition = V1 + V2 + V3
```

Part b:

```
46 V1fifth = V1(1,5)
47 V2fifth = V2(1,5)
48 V3fifth = V3(1,5)
49 %V1(0)
50 %V1(11)
```

Part c:

```
51 V4 = V2(1,1:5)
52 V5 = V2(1,6:end)
```

Part d:

```
53 V20 = V2
54 V20(6) = []
55 V6 = V20
56 V7 = V2
57 V7(1,7) = 1.4
58 V8 = V2([1 3 5 7 9])
```

Part e:
All elements are subtracted individually from 9 to create a new matrix.

```
59 9 - V1
```

Each element is multiplied by 5.

```
60 V1*5
```

Vector V1 is being added to Vector V2.

```
61 V1+V2
```

Vector V3 is subtracted from vector V1.

```
62 V1-V3
```

Element-wise multiplication of vector V1 and vector V2.

```
63 V1.*V2
```

Due to dimension incompatibilities, matrix multiplication cant take place so we commented it out.

```
64 %V1.*V2
```

Each element is raised to the power of two.

```
65 V1.^2
```

Element-wise the elements of vector V1 are raised to the power of their corresponding elements in V3.

```
66 V1.^V3
```

	A matrix power cannot be another matrix.
67	<code>%V1^V3</code>
	This operation returns a vector with all elements which are equal to their corresponding element to return a 1 and those that are not to return a 0.
68	<code>V1 == V3</code>
	This element checks all elements in V1 and returns 1 for all those greater than 6. Otherwise it is a 0.
69	<code>V1>6</code>
	This element checks all elements in V1 and returns 1 for all those greater than their corresponding element in V3. Otherwise it is a 0.
70	<code>V1>V3</code>
	First, a vector is created with the answers of the operation (V1>2) and then they are subtracted from the vector V3.
71	<code>V3-(V1>2)</code>
	It checks the elements of V1 so that they are greater than 2 but less than 6.
72	<code>(V1>2) & (V1<6)</code>
	It checks the elements of V1 so that they are either greater than 2 or less than 6.
73	<code>(V1>2) (V1<6)</code>
	'any' function checks whether there is even a single non-zero element in the vector or matrix. If yes, then its returns a single 1. Otherwise 0.
74	<code>any(V1)</code>
	'all' function checks whether all elements in a vector or matrix are non-zero. If yes, then its returns a single 1. Otherwise 0.
75	<code>all(V1)</code>

Task 2 Description:

We have created three vectors **V1**, **V2** and **V3**. **V1** has elements from 1 to 9 with a zero at the end. **V2** has decimal vectors and **V3** has different combinations of 4,3,2 and 1. All of these vectors are 1x10 in size. In part a, we add all three vectors and store the result in a variable '**Addition**'. In part b, we extract the fifth element of all three vectors. In part c, the first five elements of **V2** are stored in **V4** and last five are stored in **V5**. In part d, we store elements from **V2** into variable **V20** and then remove its sixth element. **V7** has values of **V2** stored which then have their 7th element replaced by 1.4. Meanwhile **V8** has all the odd elements of **V2** stored in it. In last part e, some basic arithmetic and logic operations and carried out and we make use of the **any** and **all** functions.

Conclusion:

In this lab, we learned MATLAB basics, a powerful tool for data analysis and numerical tasks. We covered vector creation techniques, including manual input and sequence generation with the colon operator. We also explored vector operations like summing, element access, and extraction. These skills are essential for data manipulation in MATLAB, which is valuable for various applications, including data analysis and engineering tasks

Verified: _____

Date/Time: _____

Experiment No. 2

Objective:

The objective of this lab is to create an understanding of using loops and conditional statements in MATLAB. It also introduces the basics of plotting signals.

Theoretical Background:

As in any language, MATLAB also has loops and conditional statements. The ‘for’ and ‘while’ loops can be used to run a specific set of commands a number of times based on some conditions. Similarly, the ‘if-else’ conditional statements are used to run a specific set of commands only if a specified condition is true, and another set of commands may be run if the condition is not true.

A very important feature of MATLAB is the plotting of signals. It is exceptionally important in reference to the Signals and Systems course. There are many different commands in MATLAB that can be used to plot different types of signals or vectors. In this lab, the students will learn to plot signals in various ways and to use different features of plotting signals in MATLAB.

Tasks:

The following tasks are to be performed by the students.

Task 1:

Write a MATLAB code to display the following using for loop, while loop and if statements separately:

- First 30 numbers
- First 30 even numbers
- First 30 odd numbers

Task 2:

(a) Check whether the following set of commands :

```
for i = 1:20
```

```
H(i) = i * 5
```

```
End
```

have the same result as:

```
H = 1:20;
```

```
H = H*5
```

(b) Check whether following set of commands:

```
for n = 1:100
```

```
x(n) = sin(n*pi/10)
```

```
    end  
have the same result as:  
n = 1:100;  
x = sin(n*pi/10)
```

Task 3:

Run the following three MATLAB lines of code and explain why the plots are different:

- `t=0:2*pi; plot(t, sin(t))`
- `t=0:0.2:2*pi;plot(t, sin(t))`
- `t=0:0.02:2*pi; plot(t, sin(t))`

Task 4:

For the following, use the signal described as:

`t=0:0.2:2*pi`

Now perform the following operations on the signal t:

- Put two plots on the same axis, i.e. $\sin(t)$ and $\sin(2t)$
- Produce a plot without connecting the points
- Try the following command and comment:
- `t=0:0.2:2*pi; plot (t, sin(t), t, sin(t),'r.')`

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

Tasks:

Task 1:

```
52 first_30 = []
53
54 for i = 1:30
55     first_30(i) = i;
56 end
57
58 display(first_30, 'first_30')
59
60 first_30 = 1×30
```

```
59     first_30_even = []
60     a = 1;
61     for i = 0:100;
62         if mod(i, 2) == 0
63             first_30_even(a) = i;
64             a = a+1;
65         elseif size(first_30_even,2) == 30
66             break
67         end
68     end
69
70     display(first_30_even, 'first_30_even')
71
72     first_30_even = 1x30
73         0    2    4    6    8    10   12   14   16   18   20   22   24   26   28   30   32   34   36   38   40   42   44   46   48   50   52   54   56   58
```

```
71 first_30_even = [];
72 a = 1;
73 i = 0;
74
75 while i <= 100;
76   if mod(i, 2) == 0
77     first_30_even(a) = i;
78     a = a+1;
79   else size(first_30_even,2) == 30
80     break
81   end
82   i = i+1;
83
84 end
85 display(first_30_even, 'first_30_even')
```

```
86     first_30_odd = [];
87     a = 1;
88     for i = 0:100;
89       if mod(i, 2) == 0
90         first_30_odd(a) = i;
91       a = a+1;
92     elseif size(first_30_odd,2) == 30
93       break
94     end
95   end
96
97   display(first_30_odd, 'first_30_odd')

first_30_odd = 1×30
    0    2    4    6    8    10   12   14   16   18   20   22   24   26   28   30   32   34   36   38   40   42   44   46   48   50   52   54   56   58
```

```

98 first_30_odd = [];
99 a = 1;
100 for i = 0:100;
101 if mod(i, 2) == 1
102 first_30_odd(a) = i;
103 a = a+1;
104 elseif size(first_30_odd,2) == 30
105 break
106 end
107 end
108
109 display(first_30_odd, 'first_30_odd')
110
111 first_30_odd = [];
112 a = 1;
113 i = 0;
114 while i <= 100;
115 if mod(i, 2) == 1
116 first_30_odd(a) = i;
117 a = a+1;
118 elseif size(first_30_odd,2) == 30
119 break
120 end
121 i = i+1;
122 end
123
124 display(first_30_odd, 'first_30_odd')
125
126 first_30_odd = 1:30
127
128
129
130
131
132
133
134
135
136
137
138

```

Task 1 Description:

This code segment generates two sets of numbers, 'first_30_even' and 'first_30_odd,' containing the first 30 even and odd numbers, respectively, within the range of 0 to 100. It employs 'for' loops and 'while' loops to iterate through the numbers, adding them to the respective arrays while ensuring a maximum size of 30. The resulting arrays are then displayed with appropriate labels, showcasing both 'for' and 'while' loop implementations for this task.

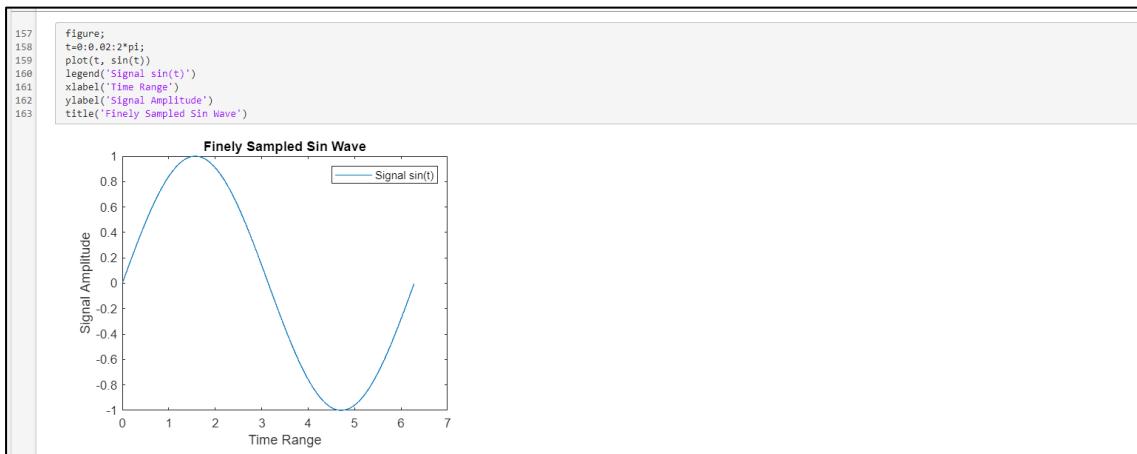
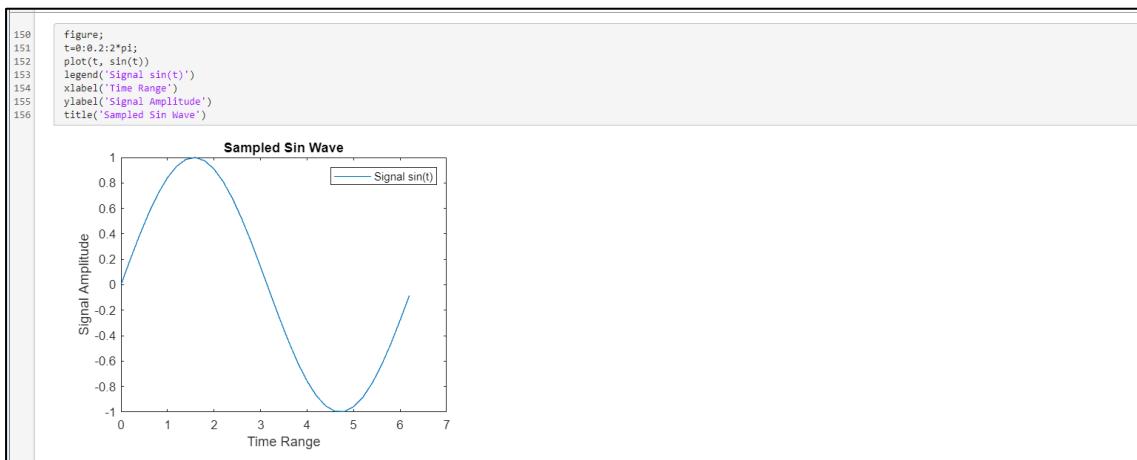
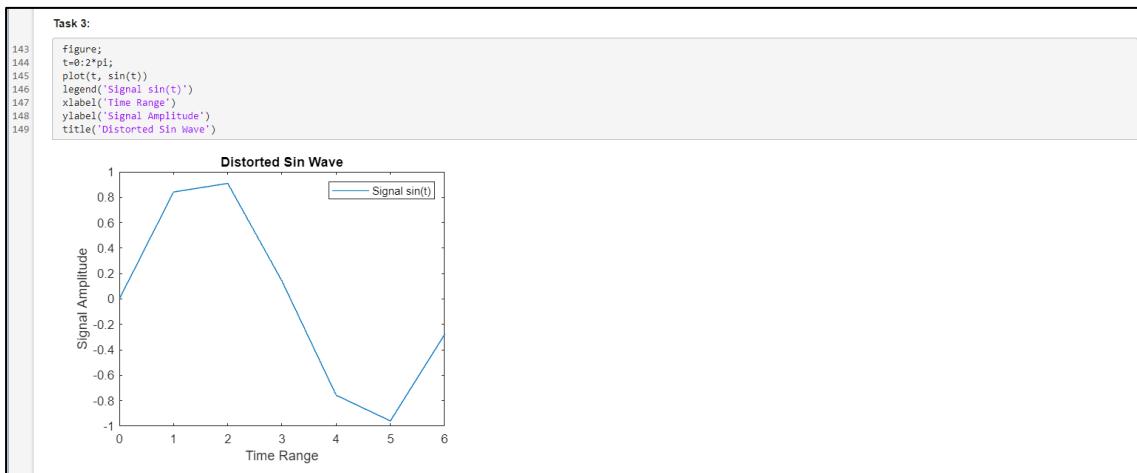
Task 2:

<pre> Task 2: 125 for i = 1:20 126 H(i) = i*5; 127 end 128 129 H = 130 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 131 132 133 134 H = 1:20; 135 H = H * 5; 136 137 H = 138 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 139 140 141 142 </pre>
--

Task 2 Description:

In Task 2 of this code segment, we demonstrate signal generation and manipulation techniques. We create an array 'H' containing scaled integers using a 'for' loop, then redefine 'H' with a vectorized approach. Similarly, we generate a sinusoidal signal 'x' using a 'for' loop and later redefine it with vectorized operations. These steps exemplify MATLAB's array-handling capabilities, showcasing efficient signal processing methods relevant to the study of signals and systems.

Task 3:

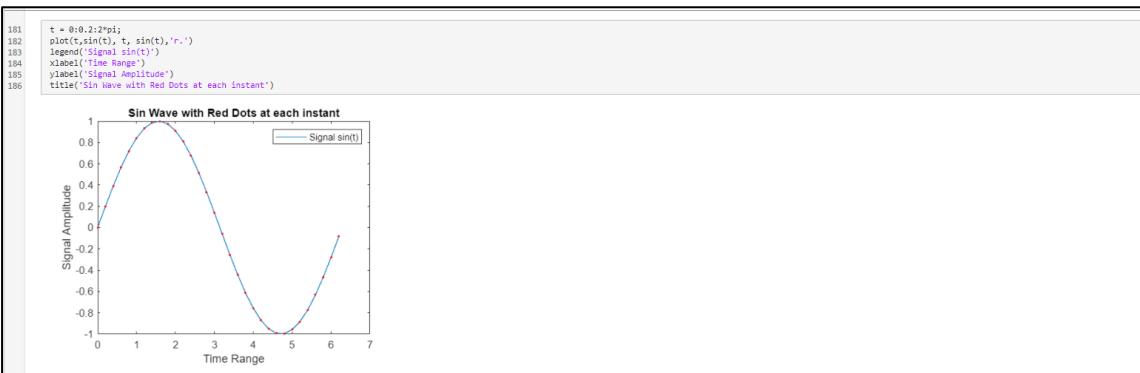
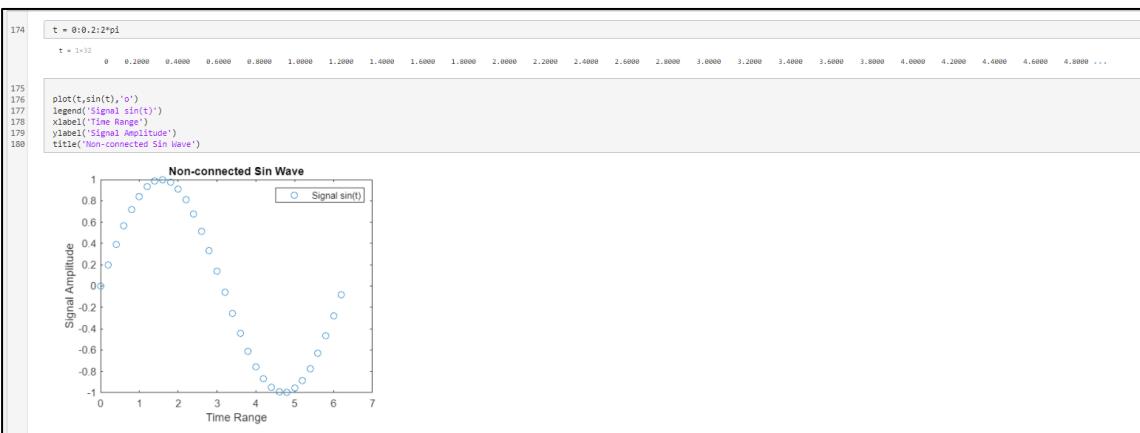
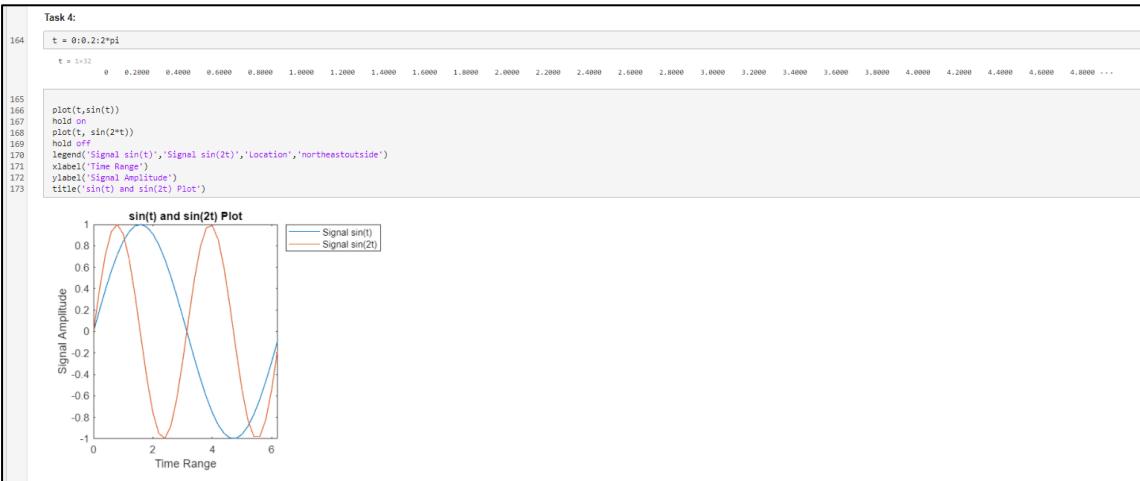


Task 3 Description:

Task 3 of this code segment focuses on visualizing and sampling a sinusoidal signal. It begins by displaying an undistorted continuous sinusoidal waveform, followed by sampled versions with varying time intervals, highlighting the concept of signal discretization and its effect on signal representation. This code provides a clear

demonstration of signal visualization and the crucial role of sampling in understanding signals within the context of signals and systems.

Task 4:



Task 4 Description:

In Task 4 of this code segment, we compare and visualize two sinusoidal signals, ' $\sin(t)$ ' and ' $\sin(2t)$ ', using various plotting techniques. The first graph juxtaposes both signals over a shared time range, offering a clear comparison, while the second figure displays ' $\sin(t)$ ' with non-connected data points, emphasizing discrete values. The third plot presents ' $\sin(t)$ ' with red dots at each data point, providing an alternative visual representation. These visualizations illustrate the versatility of MATLAB in signal analysis, offering valuable insights into signal comparison and representation techniques, essential for comprehending signals and systems.

Conclusion:

In this lab, important MATLAB ideas like loops, conditional statements, and signal visualization were explored practically. Through a series of exercises, we discovered how to take advantage of vectorized operations for efficiency and how to carry out repetitive activities using for and while loops. We also learned how important it is to use the right sampling rate when graphing signals.

Verified: _____

Date/Time: _____

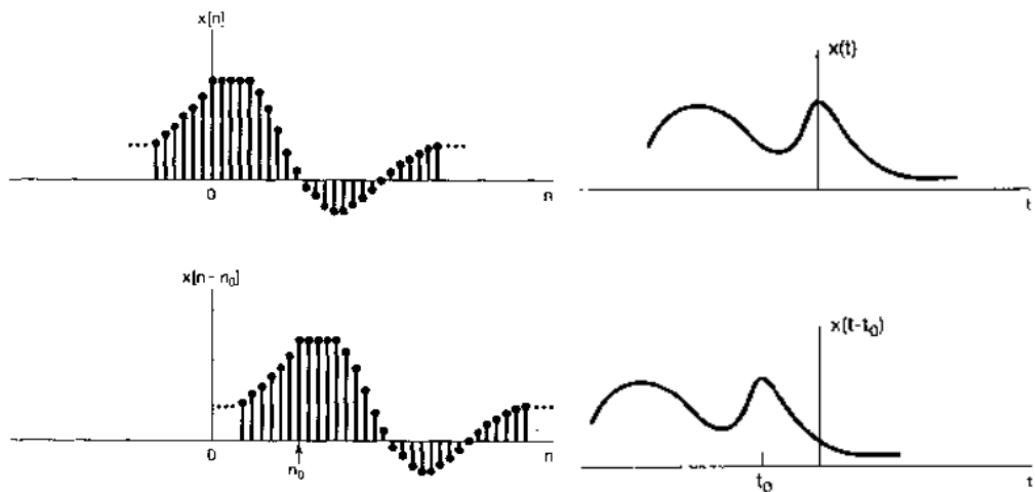
Experiment No. 3

Objective:

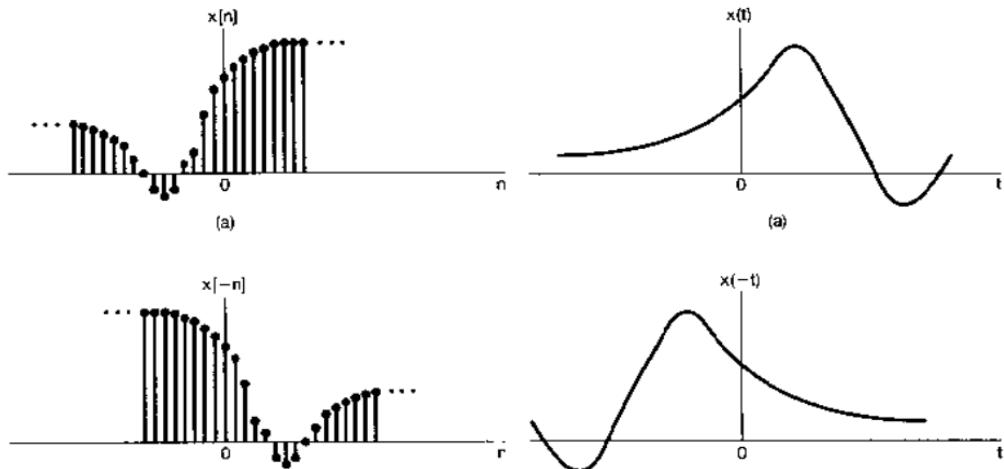
The objective of this lab is to practice some basic operations performed on a signal like shifting and flipping. Also, students will learn how to prove the stability and causality properties of a signal in MATLAB. An introduction to convolution is also included in this lab.

Theoretical Background:

For any signal, say $x(t)$, the independent variable ‘t’ can be transformed in various ways, including shifting and flipping. These two operations will be observed in this lab. Shifting of a signal in time, or the independent variable, is shown in the figure below for a discrete time signal, and a continuous time signal. It can be seen in this figure that the signal $x[n]$ is shifted by a factor of n_0 , making the signal $x[n-n_0]$. Similar observations can be made about $x(t)$. These signals are exactly similar in shape, but are shifted or displaced relative to each other. Time shifts can be observed for both continuous and discrete time signals.



A second basic transformation of the time axis is the time reversal. The time reversed or time flipped signal is obtained from a signal $x[n]$ by reflection about $n=0$. Thus $x[-n]$ is the signal $x[n]$ displayed backwards. Similarly, $x(-t)$ is the flipped signal for $x(t)$. The time reversal of discrete and continuous time signals is shown below:



Tasks:

The following tasks are to be performed by the students.

Task 1:

Generate four basic discrete time signals(unit step, unit impulse, sinusoid and exponential). Perform following operations on them:

- Shifting (with user defined shift)
- Flipping

Task 2:

Make stem plots of the following signals. Decide for yourself what the range of n should be.

- $f(n) = u(n) - u(n-4)$
- $g(n) = n \cdot u(n) - 2(n-4) \cdot u(n-4) + (n-8) \cdot u(n-8)$
- $x(n) = \delta(n) - 2\delta(n-4)$
- $y(n) = 0.9^n (u(n) - u(n-20))$
- $v(n) = \cos(0.12\pi n) u(n)$

Task 3:

$$f(n) = u(n) - u(n-4)$$

$$g(n) = n \cdot u(n) - 2(n-4) \cdot u(n-4) + (n-8) \cdot u(n-8).$$

Make stem plots of the following convolutions. Use the MATLAB conv command to compute the convolutions.

- (a) $f(n)*f(n)$
- (c) $f(n)*g(n)$
- (d) $g(n)*\delta(n)$

(e) $g(n)*g(n)$

Comment on your observations:

Use the commands title, xlabel, ylabel to label the axes of your plots.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Description of the Lab:

The objective of this laboratory session is to provide students with hands-on experience in working with signals and systems, focusing on fundamental operations such as shifting, flipping, stability, causality, and convolution. This lab is designed to reinforce theoretical knowledge with practical implementation in MATLAB, helping students gain a deeper understanding of signal processing concepts.

Theoretical Background:

Before delving into the tasks, it's crucial to understand the theoretical concepts underlying the lab experiments. In signal processing, a signal $x(t)$ or $x[n]$ can be transformed through operations like shifting and flipping. Shifting a signal introduces a time delay, making the original and shifted signals identical in shape but offset in time. Time reversal, on the other hand, reflects a signal about an axis (usually $n=0$ or $t=0$), creating a mirrored version of the original signal.

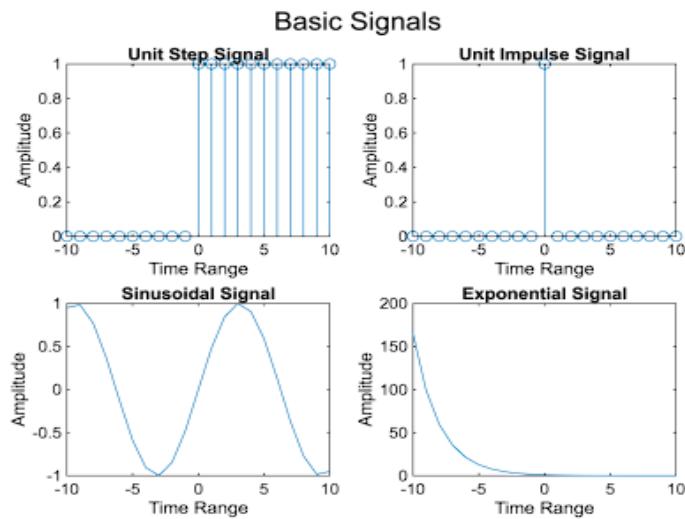
Task 1:

```
Task 1:  
  
% Generate basic signals  
n = -10:10; % Define the time range  
  
% Unit Step Signal  
u = @(n) (n >= 0);  
unit_step = u(n);  
  
% Unit Impulse Signal  
unit_impulse = zeros(size(n));  
unit_impulse(n == 0) = 1;  
  
% Sinusoidal Signal  
frequency = 0.08; % Adjust frequency as needed  
sinusoid = sin(2*pi*frequency*n);  
  
% Exponential Signal  
alpha = 0.6; % Adjust alpha as needed  
exponential = alpha.^n;  
  
% Plot the original signals  
figure;  
  
subplot(2, 2, 1);  
stem(n, unit_step);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Unit Step Signal');  
  
subplot(2, 2, 2);  
stem(n, unit_impulse);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Unit Impulse Signal');  
  
subplot(2, 2, 3);  
plot(n, sinusoid);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Sinusoidal Signal');  
  
subplot(2, 2, 4);  
plot(n, exponential);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Exponential Signal');
```

Code Description:

In this code, we have generated a discrete range of values of a variable ‘n’ and then used that n to define the unit step signal, unit impulse signal, sinusoidal signal, and an exponential signal. We then use these signals to generate subplots of each of these signals.

```
sgtitle('Basic Signals');
```



```
% Shifting operation (user-defined shift)
shift_amount = input('Enter the amount to shift: ');

shifted_unit_step = u(n - shift_amount);
shifted_unit_impulse = zeros(size(n));
shifted_unit_impulse(n == shift_amount) = 1;
shifted_sinusoid = sin(2*pi*frequency*(n - shift_amount));
shifted_exponential = alpha.^ (n - shift_amount);

% Plot the shifted signals
figure;

subplot(5, 1, 1);
stem(n, shifted_unit_step);
xlabel('Time Range')
ylabel('Amplitude')
title(['Shifted Unit Step Signal (Shifted by ' num2str(shift_amount), ')'])

subplot(5, 1, 2);
stem(n, shifted_unit_impulse);
xlabel('Time Range')
ylabel('Amplitude')
title(['Shifted Unit Impulse Signal (Shifted by ' num2str(shift_amount), ')'])
```

Code Description:

The basic signals that we previously defined are now all shifted by an amount that is defined by the user each time. Then, these shifted signals are plotted onto separate subplots. We can compare this with the previous subplots to identify the shift.

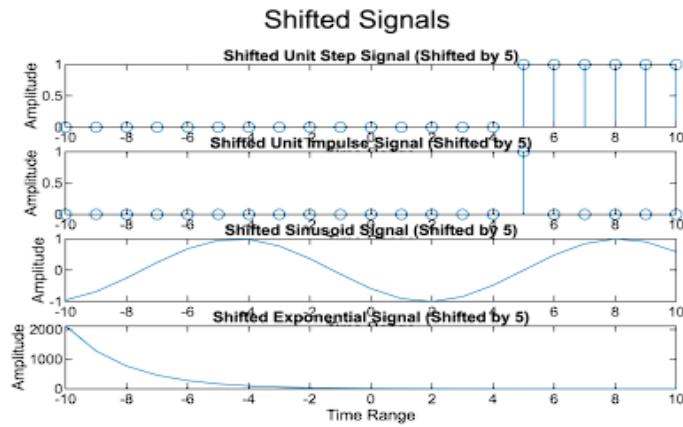
```

subplot(5, 1, 3);
plot(n, shifted_sinusoid);
xlabel('Time Range')
ylabel('Amplitude')
title(['Shifted Sinusoid Signal (Shifted by ' num2str(shift_amount),')'])

subplot(5, 1, 4);
plot(n, shifted_exponential);
xlabel('Time Range')
ylabel('Amplitude')
title(['Shifted Exponential Signal (Shifted by ' num2str(shift_amount),')'])

sgtitle('Shifted Signals');

```



```

flipped_unit_step = u(-n);
flipped_unit_impulse = zeros(size(n));
flipped_unit_impulse(n == 0) = -1;
flipped_sinusoid = sin(2*pi*frequency*(-n));
flipped_exponential = alpha.^(-n);

% Plot the shifted signals
figure;

subplot(4, 1, 1);
stem(n, flipped_unit_step);
xlabel('Time Range')
ylabel('Amplitude')

```

```

title(['Flipped Unit Step Signal (Shifted by ' num2str(shift_amount),')'])

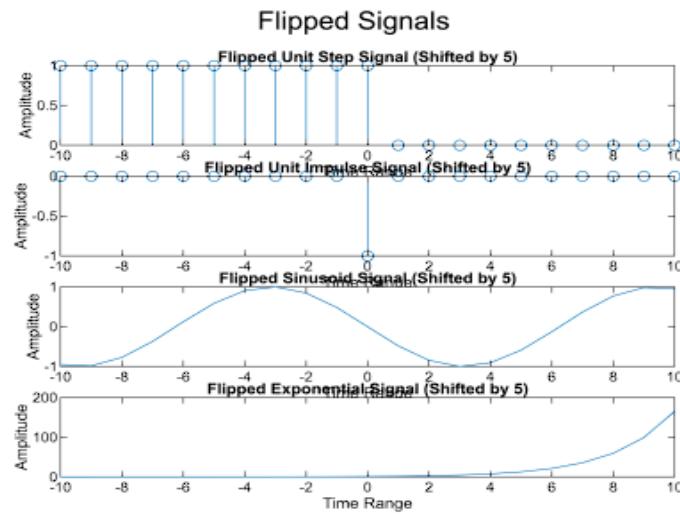
subplot(4, 1, 2);
stem(n, flipped_unit_impulse);
xlabel('Time Range')
ylabel('Amplitude')
title(['Flipped Unit Impulse Signal (Shifted by ' num2str(shift_amount),')'])

subplot(4, 1, 3);
plot(n, flipped_sinusoid);
xlabel('Time Range')
ylabel('Amplitude')
title(['Flipped Sinusoid Signal (Shifted by ' num2str(shift_amount),')'])

subplot(4, 1, 4);
plot(n, flipped_exponential);
xlabel('Time Range')
ylabel('Amplitude')
title(['Flipped Exponential Signal (Shifted by ' num2str(shift_amount),')'])

sgtitle('Flipped Signals');

```



Code Description:

The last part of the code is to flip the signals. We input inverse values of 'n' into the functions of the basic signals that we have defined and then we plotted these values onto a graph. Hence, we were able to generate inverse/flipped versions of each signal.

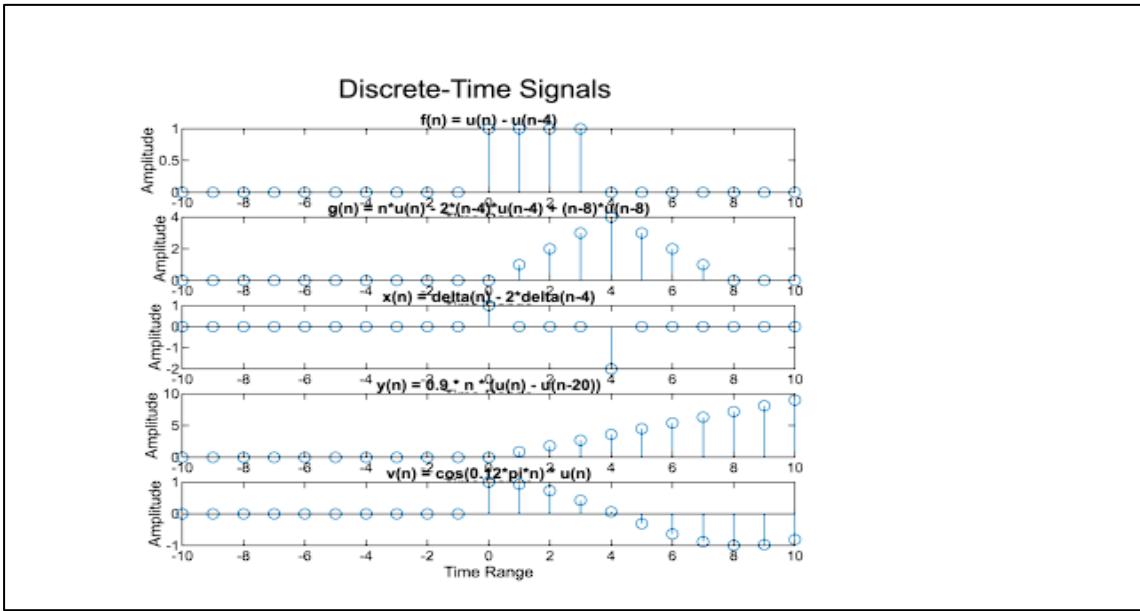
Task 2:

Task 2:

```
% Generate the impulse signal  
delta = @(n) (n == 0); % Unit impulse function
```

4

```
f = u(n) - u(n - 4);  
  
g = n .* u(n) - 2 * (n - 4) .* u(n - 4) + (n - 8) .* u(n - 8);  
  
x = delta(n) - 2 * delta(n - 4);  
  
y = 0.9 * n .* (u(n) - u(n - 20));  
  
v = cos(0.12 * pi * n) .* u(n);  
  
%Plot the subplots  
figure;  
  
subplot(5, 1, 1);  
stem(n, f);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('f(n) = u(n) - u(n-4)');  
  
subplot(5, 1, 2);  
stem(n, g);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('g(n) = n*u(n) - 2*(n-4)*u(n-4) + (n-8)*u(n-8)');  
  
subplot(5, 1, 3);  
stem(n, x);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('x(n) = delta(n) - 2*delta(n-4)');  
  
subplot(5, 1, 4);  
stem(n, y);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('y(n) = 0.9 * n * (u(n) - u(n-20))');  
  
subplot(5, 1, 5);  
stem(n, v);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('v(n) = cos(0.12*pi*n) * u(n)');  
  
sgtitle('Discrete-Time Signals');
```



Code Description:

This task involves creating stem plots for five specific signals, each with distinct characteristics:

1. $f(n)=u(n)-u(n-4)$
2. $g(n)=n\cdot u(n)-2(n-4)\cdot u(n-4)+(n-8)\cdot u(n-8)$
3. $x(n)=\delta(n)-2\delta(n-4)$
4. $y(n)=0.9n\cdot(u(n)-u(n-20))$
5. $v(n)=\cos(0.12\pi n)\cdot u(n)$

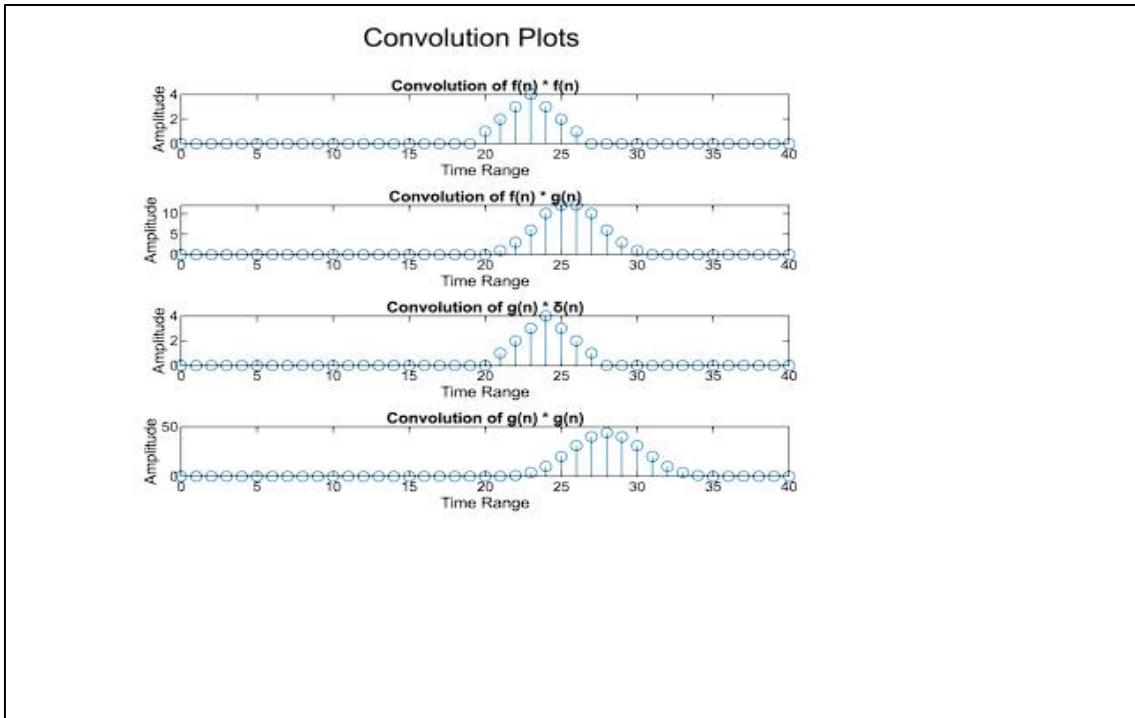
Students are encouraged to determine the appropriate range for 'n' and observe the characteristics of these signals using stem plots.

Task 3:

```
Task 3:  
f = u(n) - u(n - 4);  
g = n .* u(n) - 2 * (n - 4) .* u(n - 4) + (n - 8) .* u(n - 8);  
  
conv_ff = conv(f, f);  
  
conv_fg = conv(f, g);  
  
conv_g_delta = conv(g, delta(n));  
  
conv_gg = conv(g, g);  
  
% Plot the convolutions  
figure;  
  
subplot(4, 1, 1);  
stem(0:2*length(n)-2, conv_ff);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Convolution of f(n) * f(n)');  
  
subplot(4, 1, 2);  
stem(0:2*length(n)-2, conv_fg);  
xlabel('Time Range')  
ylabel('Amplitude')
```

6

```
title('Convolution of f(n) * g(n)');  
  
subplot(4, 1, 3);  
stem(0:2*length(n)-2, conv_g_delta);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Convolution of g(n) * δ(n)');  
  
subplot(4, 1, 4);  
stem(0:2*length(n)-2, conv_gg);  
xlabel('Time Range')  
ylabel('Amplitude')  
title('Convolution of g(n) * g(n)');  
  
sgtitle('Convolution Plots');
```



Code Description:

The final task introduces convolution, a fundamental operation in signal processing. Students perform convolution on the following signal pairs and examine the results:

1. $f(n)*f(n)$
2. $f(n)*g(n)$
3. $g(n)*\delta(n)$
4. $g(n)*g(n)$

The MATLAB conv command is employed to compute the convolutions, and students are prompted to provide observations and insights into the outcomes.

Conclusion:

In this lab, students engaged in hands-on exploration of fundamental signal operations, including shifting, flipping, and convolution, reinforcing their theoretical knowledge and equipping them with practical skills in MATLAB. This practical experience deepened their understanding of signals and systems, laying a strong foundation for future applications in engineering and scientific disciplines.

Verified: _____

Date/Time: _____

Experiment No. 4

Objective:

The objective of this lab is to learn the use of symbolic variables and use them to solve differential equations, and find differentiation and integration of functions.

Theoretical Background:

One of the very attractive features of MATLAB includes the symbolic math toolbox. It is of great utility in applications in which symbolic expressions are necessary for reasons of accuracy in calculations. Symbolic numbers, variables and expressions may be declared and used in MATLAB. The command used to declare a symbolic variable, expression or number is ‘sym’. The command ‘syms’ can be used to declare multiple symbolic objects at a time. Symbolic math is very useful in finding exact solution of differential equations, differentiation, integration, and simultaneous solution of equations.

Tasks:

The following tasks will help the students to familiarize themselves with the symbolic math in MATLAB and to use symbolic expressions to practice various operations.

Task 1:

Define five 5th order equations using symbolic variables. Solve each of the equations separately with respect to one variable.

Task 2:

Declare two 2nd order equations using symbolic variables and solve them simultaneously. Make five sets of equations.

Task 3 :

Declare five 5th order symbolic equations and differentiate them. Find first, second, third, fourth and fifth order derivatives

Task 4:

Find the definite integral of five symbolic expressions with lower and upper limits 0 and 1 respectively.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Description of the Lab:

The objective of this laboratory session is to introduce students to symbolic variables and their use in solving differential equations, as well as performing differentiation and integration of functions using MATLAB's Symbolic Math Toolbox. Symbolic variables are essential in situations where precise calculations are required, such as in differential equations, integration, and simultaneous equation solving.

Theoretical Background:

This lab explores the power of MATLAB's symbolic math toolbox, allowing for the declaration and use of symbolic variables, expressions, and numbers using the 'sym' and 'syms' commands. Symbolic math is particularly valuable in finding exact solutions to differential equations, performing differentiation and integration, and solving systems of equations simultaneously.

Task 1:

Task 1:

Define five 5th order equations using symbolic variables. Solve each of the equations separately with respect to one variable.

```
syms x  
f1 = 10*(x ^ 5) + 6*(x ^ 4) + 2*x + 9  
f1 = 10 x^5 + 6 x^4 + 2 x + 9
```

```
f2 = (40 * x ^ 5) + (6 * x ^ 3) + (2 * x ^ 2) + (9*x) + 1  
f2 = 40 x^5 + 6 x^3 + 2 x^2 + 9 x + 1
```

```
Equ_3 = (x ^ 5) + (6 * x) + 64  
Equ_3 = x^5 + 6 x + 64
```

```
Equ_4 = (2 * x ^ 5) + (x ^ 4) + (3* x ^3)  
Equ_4 = 2 x^5 + x^4 + 3 x^3
```

```
Equ_5 = (-21 * x ^ 5) + x - 23  
Equ_5 = -21 x^5 + x - 23
```

```
Sol_1 = solve(f1,x)
```

```
Sol_1 =  
(root(σ1,z,1))  
(root(σ1,z,2))  
(root(σ1,z,3))  
(root(σ1,z,4))  
(root(σ1,z,5))
```

where

$$σ1 = z^5 + \frac{3z^4}{5} + \frac{z}{5} + \frac{9}{10}$$

```
Sol_2 = solve(f2,x)
```

```
Sol_2 =
```

$$\begin{pmatrix} \text{root}(\sigma_1, z, 1) \\ \text{root}(\sigma_1, z, 2) \\ \text{root}(\sigma_1, z, 3) \\ \text{root}(\sigma_1, z, 4) \\ \text{root}(\sigma_1, z, 5) \end{pmatrix}$$

where

$$\sigma_1 = z^5 + \frac{3z^3}{20} + \frac{z^2}{20} + \frac{9z}{40} + \frac{1}{40}$$

```
Sol_3 = solve(Equ_3,x)
```

$$\begin{pmatrix} \text{root}(z^5 + 6z + 64, z, 1) \\ \text{root}(z^5 + 6z + 64, z, 2) \\ \text{root}(z^5 + 6z + 64, z, 3) \\ \text{root}(z^5 + 6z + 64, z, 4) \\ \text{root}(z^5 + 6z + 64, z, 5) \end{pmatrix}$$

```
Sol_4 = solve(Equ_4,x)
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ -\frac{1}{4} - \frac{\sqrt{23}i}{4} \\ -\frac{1}{4} + \frac{\sqrt{23}i}{4} \end{pmatrix}$$

```
Sol_5 = solve(Equ_5,x)
```

$$\begin{pmatrix} \text{root}\left(z^5 - \frac{z}{21} + \frac{23}{21}, z, 1\right) \\ \text{root}\left(z^5 - \frac{z}{21} + \frac{23}{21}, z, 2\right) \\ \text{root}\left(z^5 - \frac{z}{21} + \frac{23}{21}, z, 3\right) \\ \text{root}\left(z^5 - \frac{z}{21} + \frac{23}{21}, z, 4\right) \\ \text{root}\left(z^5 - \frac{z}{21} + \frac{23}{21}, z, 5\right) \end{pmatrix}$$

Task 1 - Solving 5th Order Equations:

In this task, five 5th-order equations are defined using symbolic variables, and each equation is solved separately with respect to one variable. The symbolic variable 'x' is utilized to illustrate how MATLAB's symbolic math capabilities can solve complex equations accurately.

Task 2:

Task 2:

Declare two 2nd order equations using symbolic variables and solve them simultaneously. Make five sets of equations.

```
syms x y f(x,y) g(x,y)
```

```
%Set 1
```

```
f(x,y) = (x ^ 2) - (3*x*y) - x + (2 * y ^2)
```

```
f(x, y) =  $x^2 - 3xy - x + 2y^2$ 
```

```
g(x,y) = (2 * x ^ 2) - (3 *x*y)+ (y^2) - 20
```

```
g(x, y) =  $2x^2 - 3xy + y^2 - 20$ 
```

```
%Set 2
```

```
Equ_3 = (2 * x ^ 2) - (5 * x * y) - (3 * y)
```

```
Equ_3 =  $2x^2 - 5xy - 3y$ 
```

```
Equ_4 = (x ^ 2) + (4 * x) - 5
```

```
Equ_4 =  $x^2 + 4x - 5$ 
```

```
%Set 3
```

```
Equ_5 = (x ^ 2) + (y ^ 2) - 9
```

```
Equ_5 =  $x^2 + y^2 - 9$ 
```

```
Equ_6 = (y ^ 2) + ((2 * y - 3) ^ 2) - 9
```

```
Equ_6 =  $(2y - 3)^2 + y^2 - 9$ 
```

```
%Set 4
```

```
Equ_7 = (5 * x ^ 2) - (6 * y) + 5
```

```
Equ_7 =  $5x^2 - 6y + 5$ 
```

```
Equ_8 = (5 * y ^ 2) - (6 * x) + 5
```

```
Equ_8 =  $5y^2 - 6x + 5$ 
```

```
%Set 5
```

```
Equ_9 = (x ^ 2) - (3 * y) - 6
```

```
Equ_9 =  $x^2 - 3y - 6$ 
```

```
Equ_10 = (5 * x ^ 2) - (6 * y) - 1
```

```
Equ_10 =  $5x^2 - 6y - 1$ 
```

```
%Sol 1  
[Sol1_x, Sol1_y] = solve(f(x,y),g(x,y))
```

$$\begin{aligned} \text{Sol1_x} = \\ \left(\begin{array}{c} -8 \\ -\frac{5\sqrt{145}}{6} - \frac{35}{6} \\ \frac{5\sqrt{145}}{6} - \frac{35}{6} \end{array} \right) \end{aligned}$$

$$\begin{aligned} \text{Sol1_y} = \\ \left(\begin{array}{c} -6 \\ -\frac{2\sqrt{145}}{3} - \frac{20}{3} \\ \frac{2\sqrt{145}}{3} - \frac{20}{3} \end{array} \right) \end{aligned}$$

```
%Sol 2  
[Sol2_x, Sol2_y] = solve(Equ_3,Equ_4,x,y)
```

$$\text{Sol2_x} =$$

$$\begin{pmatrix} 1 \\ -5 \end{pmatrix}$$

$$\text{Sol2_y} =$$

$$\begin{pmatrix} \frac{1}{4} \\ -\frac{25}{11} \end{pmatrix}$$

```
%Sol 3  
[Sol3_x, Sol3_y] = solve(Equ_5,Equ_6,x,y)
```

$$\text{Sol3_x} =$$

$$\begin{pmatrix} -3 \\ 3 \\ -\frac{9}{5} \\ \frac{9}{5} \end{pmatrix}$$

$$\text{Sol3_y} =$$

$$\begin{pmatrix} 0 \\ 0 \\ \frac{12}{5} \\ \frac{12}{5} \end{pmatrix}$$

```
%Sol 4
```

```
[Sol4_x, Sol4_y] = solve(Equ_7,Equ_8,x,y)
```

$$\text{Sol4_x} = \begin{pmatrix} \frac{3}{5} - \frac{4}{5}i \\ \frac{3}{5} + \frac{4}{5}i \\ -\frac{3}{5} + \frac{2\sqrt{13}}{5}i \\ -\frac{3}{5} - \frac{2\sqrt{13}}{5}i \end{pmatrix}$$

$$\text{Sol4_y} = \begin{pmatrix} \frac{3}{5} - \frac{4}{5}i \\ \frac{3}{5} + \frac{4}{5}i \\ -\frac{3}{5} - \frac{2\sqrt{13}}{5}i \\ -\frac{3}{5} + \frac{2\sqrt{13}}{5}i \end{pmatrix}$$

```
%Sol1_5
```

```
[Sol5_x, Sol5_y] = solve(Equ_9,Equ_10,x,y)
```

$$\text{Sol5_x} = \begin{pmatrix} -\frac{\sqrt{33}}{3}i \\ \frac{\sqrt{33}}{3}i \end{pmatrix}$$

$$\text{Sol5_y} = \begin{pmatrix} -\frac{29}{9} \\ -\frac{29}{9} \end{pmatrix}$$

Task 2 - Simultaneous Equation Solving:

For this task, two 2nd-order equations are declared with symbolic variables and solved simultaneously in five different sets. This demonstrates how the symbolic math toolbox can handle systems of equations effectively, providing solutions for both 'x' and 'y.'

Task 3:

Task 3:

Declare five 5th order symbolic equations and differentiate them. Find first, second, third, fourth and fifth order derivatives.

%First Equation

```
Equ_1 = (4 * x ^ 5) - (2 * x ^ 4) + (5 * x ^ 3) - (4 * x ^ 2) + (3 * x) + 7
```

```
Equ_1 = 4 x^5 - 2 x^4 + 5 x^3 - 4 x^2 + 3 x + 7
```

```
differential1_Equ1 = diff(Equ_1,x,1)
```

5

```
differential1_Equ1 = 20 x^4 - 8 x^3 + 15 x^2 - 8 x + 3
```

```
differential2_Equ1 = diff(Equ_1,x,2)
```

```
differential2_Equ1 = 80 x^3 - 24 x^2 + 30 x - 8
```

```
differential3_Equ1 = diff(Equ_1,x,3)
```

```
differential3_Equ1 = 240 x^2 - 48 x + 30
```

```
differential4_Equ1 = diff(Equ_1,x,4)
```

```
differential4_Equ1 = 480 x - 48
```

```
differential5_Equ1 = diff(Equ_1,x,5)
```

```
differential5_Equ1 = 480
```

%Second Equation

```
Equ_2 = (4 * x ^ 5) + (2 * x ^ 4) - (3 * x ^ 3) + 1
```

```
Equ_2 = 4 x^5 + 2 x^4 - 3 x^3 + 1
```

```
differential1_Equ2 = diff(Equ_2,x,1)
```

```
differential1_Equ2 = 20 x^4 + 8 x^3 - 9 x^2
```

```
differential2_Equ2 = diff(Equ_2,x,2)
```

```
differential2_Equ2 = 80 x^3 + 24 x^2 - 18 x
```

```
differential3_Equ2 = diff(Equ_2,x,3)
```

```
differential3_Equ2 = 240 x^2 + 48 x - 18
```

```
differential4_Equ2 = diff(Equ_2,x,4)
```

```
differential4_Equ2 = 480 x + 48
```

```

differential5_Equ2 = diff(Equ_2,x,5)

differential5_Equ2 = 480

%Third Equation
Equ_3 = (x ^ 5) + (6 * x) + 64

Equ_3 = x^5+6 x+64

differential1_Equ3 = diff(Equ_3,x,1)

differential1_Equ3 = 5 x^4+6

```

6

```

differential2_Equ3 = diff(Equ_3,x,2)

differential2_Equ3 = 20 x^3

differential3_Equ3 = diff(Equ_3,x,3)

differential3_Equ3 = 60 x^2

differential4_Equ3 = diff(Equ_3,x,4)

differential4_Equ3 = 120 x

differential5_Equ3 = diff(Equ_3,x,5)

differential5_Equ3 = 120

%Fourth Equation
Equ_4 = (2 * x ^ 5) + (x ^ 4) + (3* x ^3)

Equ_4 = 2 x^5+x^4+3 x^3

differential1_Equ4 = diff(Equ_4,x,1)

differential1_Equ4 = 10 x^4+4 x^3+9 x^2

differential2_Equ4 = diff(Equ_4,x,2)

differential2_Equ4 = 40 x^3+12 x^2+18 x

differential3_Equ4 = diff(Equ_4,x,3)

differential3_Equ4 = 120 x^2+24 x+18

differential4_Equ4 = diff(Equ_4,x,4)

differential4_Equ4 = 240 x+24

differential5_Equ4 = diff(Equ_4,x,5)

differential5_Equ4 = 240

```

```
%Fifth Equation
Equ_5 = (-21 * x ^ 5) + x - 23

Equ_5 = -21x^5 + x - 23

differential1_Equ5 = diff(Equ_5,x,1)

differential1_Equ5 = 1 - 105x^4

differential2_Equ5 = diff(Equ_5,x,2)
```

7

```
differential2_Equ5 = -420x^3

differential3_Equ5 = diff(Equ_5,x,3)

differential3_Equ5 = -1260x^2

differential4_Equ5 = diff(Equ_5,x,4)

differential4_Equ5 = -2520x

differential5_Equ5 = diff(Equ_5,x,5)

differential5_Equ5 = -2520
```

Task 3 - Differentiation:

Five 5th-order symbolic equations are declared, and their first, second, third, fourth, and fifth-order derivatives are computed. This task showcases the utility of symbolic math in performing differentiation, aiding in the understanding of higher-order derivatives.

Task 4:

Task 4:

Find the definite integral of five symbolic expressions with lower and upper limits 0 and 1 respectively.

%First Equation

$$\text{Equ_1} = (4 * x^5) - (2 * x^4) + (5 * x^3) - (4 * x^2) + (3 * x) + 7$$

$$\text{Equ_1} = 4x^5 - 2x^4 + 5x^3 - 4x^2 + 3x + 7$$

$$\text{Integral_Equ1} = \int(\text{Equ_1}, x, 0, 1)$$

$$\text{Integral_Equ1} =$$

$$\frac{521}{60}$$

%Second Equation

$$\text{Equ_2} = (4 * x^5) + (2 * x^4) - (3 * x^3) + 1$$

$$\text{Equ_2} = 4x^5 + 2x^4 - 3x^3 + 1$$

$$\text{Integral_Equ2} = \int(\text{Equ_2}, x, 0, 1)$$

$$\text{Integral_Equ2} =$$

$$\frac{79}{60}$$

%Third Equation

$$\text{Equ_3} = (x^5) + (6 * x) + 64$$

$$\text{Equ_3} = x^5 + 6x + 64$$

$$\text{Integral_Equ3} = \int(\text{Equ_3}, x, 0, 1)$$

$$\text{Integral_Equ3} =$$

$$\frac{403}{6}$$

%Fourth Equation

8

$$\text{Equ_4} = (2 * x^5) + (x^4) + (3 * x^3)$$

$$\text{Equ_4} = 2x^5 + x^4 + 3x^3$$

$$\text{Integral_Equ4} = \int(\text{Equ_4}, x, 0, 1)$$

$$\text{Integral_Equ4} =$$

$$\frac{77}{60}$$

%Fifth Equation

$$\text{Equ_5} = (-21 * x^5) + x - 23$$

$$\text{Equ_5} = -21x^5 + x - 23$$

```
Integral_Equ5 = int(Equ_5,x,0,1)
```

```
Integral_Equ5 = -26
```

Conclusion:

This report has explored one of MATLAB's key capabilities; that is Symbolic Lab. This capability allows us to find solutions for complicated equations quickly. The main objectives of this lab are to solve one-variable linear equations, quadratic equations, derivatives, and integrals.

Task 4 - Definite Integration:

In this task, the definite integral of five symbolic expressions is calculated, with lower and upper limits set to 0 and 1, respectively. It illustrates how symbolic math facilitates precise integration calculations for complex functions.

Conclusion:

This lab has provided a comprehensive introduction to MATLAB's Symbolic Math Toolbox. By exploring the use of symbolic variables and expressions, students have gained the ability to solve complex equations, perform differentiation of functions, and calculate definite integrals accurately. The lab's objectives, which include solving one-variable linear equations, quadratic equations, derivatives, and integrals, have been successfully achieved. Symbolic math in MATLAB proves to be an invaluable tool for accurate mathematical analysis and problem-solving, particularly in situations where exact solutions are required.

Verified: _____

Date/Time: _____

Experiment No. 5

Objective:

The objective of this lab is to create an understanding of convolution by writing a code to perform convolution of two signals.

Theoretical Background:

Convolution is the representation of an LTI system in terms of its unit impulse response. Impulse response of a system $h[n]$ is the output when a unit impulse $\delta[n]$ is given at its input. For a system with input $x[n]$ and the system impulse response $h[n]$, the output $y[n]$ of the system is calculated by convolution of the system response and input, given as:

$$y[n] = x[n] * h[n]$$

Tasks:

Write your own code for convolution of the following sets of discrete sequences, such that the convolved signal $y[n]$ is given as described above. Explain each and every step in your code. Compare your results with the results of built in conv function

Task 1:

When $x[n]$ is a unit impulse and $h[n]$ is a unit step function.

Task 2:

When both $x[n]$ and $h[n]$ are unit step functions.

Task 3:

When $x[n] = (0.5)^n u[n]$ and $h[n]$ is a unit step function.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Description of Lab:

The objective of Experiment No. 5 in this Signal and Systems laboratory is to develop a fundamental understanding of convolution. In this experiment, students will write code to perform convolution of two discrete signals and analyze the results. The primary goal is to gain practical experience in applying the convolution operation and compare the outcomes with a built-in convolution function.

Theoretical Background:

Convolution is a critical concept in the realm of Linear Time-Invariant (LTI) systems, as it allows us to represent an LTI system in terms of its unit impulse response. The impulse response of a system, denoted as $h[n]$, represents the output when a unit impulse $\delta[n]$ is given as input to the system. For a system with an input signal $x[n]$ and an impulse response $h[n]$, the output signal $y[n]$ can be calculated using the convolution operation:

Task 1:

```
% Plot the signals and their convolution result
subplot(3, 1, 1);
stem(n, x);
title('Unit Impulse Function δ[n]');
xlabel('n');
ylabel('Amplitude');

subplot(3, 1, 2);
stem(n, h);
title('Unit Step Function u[n]');
xlabel('n');
ylabel('Amplitude');

z = 1: K;
subplot(3, 1, 3);
stem(z, y_conv);
title('Convolution y[n] (Using conv)');
xlabel('n');
ylabel('Amplitude');

% Plot the signals and their convolution result

```

The figure consists of three vertically stacked stem plots. The top plot, titled 'Unit Impulse Function δ[n]', shows a single vertical blue stem at n=0 with an amplitude of 1. The middle plot, titled 'Unit Step Function u[n]', shows a series of vertical blue stems starting from n=-10 up to n=10, with amplitudes increasing from 0 to 1. The bottom plot, titled 'Convolution y[n] (Using conv)', shows a series of vertical blue stems starting from n=0 up to n=45, representing the convolution result.

Task 1 Description:

In Task 1 of this experiment, the provided code serves as a hands-on introduction to convolution, utilizing a unit impulse signal ($\delta[n]$) and a unit step function ($u[n]$) as input signals. The code showcases the manual computation of convolution through nested loops, systematically summing the product of 'x' and 'h' values at appropriate offsets to construct the 'y_manual' signal. Simultaneously, the built-in 'conv' function is employed to generate the convolution result, stored in 'y_conv'. A subsequent comparison is made to verify the equality of the manually computed result and the one produced by the 'conv' function. The code also visually represents the unit impulse, unit step, and convolution results through stem plots. This task offers a practical understanding of convolution and underscores the significance of the convolution operation in signal processing and systems analysis.

Task 2:

```
Task 2:
When both x[n] and h[n] are unit step functions.

% Define the unit step function u[n] one
n = -10:10;
x = (n >= 0);

% Define the unit step function u[n] two
h = (n >= 0);

m = length(x);
o = length(h);
k = m + o - 1;

% Initialize y_manual and y_conv as arrays of zeros
y_manual = zeros(1, k);
y_conv = zeros(1, k);

% Perform convolution manually using nested loops
for i = 1:k
    for j = 1:m
        if i - j + 1 > 0 && i - j + 1 <= o
            y_manual(i) = y_manual(i) + x(j) * h(i - j + 1);
        end
    end
end

y_manual;

% Perform convolution using the built-in 'conv' function
y_conv = conv(x, h);

% Compare the results
isequal(y_manual, y_conv) % Check if the results are equal

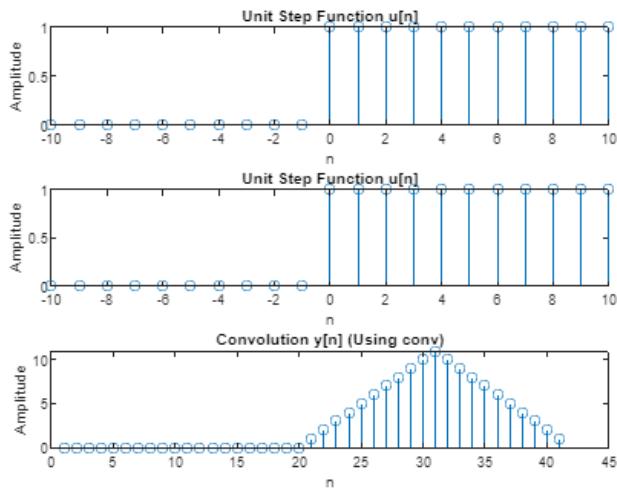
ans = logical
1
```

```
% Plot the signals and their convolution result
subplot(3, 1, 1);
stem(n, x);
title('Unit Step Function u[n]');
xlabel('n');
ylabel('Amplitude');

subplot(3, 1, 2);
stem(n, h);
title('Unit Step Function u[n]');
xlabel('n');
ylabel('Amplitude');

z = 1:k;

subplot(3, 1, 3);
stem(z, y_conv);
title('Convolution y[n] (Using conv)');
xlabel('n');
ylabel('Amplitude');
```



Task 2 Description:

In Task 2, we continue our exploration of convolution, this time with both 'x[n]' and 'h[n]' being unit step functions. The code defines these unit step functions and uses them to create 'x' and 'h.' Following the structure of Task 1, the code calculates signal lengths and initializes 'y_manual' and 'y_conv' for manual and built-in convolution results. Manual convolution is achieved through nested loops, combining 'x' and 'h' to create 'y_manual.' Simultaneously, the 'conv' function is applied to obtain 'y_conv.' A comparison confirms result equality, and stem plots visualize the unit step functions and their convolution outcome. This task deepens your understanding of convolution with unit step functions in signal processing and systems analysis.

Task 3:

When $x[n] = (0.5)^n$ and $h[n]$ is a unit step function.

```

% Define the unit step function u[n]
n = -10:10;
h = (n >= 0);

% Define x[n]
x = (0.5).^n .* (n >= 0);

m = length(x);
o = length(h);
k = m + o - 1;

% Initialize y_manual and y_conv as arrays of zeros
y_manual = zeros(1, k);
y_conv = zeros(1, k);

% Perform convolution manually using nested loops
for i = 1:k
    for j = 1:m
        if i - j + 1 > 0 && i - j + 1 <= o
            y_manual(i) = y_manual(i) + x(j) * h(i - j + 1);
        end
    end
end

y_manual;

% Perform convolution using the built-in 'conv' function
y_conv = conv(x, h);

% Compare the results
isequal(y_manual, y_conv) % Check if the results are equal
ans = Logical
1

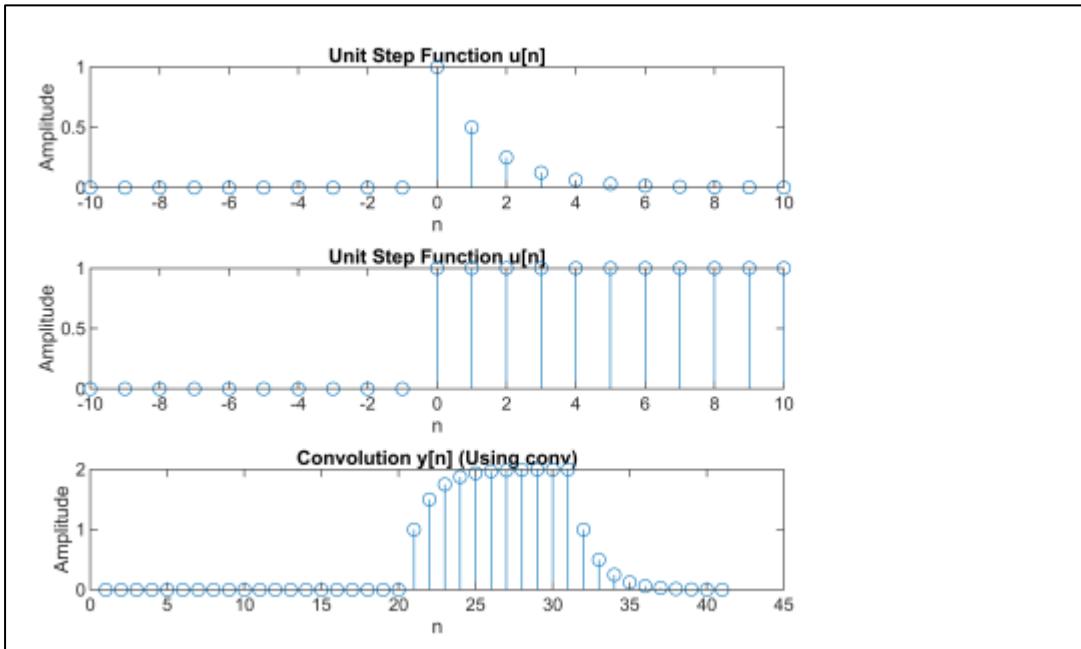
% Plot the signals and their convolution result
subplot(3, 1, 1);
stem(n, x);
title('Unit Step Function u[n]');
xlabel('n');
ylabel('Amplitude');

subplot(3, 1, 2);
stem(n, h);
title('Unit Step Function u[n]');
xlabel('n');
ylabel('Amplitude');

z = 1: k;

subplot(3, 1, 3);
stem(z, y_conv);
title('Convolution y[n] (Using conv)');
xlabel('n');
ylabel('Amplitude');

```



Conclusion:

In summary, this lab experiment has provided hands-on experience with the fundamental concept of convolution. We explored convolution for various input signals, including unit impulses, unit step functions, and custom signals. Manual convolution implementation deepened our understanding and comparing it with the 'conv' function validated its accuracy and efficiency. Visualizations in stem plots illustrated the significance of convolution in signal processing and systems analysis. This experiment has bolstered our grasp of convolution's pivotal role in real-world applications.

Verified: _____

Date/Time: _____

Experiment No. 6

Objective:

The objective of this lab is to give the students an introduction to Simulink and to learn to use some basic blocks in Simulink.

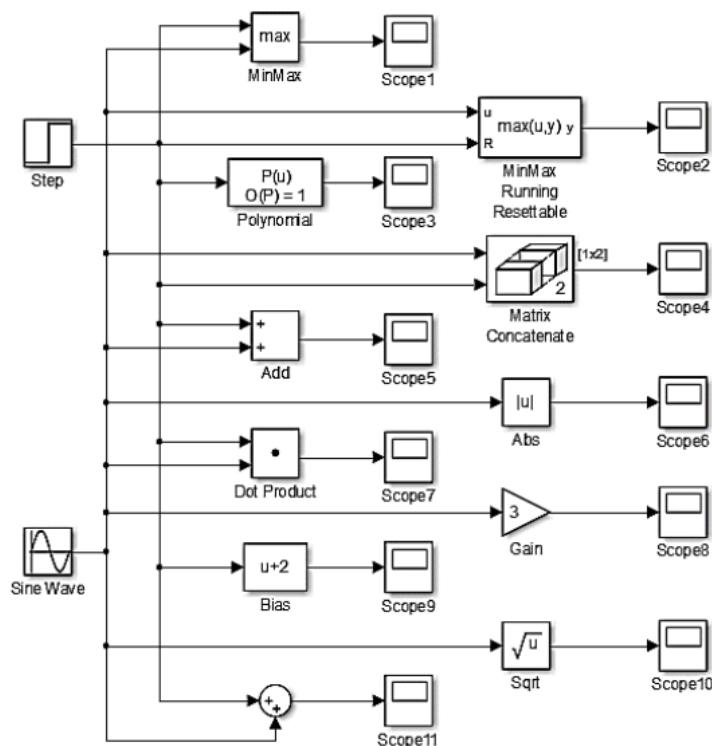
Theoretical Background:

Simulink is a graphical programming environment for modeling, simulating and analyzing dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. In this lab, the students will be introduced to the Simulink environment. The students will learn to apply various basic operations on simple signals like the unit step and sine wave, and observe the results using the ‘scope’ block in Simulink, which works like an oscilloscope.

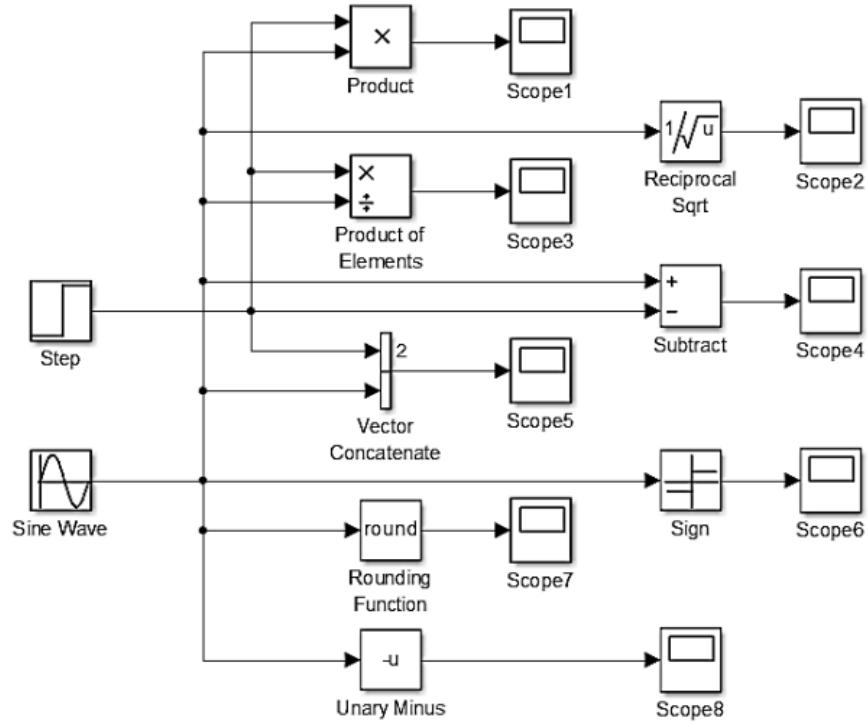
Tasks:

The students have to create each of the following block diagrams in Simulink, and learn the purpose and usage of each of the blocks used.

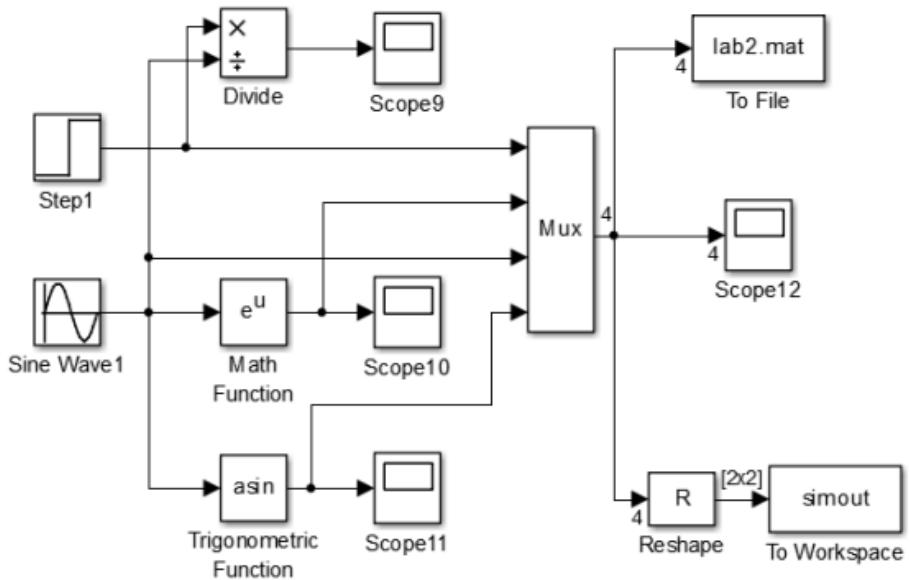
Task 1:



Task 2:



Task 3:



INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

Description:

Multiple basic operations are being done on a sine wave and a unit step. These are being shown in the images of the scope given below. The Simulink schematic for task 1 is similar to that given in the question so we have only added the output signals scope images.

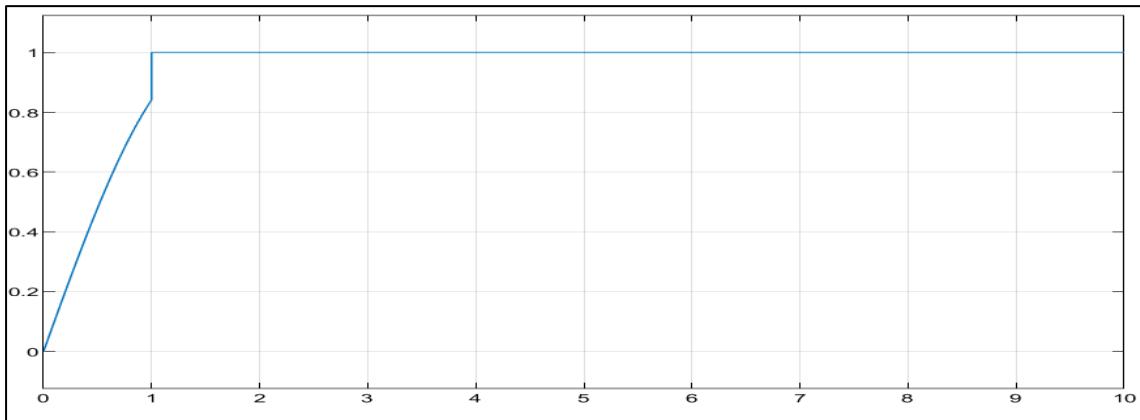


Figure 1: Scope of Max between Step Signal and Sine Wave

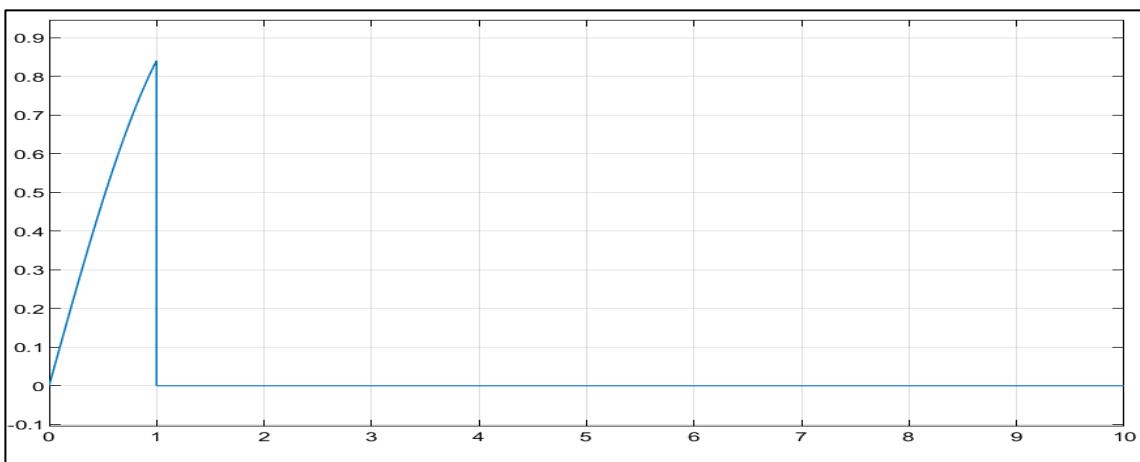


Figure 2: Scope of MinMax Running Resettable

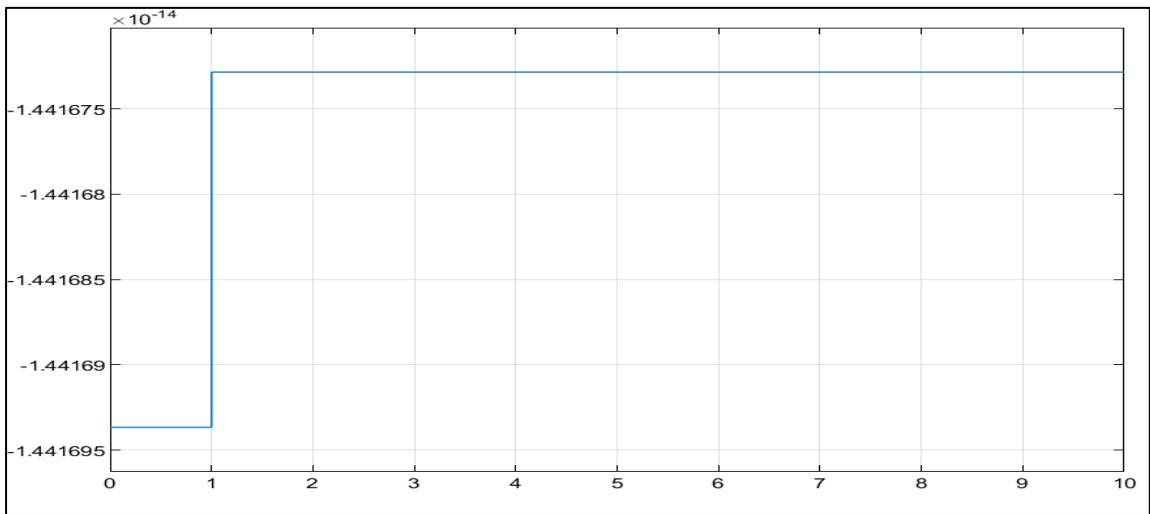


Figure 3: First Order Polynomial Evaluation of Step Function

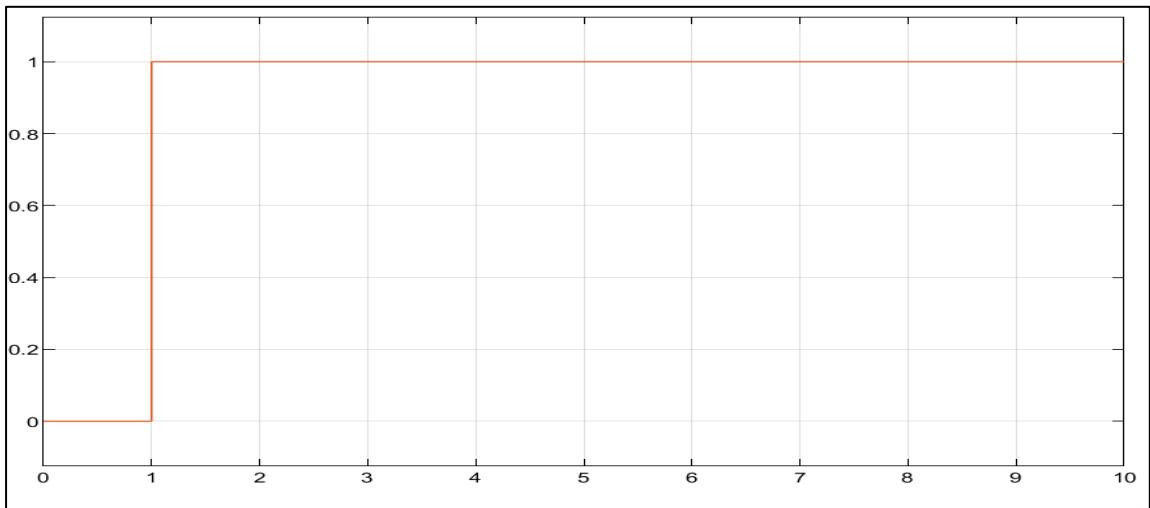


Figure 4: Matrix Concatenation of both signals

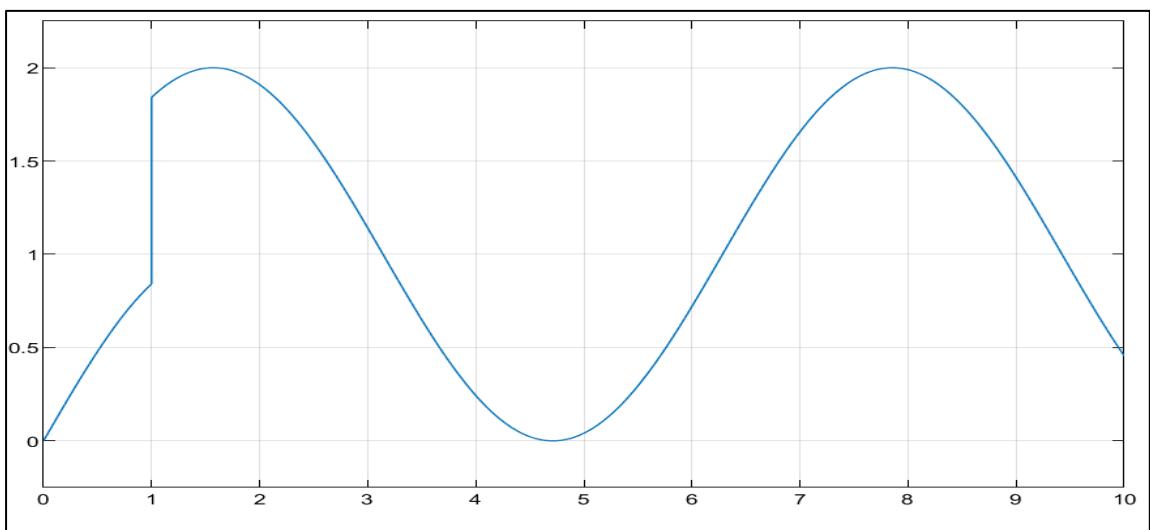


Figure 5: Addition of Step Function and Sine Wave

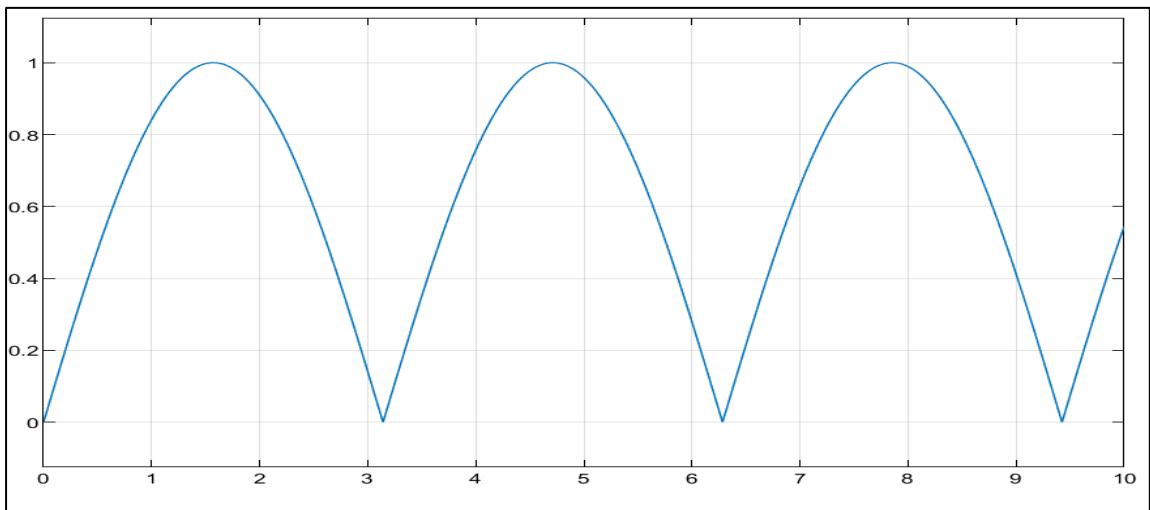


Figure 6: Absolute Values of Sine Wave

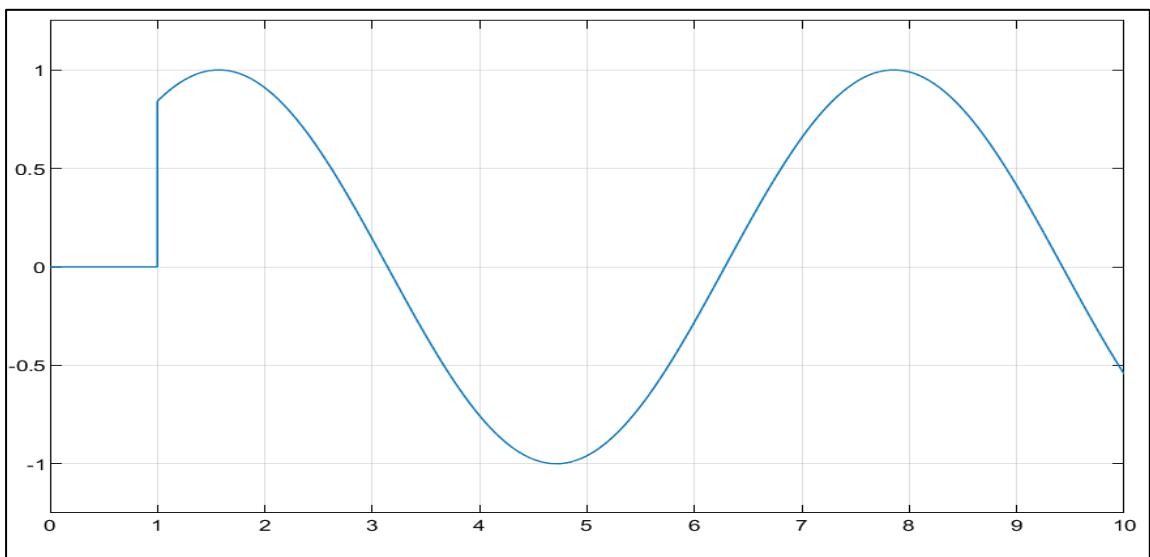


Figure 7: Dot Product of Sine Wave and Step Function

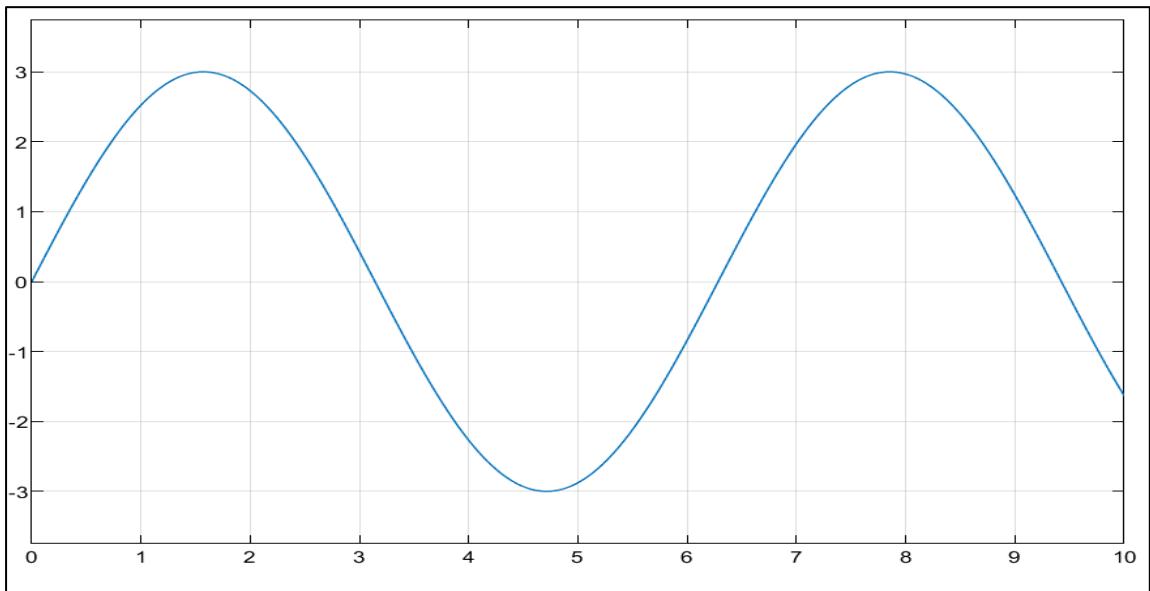


Figure 8: Sine Wave has a gain of 3

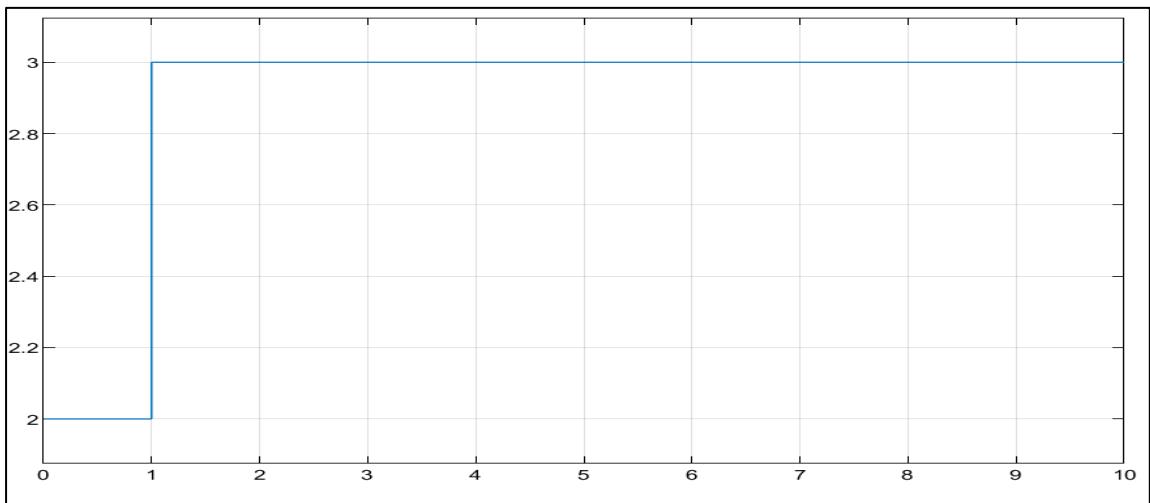


Figure 9: Step Function with Offset of 2

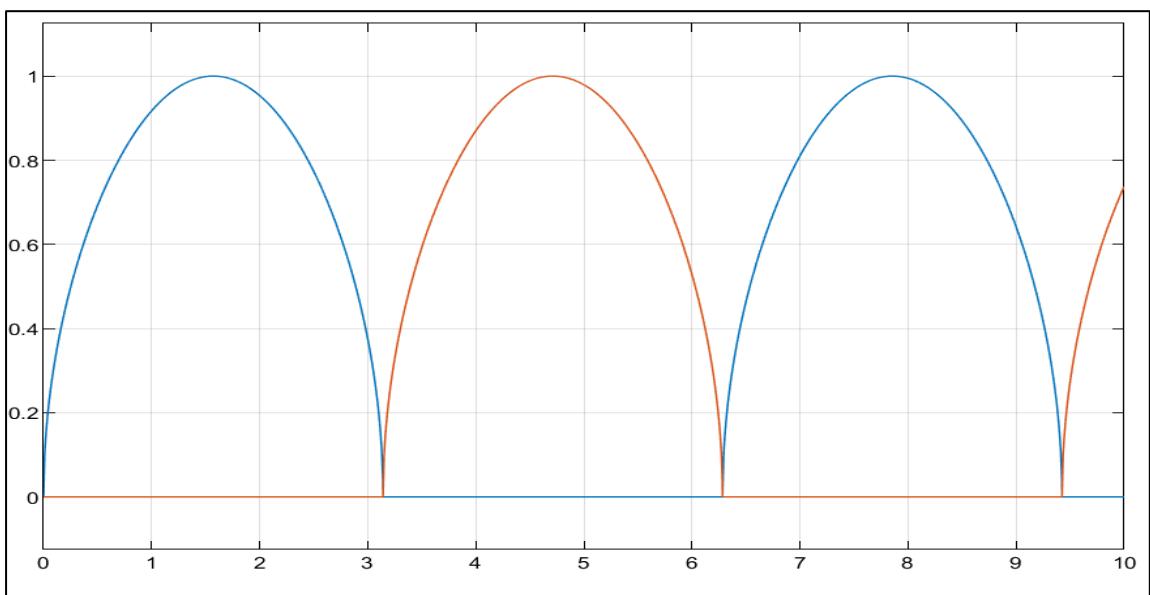


Figure 10: Square Root of Sine Wave

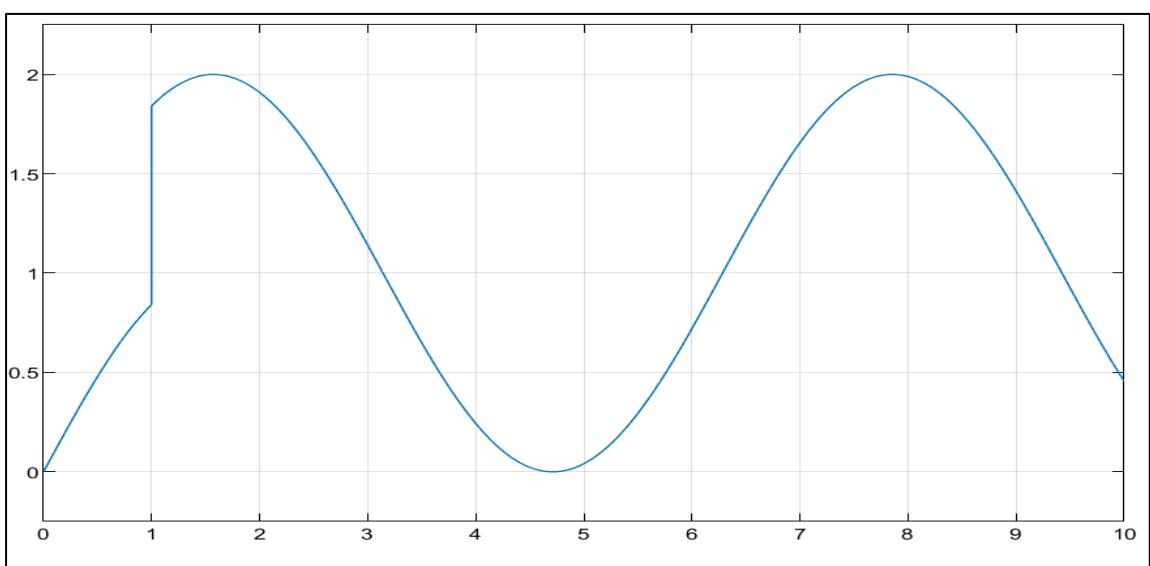


Figure 11: Sum of Step Function and Sine Wave

Task 2:

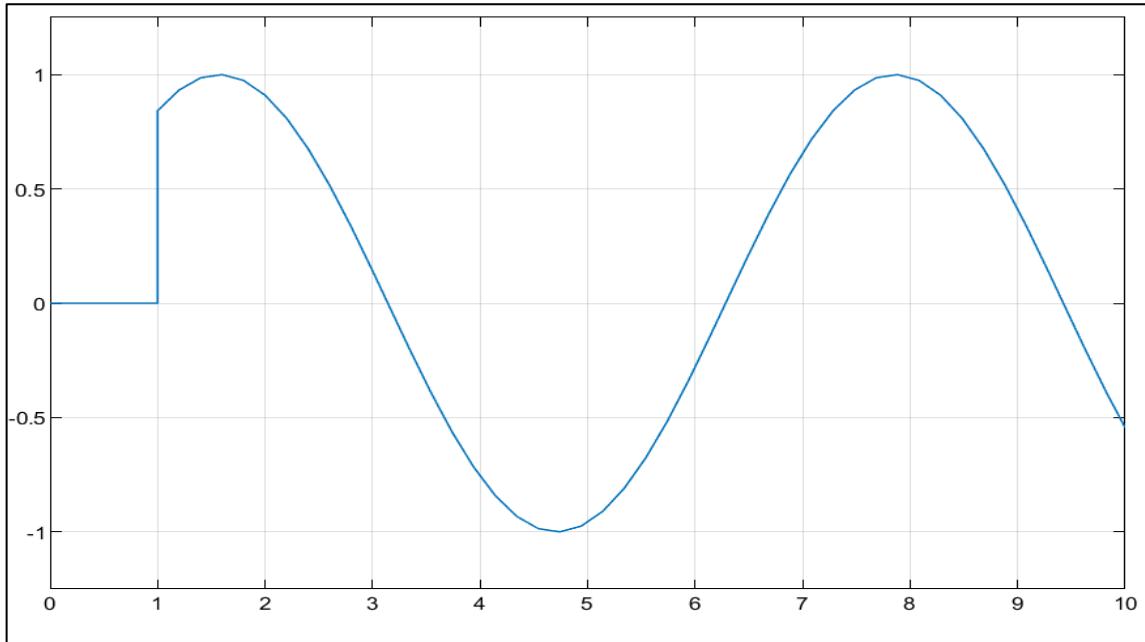


Figure 12: Product of Unit Step and Sine Wave

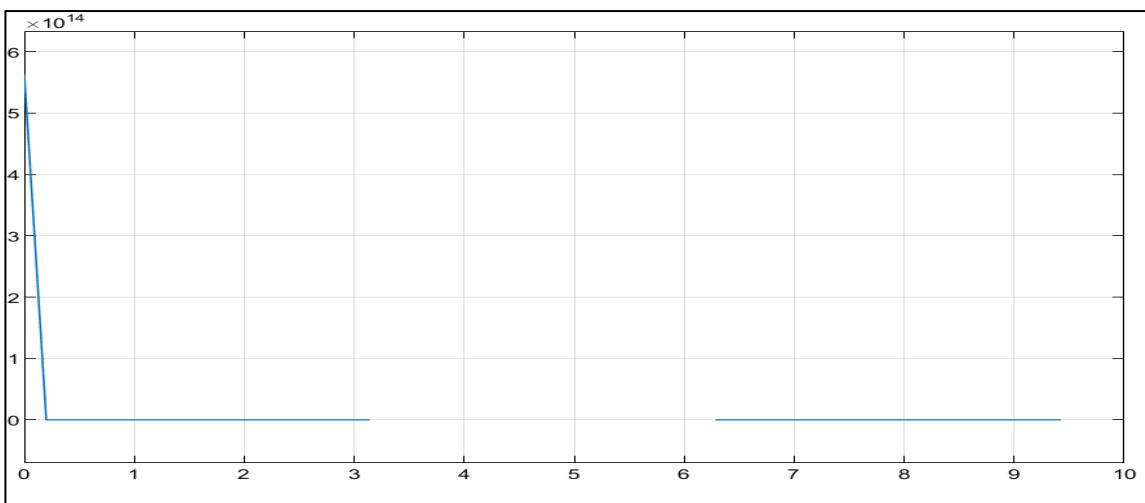


Figure 13: Reciprocal Square root of Sine Wave

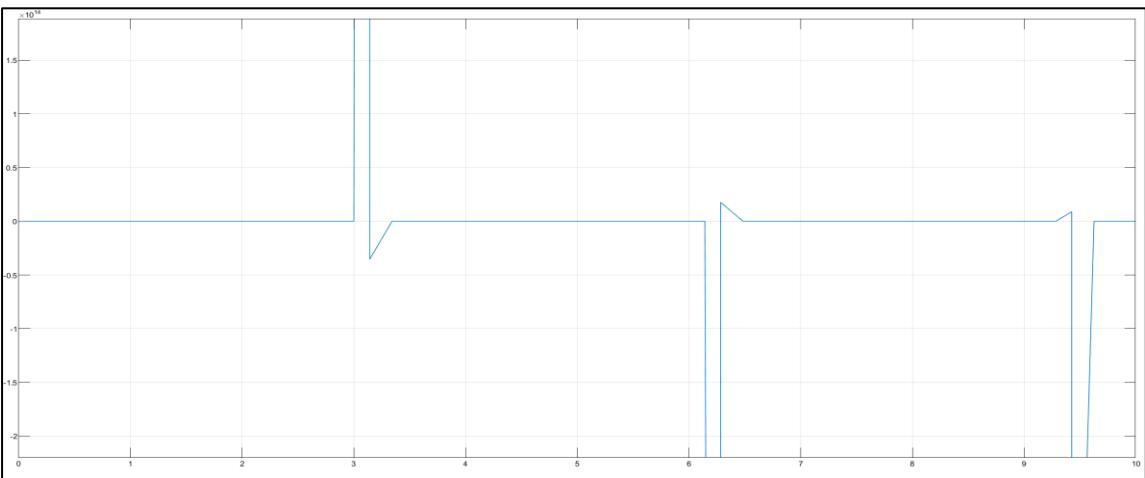


Figure 14: Product of Elements of Step Function and Sine Wave

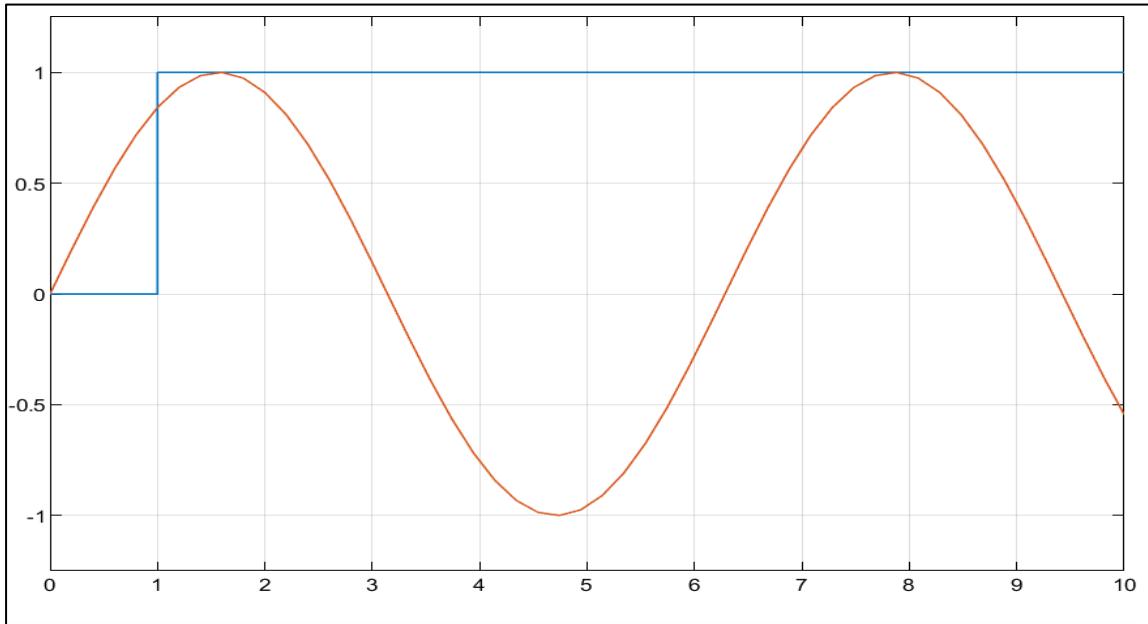


Figure 15: Vector Concatenate Unit Step and Sine Wave

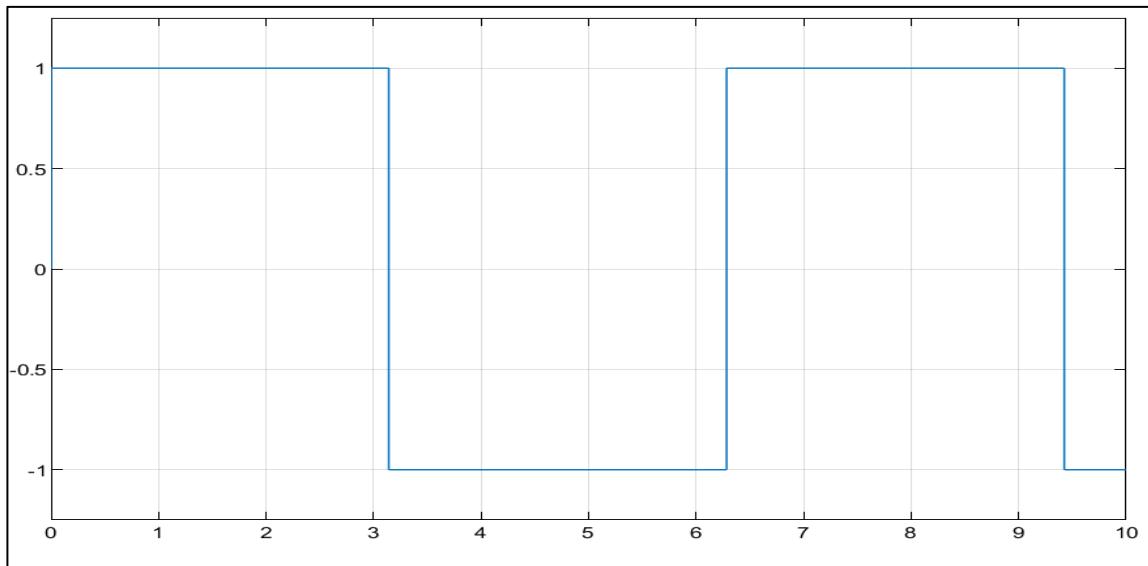


Figure 16: Signum Output of Sign Wave

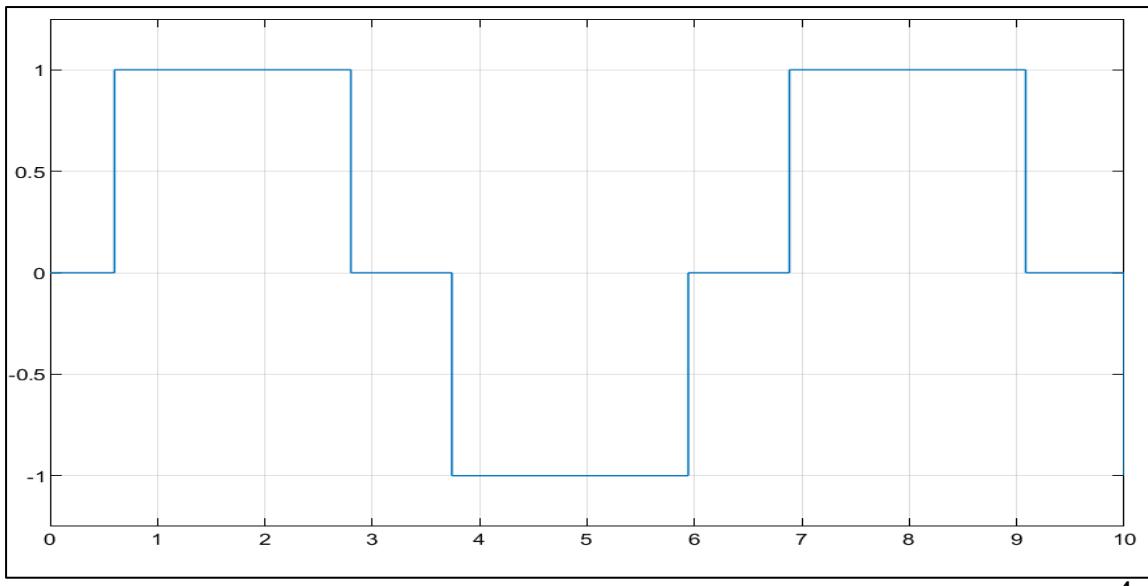


Figure 17: Round Operation on Sine Wave

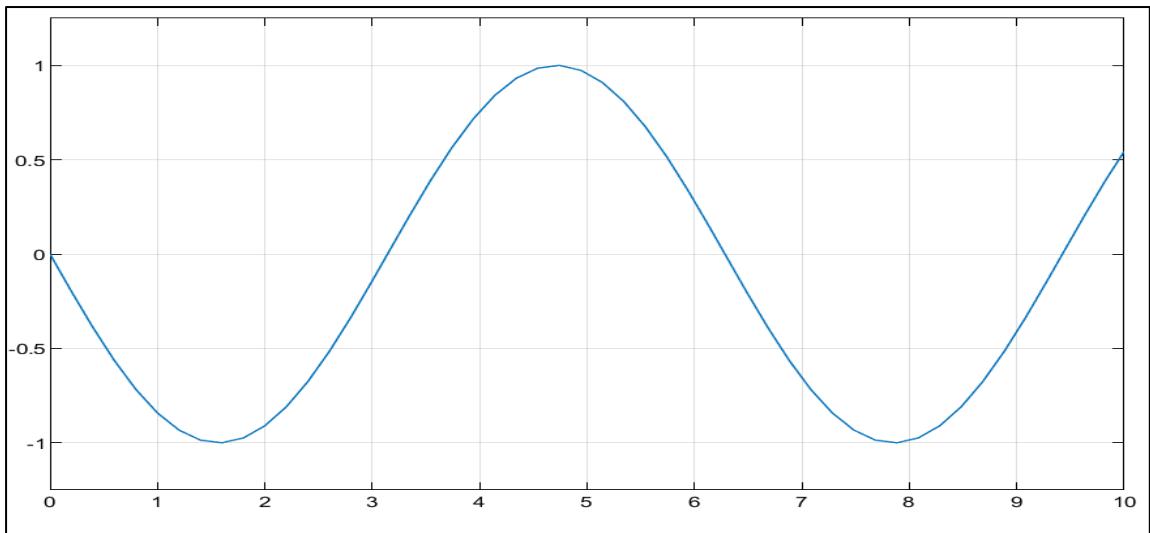


Figure 18: Unary Minus/Inverse of a Sine Wave

Task 3:

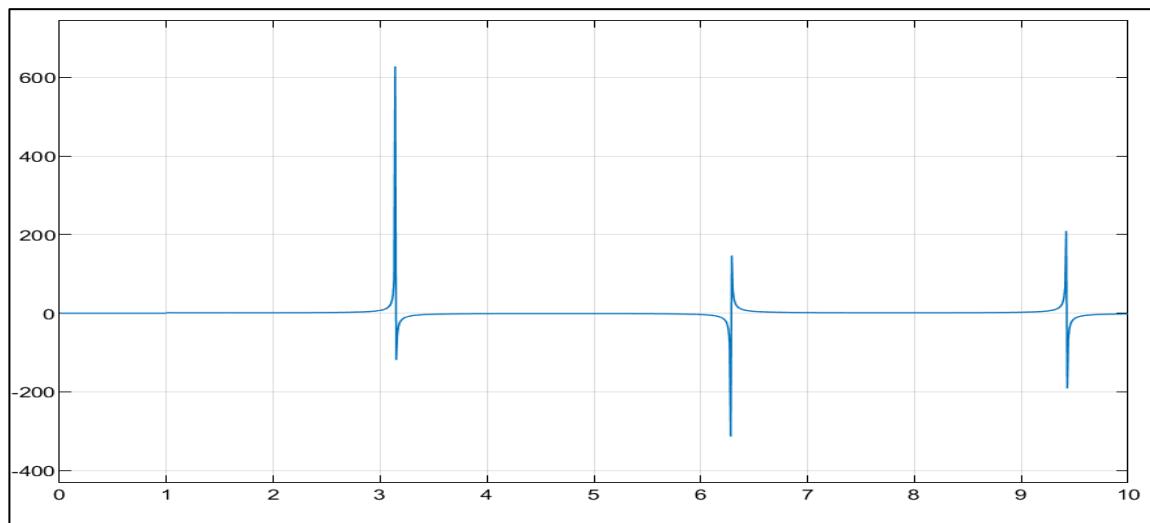


Figure 19: Element-wise Division between Unit Step and Sine Wave

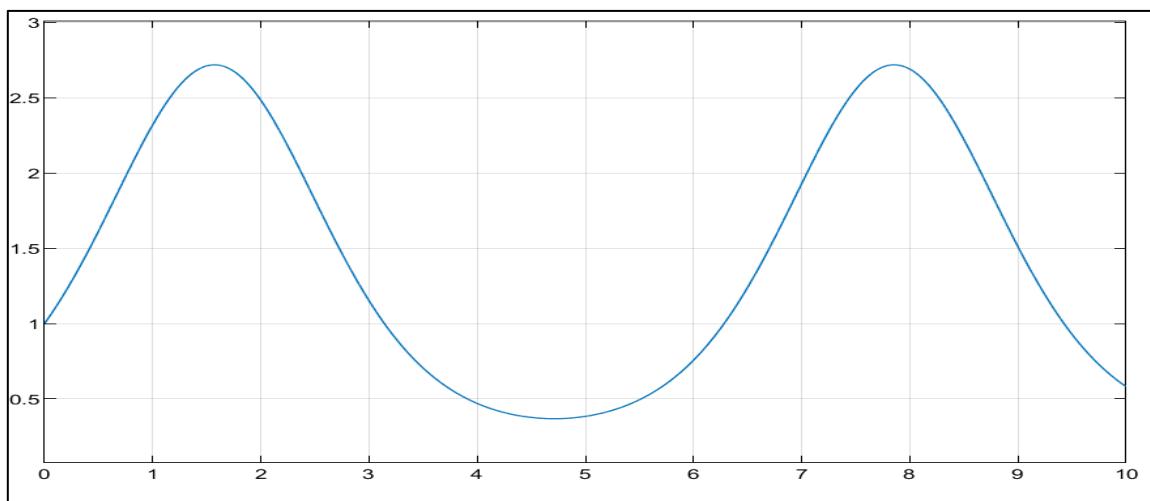


Figure 20: Exponential of Sine Wave

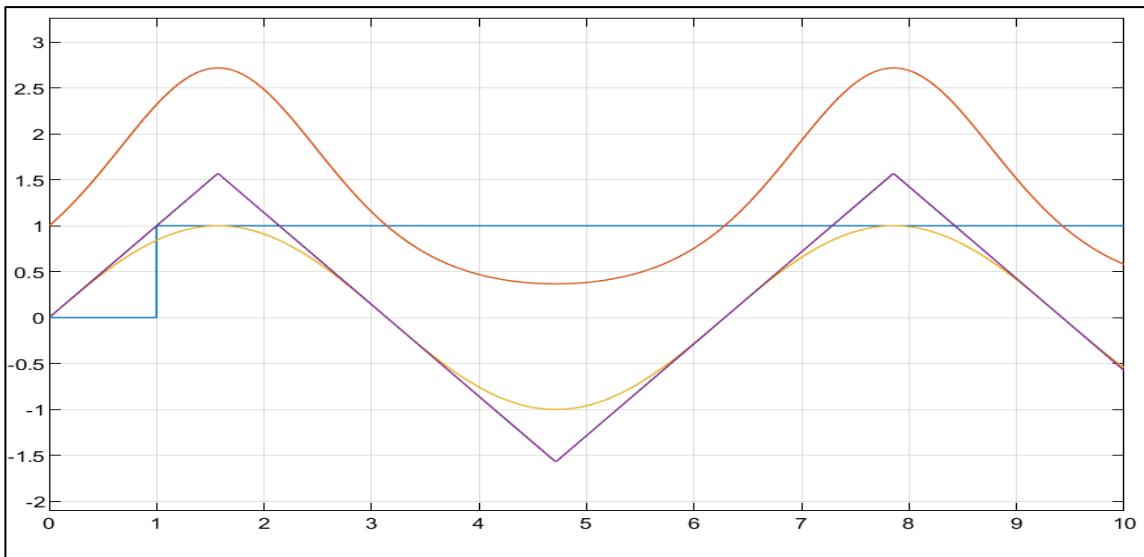


Figure 21: 4-in-1 MUX Sine Wave, Unit Step, Sine Exponential and asin of Sine Wave

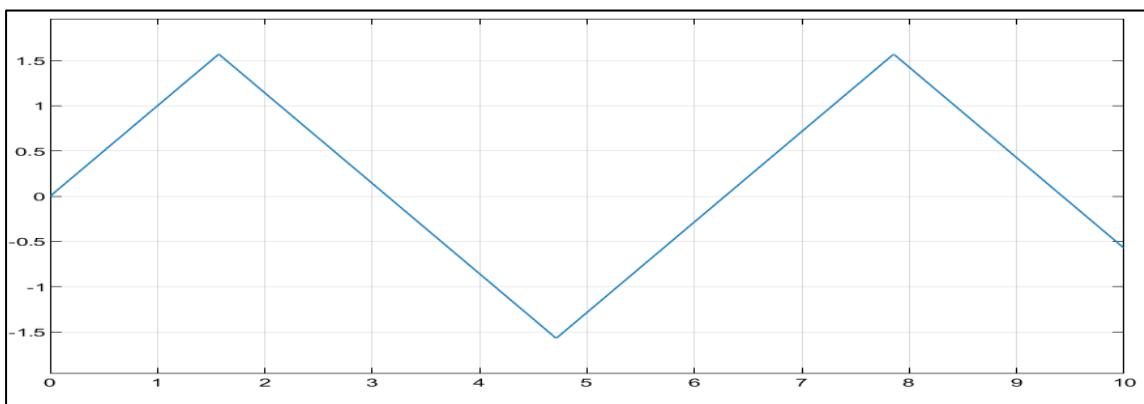


Figure 22: asin Operation on Sine Wave

```

data = out.sum;

% Example: Plot a specific slice (e.g., the first) of the data
slice = data(:,:,1); % Extract a 2x2 slice from the 3D data

figure; % Create a new figure
surf(slice); % Create a surface plot
title('2D Slice of "sum"');
xlabel('X');
ylabel('Y');
zlabel('Value');

```

Figure 23: Code to plot 3D vector variable named 'sum'

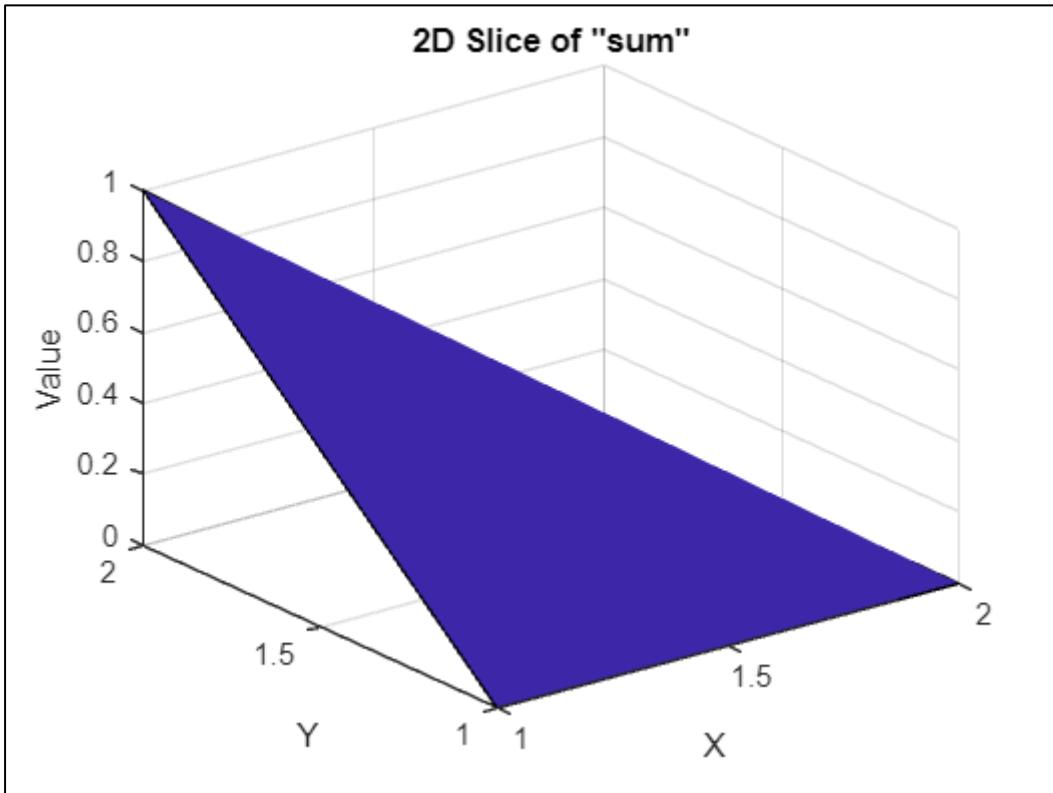


Figure 24: 2D Slice of 'sum'

Task 3 Description:

In this task, we did the operations of division, exponential and asin on both sine wave and the unit step. Additionally, we also send the unit step, sine wave, exponential and asin function into a 4x1 MUX. Afterwards, we saved the output of the MUX to the workspace and also created a variable named 'sum' after reshaping the output. Lastly, I plotted my output variable onto a 3D plot.

Conclusion:

In this lab, we had the opportunity to dive into Simulink, a valuable tool for modeling and simulating dynamic systems. It provided us with hands-on experience using basic Simulink blocks and understanding how to create block diagrams for various system simulations. We applied these blocks to simple signals like unit steps and sine waves and used the 'scope' block as an oscilloscope to visualize the results of our simulations. Overall, this lab not only introduced us to Simulink but also laid a solid foundation for our future studies and careers in system modeling and simulation.

Verified: _____

Date/Time: _____

Experiment No. 7

Objective:

The objective of this lab is to create a generalized code for convolution of two discrete signals and to perform convolution using Simulink. The students will also learn to use audio signals in MATLAB.

Theoretical Background:

Convolution is the representation of an LTI system in terms of its unit impulse response. Impulse response of a system $h[n]$ is the output when a unit impulse $\delta[n]$ is given at its input. The convolution of two discrete time signals is called the convolution sum while the convolution of two continuous time signals is referred to as the convolution integral.

Convolution of sequence $x[n]$ with the response of LTI system $h[n]$ is the convolution sum, given as:

$$y[n] = x[n] * h[n]$$

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

Tasks:

Task 1:

Write a MATLAB code for Convolution of the following signals, and plot the results for each case:

1. $x[n] = [0.5 \ 2]$, $h[n] = [1 \ 1 \ 1]$
2. $x[n] = 1, 0 \leq n \leq 4$
 $h[n] = a^n, 0 \leq n \leq 6$

Also check results using ‘conv’ command

Task 2:

- Record a 5 second sound signal using ‘audiorecorder’ and save it in a ‘.wav’ file.
- Read the ‘.wav’ file using ‘audioread’
- Plot the original sound signal.

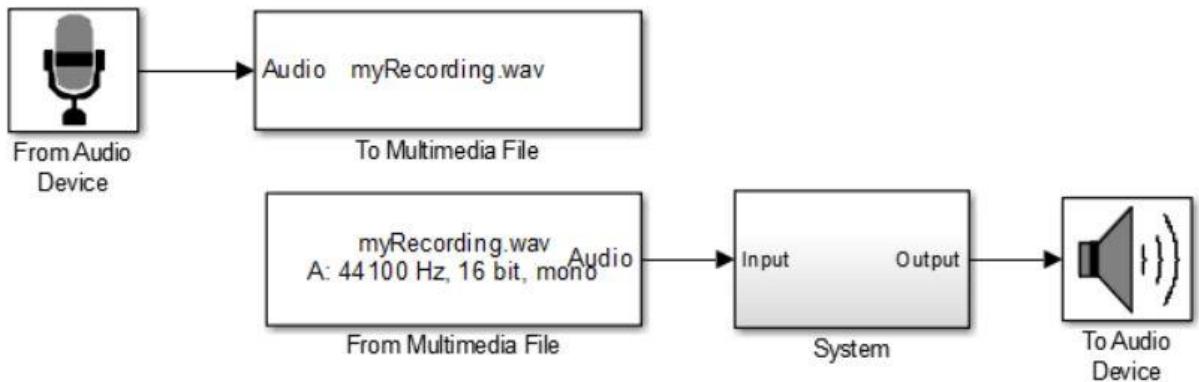
- Play the sound file in the following ways:
 - Complete file
 - First half and second half of the file separately
 - Middle one-third of the file

Task 3:

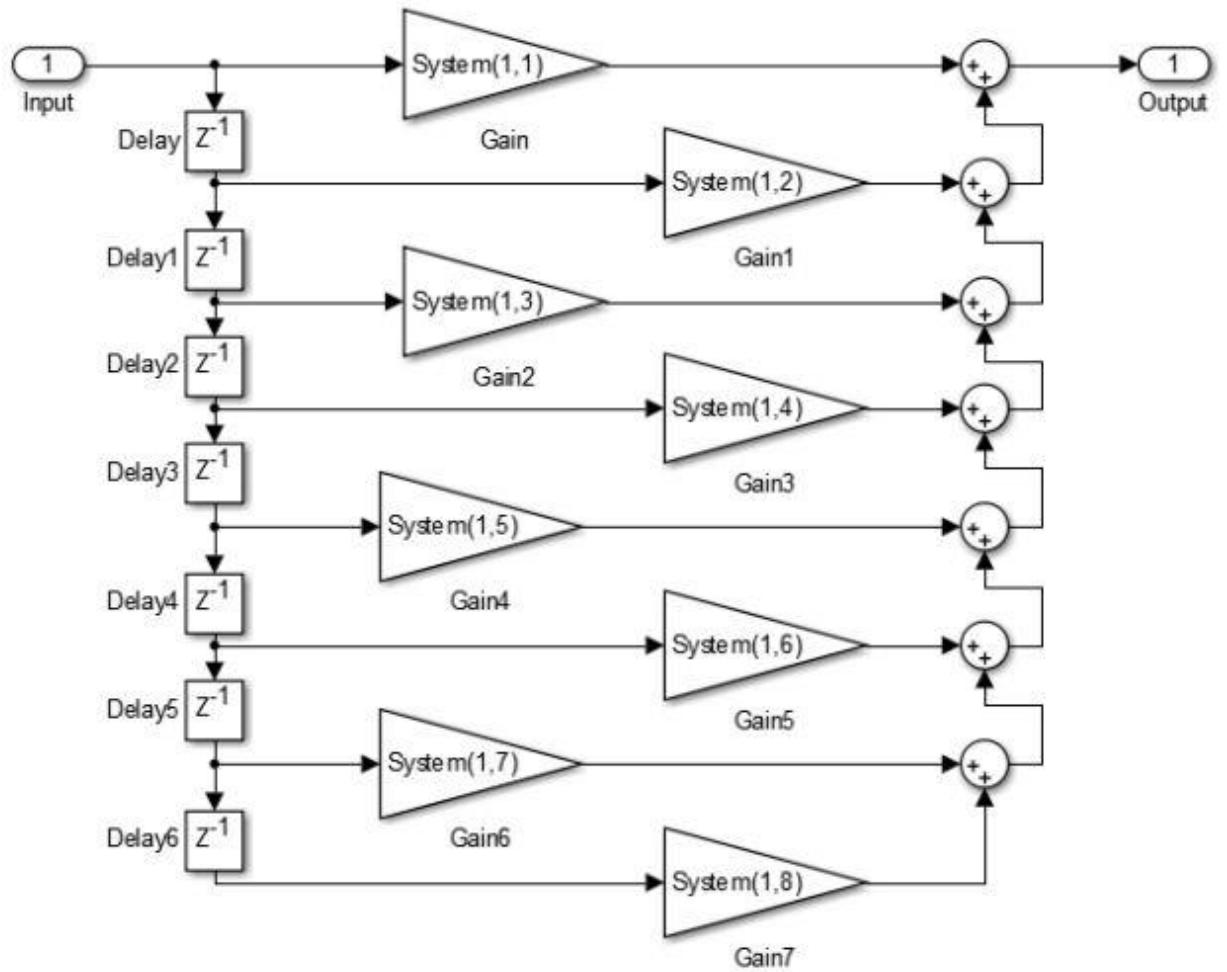
Perform convolution of an audio file imported in Simulink with a system defined below. Plot the audio signal before and after passing through the system using function callback (File -> Model properties -> Callbacks) The system is defined as:

- System= $10*[1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$

The block diagram to be created by students are:



The system block diagram is to be created as:



INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

Part (a)

```
% Define the first set of signals
x = [0.5 2];
h = [1 1 1];

m = length(x);
o = length(h);
k = m + o - 1;

% Initialize y_manual and y_conv as arrays of zeros
y_manual = zeros(1, k);

% Perform convolution manually using nested loops
for i = 1:k
    for j = 1:m
        if i - j + 1 > 0 && i - j + 1 <= o
            y_manual(i) = y_manual(i) + x(j) * h(i - j + 1);
        end
    end
end

y_manual

y_manual = 1x4
0.5000    2.5000    2.5000    2.0000

% Perform convolution using the built-in 'conv' function
y_conv = conv(x, h)

y_conv = 1x4
0.5000    2.5000    2.5000    2.0000

% Compare the results
isequal(y_manual, y_conv) % Check if the results are equal

ans = logical
1

n = 1:m;

figure;

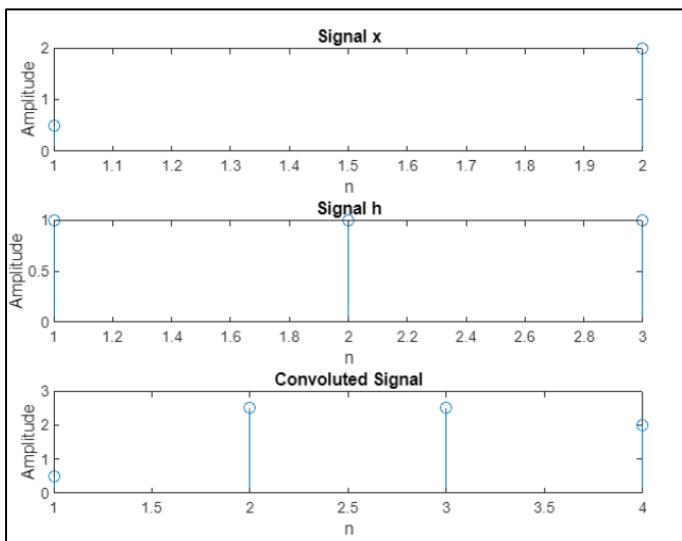
% Plot the signals and their convolution result
subplot(3, 1, 1);
stem(n, x);
title('Signal x');
xlabel('n');
ylabel('Amplitude');

n = 1:o;

subplot(3, 1, 2);
stem(n, h);
title('Signal h');
xlabel('n');
ylabel('Amplitude');

z = 1: k;

subplot(3, 1, 3);
stem(z, y_conv);
title('Convolved Signal');
xlabel('n');
ylabel('Amplitude');
```



Part (b)

```
% Define the second set of signals
n2 = 0:4;
x = ones(1, length(n2))

x = 1x5
    1      1      1      1      1

n3 = 0:6;

a = 0.5; % You can change the value of 'a' as needed
h = a.^n3

h = 1x7
    1.0000    0.5000    0.2500    0.1250    0.0625    0.0312    0.0156

m = length(x);
o = length(h);
k = m + o - 1;

% Initialize y_manual and y_conv as arrays of zeros
y_manual = zeros(1, k);

% Perform convolution manually using nested loops
for i = 1:k
    for j = 1:m
        if i - j + 1 > 0 && i - j + 1 <= o
            y_manual(i) = y_manual(i) + x(j) * h(i - j + 1);
        end
    end
end

y_manual

y_manual = 1x11
    1.0000    1.5000    1.7500    1.8750    1.9375    0.9688    0.4844    0.2344    ...

% Perform convolution using the built-in 'conv' function
y_conv = conv(x, h)

y_conv = 1x11
    1.0000    1.5000    1.7500    1.8750    1.9375    0.9688    0.4844    0.2344    ...

% Compare the results
isequal(y_manual, y_conv) % Check if the results are equal

ans = logical
    1
```

```

n = 1:m;

% Plot the signals and their convolution result
subplot(3, 1, 1);
stem(n, x);
title('Signal x');
xlabel('n');
ylabel('Amplitude');

n = 1:o;

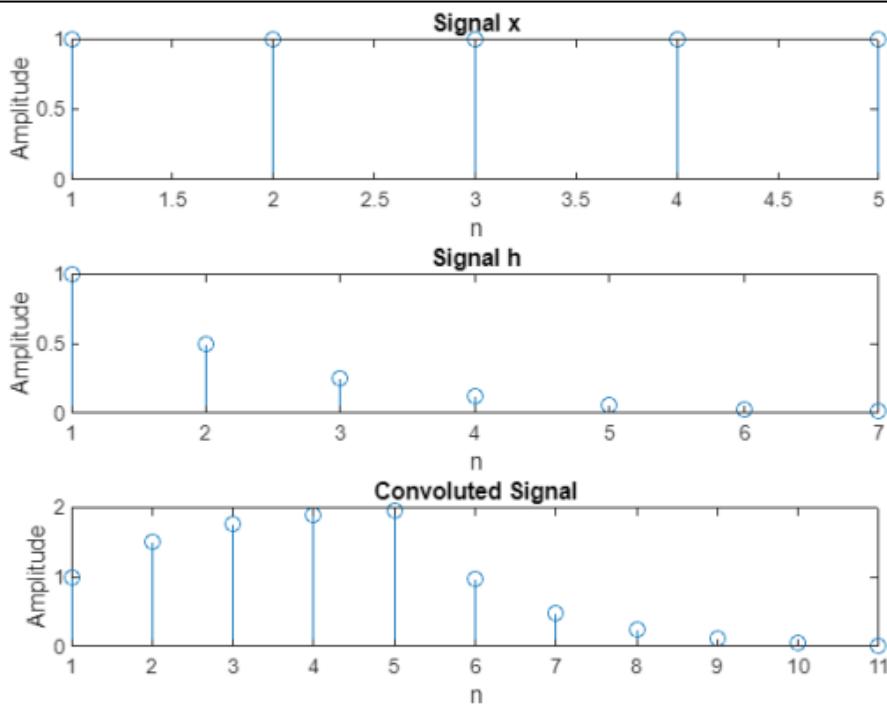
subplot(3, 1, 2);
stem(n, h);
title('Signal h');

xlabel('n');
ylabel('Amplitude');

z = 1: k;

subplot(3, 1, 3);
stem(z, y_conv);
title('Convolved Signal');
xlabel('n');
ylabel('Amplitude');

```



Task 2:

```

% Set the duration of the recording in seconds
duration = 5; % 5 seconds

% Create an audiorecorder object
recObj = audiorecorder(44100, 16, 1); % Sampling rate: 44.1 kHz, 16-bit, mono

% Record the audio for the specified duration
disp('Recording...');

Recording...

recordblocking(recObj, duration);
disp('Recording Finished.');

```

```
Recording Finished.

% Save the recorded audio as a .wav file
audiowrite('My_Recording.wav', getaudiodata(recObj), recObj.SampleRate);

% Read the .wav file
[y, fs] = audioread('My_Recording.wav');
% Play the sound file in different ways

% Play the complete file
sound(y, fs);
pause(duration); % Wait for the sound to finish

% Play the first half and second half of the file separately
half_duration = duration / 2;
sound(y(1:round(half_duration * fs)), fs);
pause(half_duration); % Wait for the first half to finish
sound(y(round(half_duration * fs) + 1:end), fs);
pause(half_duration); % Wait for the second half to finish

% Play the middle one-third of the file
third_duration = duration / 3;
start_index = round((duration - third_duration) / 2 * fs);
end_index = start_index + round(third_duration * fs);
sound(y(start_index:end_index), fs);
```

```

% Plot the original sound signal
figure;
subplot(4,1,1);
plot((1:length(y)) / fs, y);
title('Original Sound Signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Plot the first half and second half of the file separately
half_duration = duration / 2;
subplot(4,1,2);
plot((1:round(half_duration * fs)) / fs, y(1:round(half_duration * fs)));
title('First Half of the Sound Signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(4,1,3);
plot((round(half_duration * fs) + 1:length(y)) / fs, y(round(half_duration * fs) +
1:end));
title('Second Half of the Sound Signal');
xlabel('Time (s)');

```

ylabel('Amplitude');

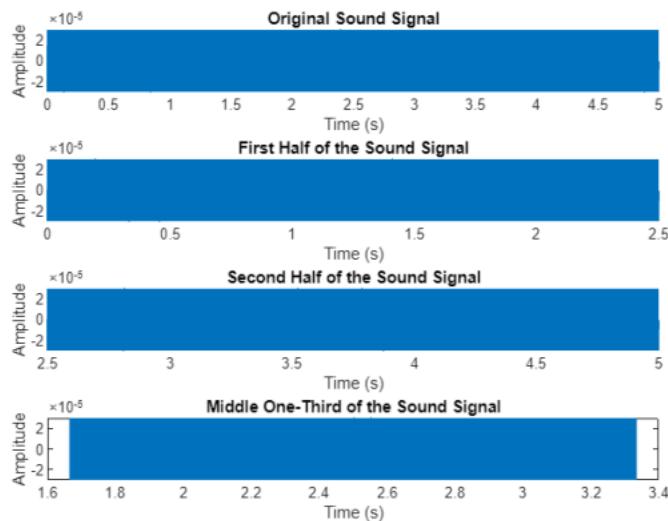
```

% Plot the middle one-third of the file
third_duration = duration / 3;
start_index = round((duration - third_duration) / 2 * fs);
end_index = start_index + round(third_duration * fs);

subplot(4,1,4);
plot((start_index:end_index) / fs, y(start_index:end_index));
title('Middle One-Third of the Sound Signal');
xlabel('Time (s)');

```

ylabel('Amplitude');



Conclusion:

In conclusion, Experiment No. 7 provided a comprehensive hands-on experience with convolution and audio signal processing in both MATLAB and Simulink. Throughout the tasks, we gained a deep understanding of the convolution process, honing our skills in applying it to specific signals and verifying results. Additionally, the lab introduced us to audio signal recording, playback, and visualization, enhancing our practical expertise in sound data manipulation. The integration of Simulink in Task 3 allowed us to extend our knowledge to real-world audio applications. This practical exposure and theoretical understanding have collectively enriched our skill set, preparing us for more advanced tasks in dynamic systems, audio processing, and beyond.

Verified: _____

Date/Time: _____

Experiment No. 8

Objective:

The aim of Today's lab is to introduce the students to create Graphical User Interfaces (GUIs) in MATLAB. By the end of this lab the students should be able to display information/instructions to user and accepting user inputs from keyboards and deal with GUIs for performing basic functions.

What Is GUI?

A graphical user interface (GUI) is a graphical display that contains devices, or components, that enable a user to perform interactive tasks. To perform these tasks, the user of the GUI does not have to create a script or type commands at the command line. Often, the user does not have to know the details of the task at hand. The GUI components can be menus, toolbars, push buttons, radio buttons, list boxes, and sliders. In MATLAB, a GUI can also display data in tabular form or as plots, and can group related components.

How Does a GUI Work?

Each component, and the GUI itself, is associated with one or more user-written routines known as callbacks. The execution of each callback is triggered by a particular user action such as a button push, mouse click, selection of a menu item, or the cursor passing over a component. This kind of programming is often referred to as event-driven programming. In event-driven programming, callback execution is asynchronous, controlled by events external to the software. In the case of MATLAB GUIs, these events usually take the form of user interactions with the GUI.

Ways to Build MATLAB GUIs

A MATLAB GUI is a figure window to which you add user-operated controls. You can select, size, and position these components as you like. Using callbacks you can make the components do what you want when the user clicks or manipulates them with keystrokes.

You can build MATLAB GUIs in two ways:

- Use GUIDE (GUI Development Environment), an interactive GUI construction kit.
- Create M-files that generate GUIs as functions or scripts (programmatic GUI construction).

The first approach starts with a figure that you populate with components from within a graphic layout editor. GUIDE creates an associated M-file containing callbacks for the GUI and its components. GUIDE saves both the figure (as a FIG-file) and the M-file. Opening either one also opens the other to run the GUI.

In the second, *programmatic*, GUI-building approach, you code an M-file that defines all component properties and behaviors; when a user executes the M-file, it creates a figure,

populates it with components, and handles user interactions. The figure is not normally saved between sessions because the M-file creates a new one each time it runs.

As a result, the M-files of the two approaches look different. Programmatic M-files are generally longer, because they explicitly define every property of the figure and its controls, as well as the callbacks. GUIDE GUIs define most of the properties within the figure itself. They store the definitions in its FIG-file rather than in its M-file. The M-file contains callbacks and other functions that initialize the GUI when it opens.

MATLAB software also provides functions that simplify the creation of standard dialog boxes, for example to issue warnings or to open and save files. The GUI-building technique you choose depends on your experience, your preferences, and the kind of application you need the GUI to operate.

You can combine the two approaches to some degree. You can create a GUI with GUIDE and then modify it programmatically. However, you cannot create a GUI programmatically and later modify it with GUIDE.

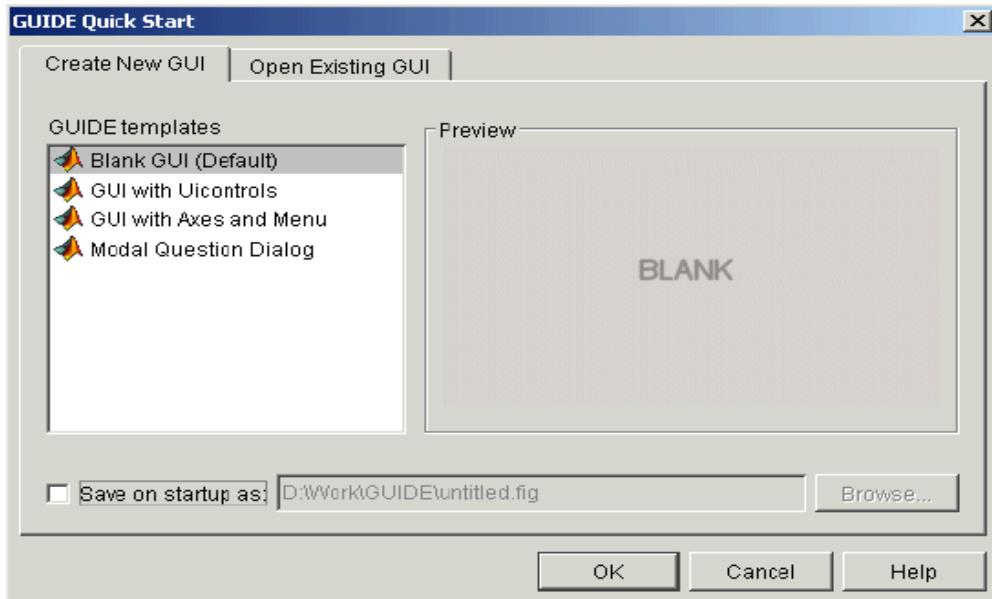
Starting GUIDE

There are many ways to start GUIDE. You can start GUIDE from the:

- Command line by typing guide
- Start menu by selecting **MATLAB > GUIDE (GUI Builder)**
- MATLAB File** menu by selecting **New > GUI**
- MATLAB toolbar by clicking the **GUIDE** button

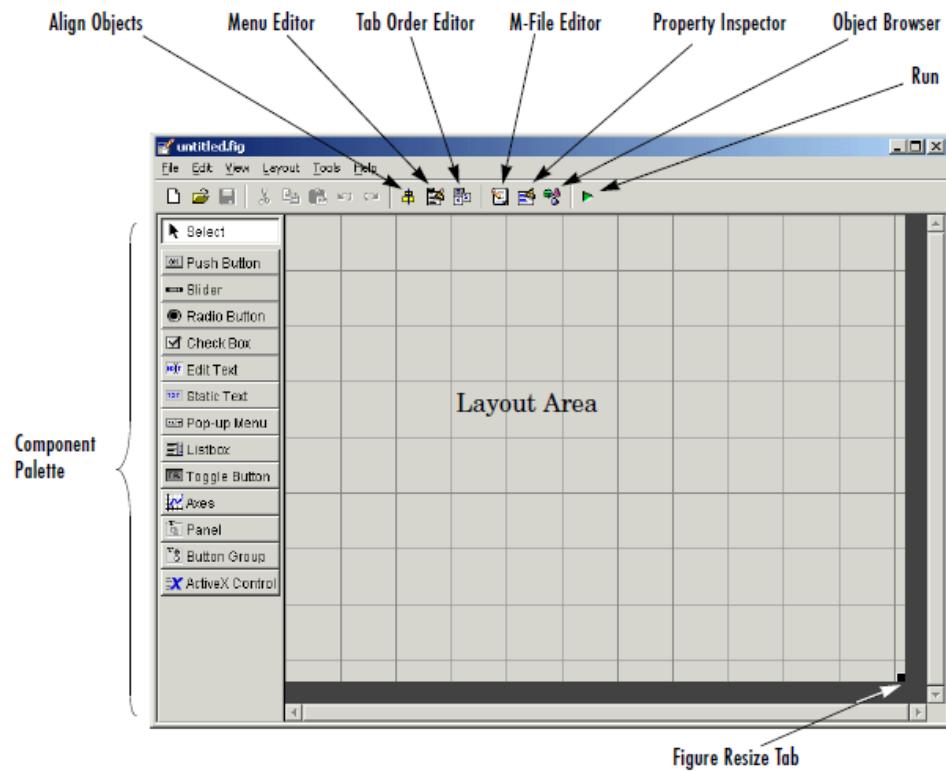


However you start GUIDE, it displays the GUIDE Quick Start dialog box shown in the following figure.



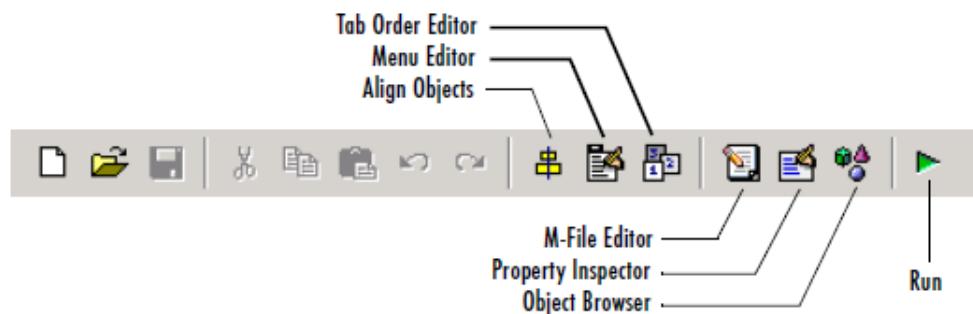
GUIDE Tools Summary

The GUIDE tools are available from the Layout Editor shown in the figure below. The tools are called out in the figure and described briefly below.



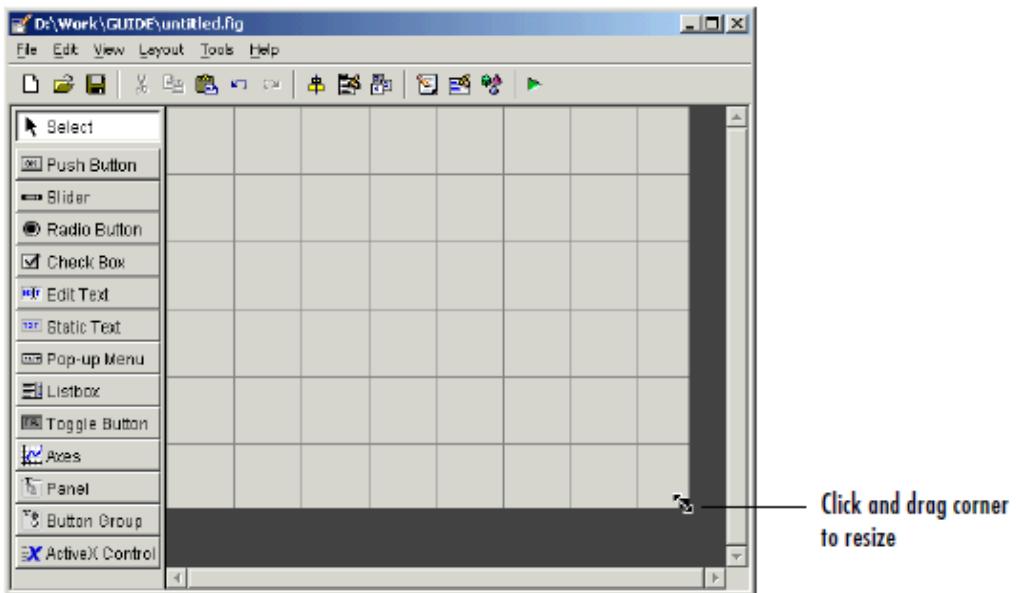
Show Toolbar

Displays the following toolbar in the Layout Editor window.



Setting the GUI Size

Set the size of the GUI by resizing the grid area in the Layout Editor. Click the lower-right corner and drag it until the GUI is the desired size. If necessary, make the window larger.



Available Components

The component palette at the left side of the Layout Editor contains the components that you can add to your GUI. You can display it with or without names.

Component	Icon	Description
Push Button		Push buttons generate an action when clicked. For example, an OK button might apply settings and close a dialog box. When you click a push button, it appears depressed; when you release
Toggle Button		Toggle buttons generate an action and indicate whether they are turned on or off. When you click a toggle button, it appears depressed, showing that it is on. When you release the mouse button, the toggle button remains depressed until you click it a second time. <u>When you do so, the button returns to the raised</u>
Radio Button		Radio buttons are similar to check boxes, but radio buttons are typically mutually exclusive within a group of related radio buttons. That is, when you select one button the previously selected button is deselected. To activate a radio button, click the <u>mouse button on the object. The display indicates the state of</u>
Check Box		Check boxes can generate an action when checked and indicate their state as checked or not checked. Check boxes are useful when providing the user with a number of independent choices, for example, displaying a toolbar.

Edit Text		Edit text components are fields that enable users to enter or modify text strings. Use edit text when you want text as input. Users can enter numbers but you must convert them to their numeric equivalents.
Static Text		Static text controls display lines of text. Static text is typically used to label other controls, provide directions to the user, or indicate values associated with a slider. Users cannot change static text interactively.
Slider		Sliders accept numeric input within a specified range by enabling the user to move a sliding bar, which is called a slider or thumb. Users move the slider by clicking the slider and dragging it, by clicking in the trough, or by clicking an arrow. The location of the slider indicates the relative location within the specified range.
List Box		List boxes display a list of items and enable users to select one or more items.
Pop-Up Menu		Pop-up menus open to display a list of choices when users click the arrow.
Axes		Axes enable your GUI to display graphics such as graphs and images. Like all graphics objects, axes have properties that you can set to control many aspects of its behavior and appearance. See “Axes Properties” in the MATLAB Graphics documentation and commands such as the following for more information on axes objects: plot, surf, line, bar, polar, pie, contour, and mesh. See Functions — By Category in the MATLAB documentation for a complete list.
Panel		Panels arrange GUI components into groups. By visually grouping related controls, panels can make the user interface easier to understand. A panel can have a title and various borders. Panel children can be user interface controls and axes as well as button groups and other panels. The position of each
Button Group		Button groups are like panels but are used to manage exclusive selection behavior for radio buttons and toggle buttons.

Callbacks: An Overview

After you have layed out your GUI, you need to program its behavior. The code you write controls how the GUI responds to events such as button clicks, slider movement, menu item selection, or the creation and deletion of components. This programming takes the form of a set of functions, called callbacks, for each component and for the GUI figure itself.

What Is a Callback?

A callback is a function that you write and associate with a specific GUI component or with the GUI figure. It controls GUI or component behavior by performing some action in response to an event for its component. This kind of programming is often called event-driven programming. When an event occurs for a component, MATLAB invokes the component's callback that is triggered by that event. As an example, suppose a GUI has a button that triggers the plotting of some data. When the user clicks the button, MATLAB calls the callback you associated with clicking that button, and the callback, which you have programmed, then gets the data and plots it. A component can be any control device such as a push button, list box, or slider. For purposes of programming, it can also be a menu or a container such as a panel or button group.

M-Files and FIG-Files

By default, the first time you save or run a GUI, GUIDE stores the GUI in two files:

- **A FIG-file**, with extension .fig, that contains a complete description of the GUI layout and the GUI components, such as push buttons, axes, panels, menus, and so on. The FIG-file is a binary file and you cannot modify it except by changing the layout in GUIDE.
- **An M-file**, with extension .m, that initially contains initialization code and templates for some callbacks that are needed to control GUI behavior. You must add the callbacks you write for your GUI components to this file. When you save your GUI the first time, GUIDE automatically opens the

M-file in your default editor. The FIG-file and the M-file, usually reside in the same directory. They correspond to the tasks of laying out and programming the GUI. When you lay out the GUI in the Layout Editor, your work is stored in the FIG-file. When you program the GUI, your work is stored in the corresponding M-file.

GUI M-File Structure

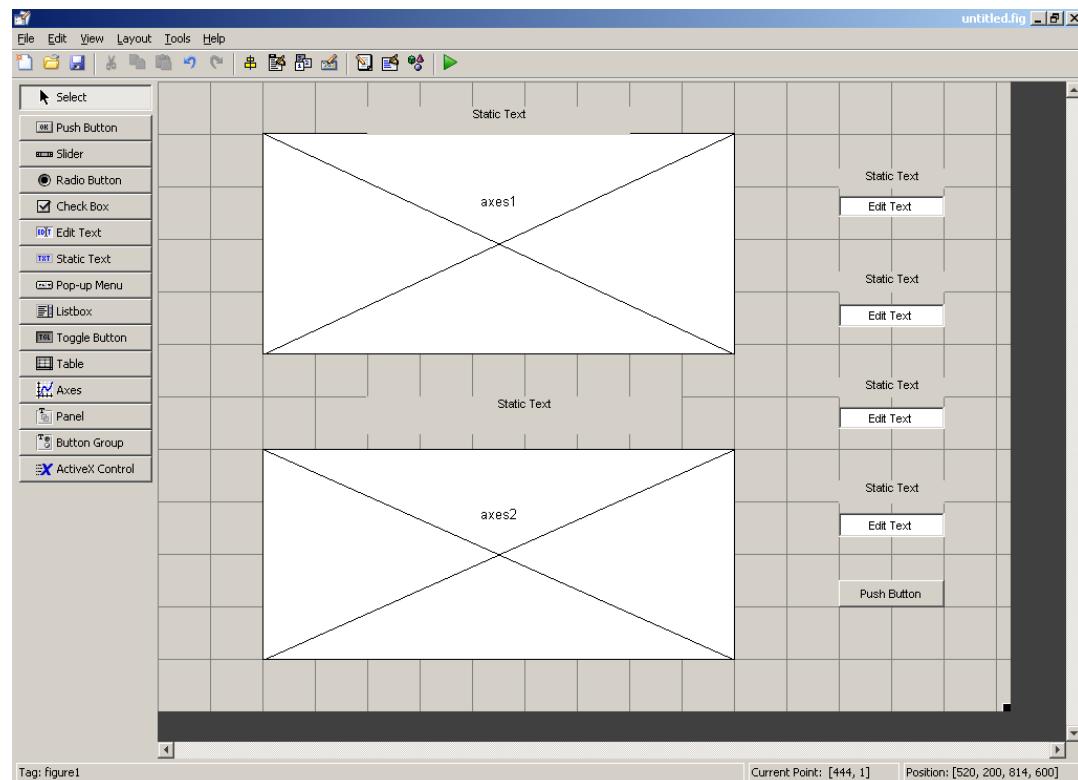
The GUI M-file that GUIDE generates is a function file. The name of the main function is the same as the name of the M-file. For example, if the name of the M-file is mygui.m, then the name of the main function is mygui. Each callback in the file is a subfunction of the main function. When GUIDE generates an M-file, it automatically includes templates for the most commonly used callbacks for each component. The M-file also contains initialization code, as well as an opening function callback and an output function callback. You must add code to the component callbacks for your GUI to work as you want. You may also want to add code to the opening function callback and the output function callback. The major sections of the GUI M-file are ordered as shown in the following table.

Task 1

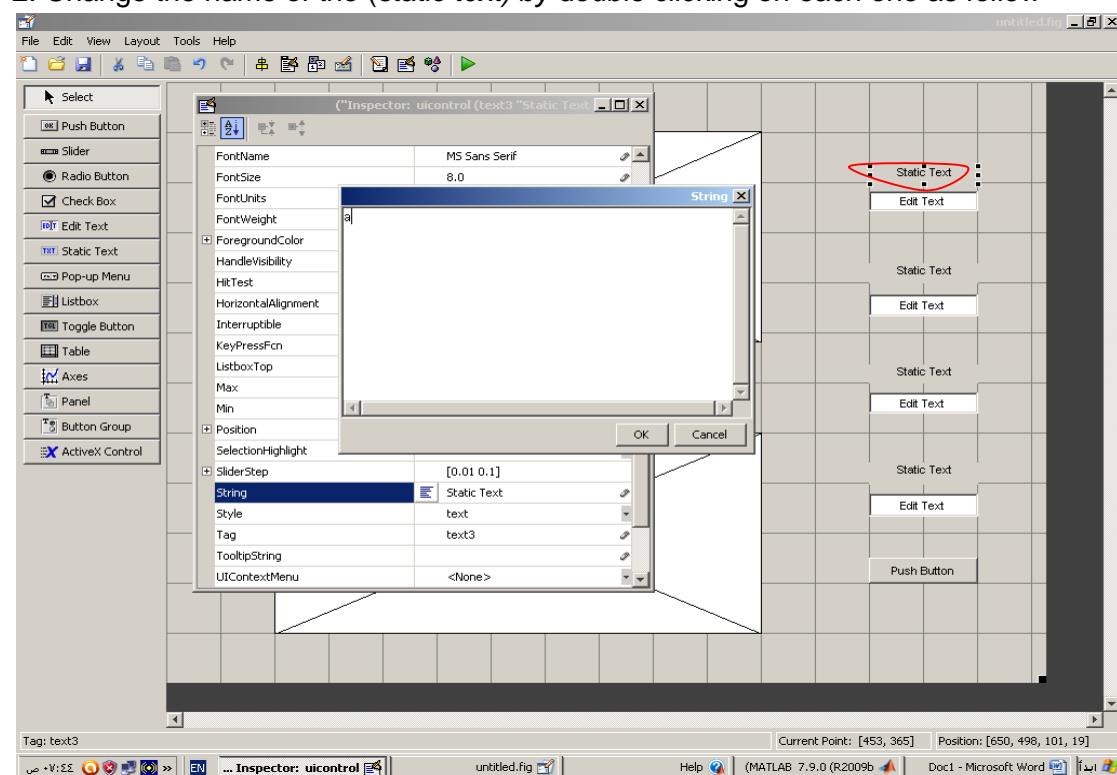
plot a function $f(t)=t*[u(t+1)-u(t-1)]+u(t-1)-u(t-3)$ and plot a time shifted and time scaled version of f(t) which has the general form $cf(at+b)$. The user can input variables values of a,b and c. The original function appears on GUI axis1 and the other on GUI axis2.

Designing steps

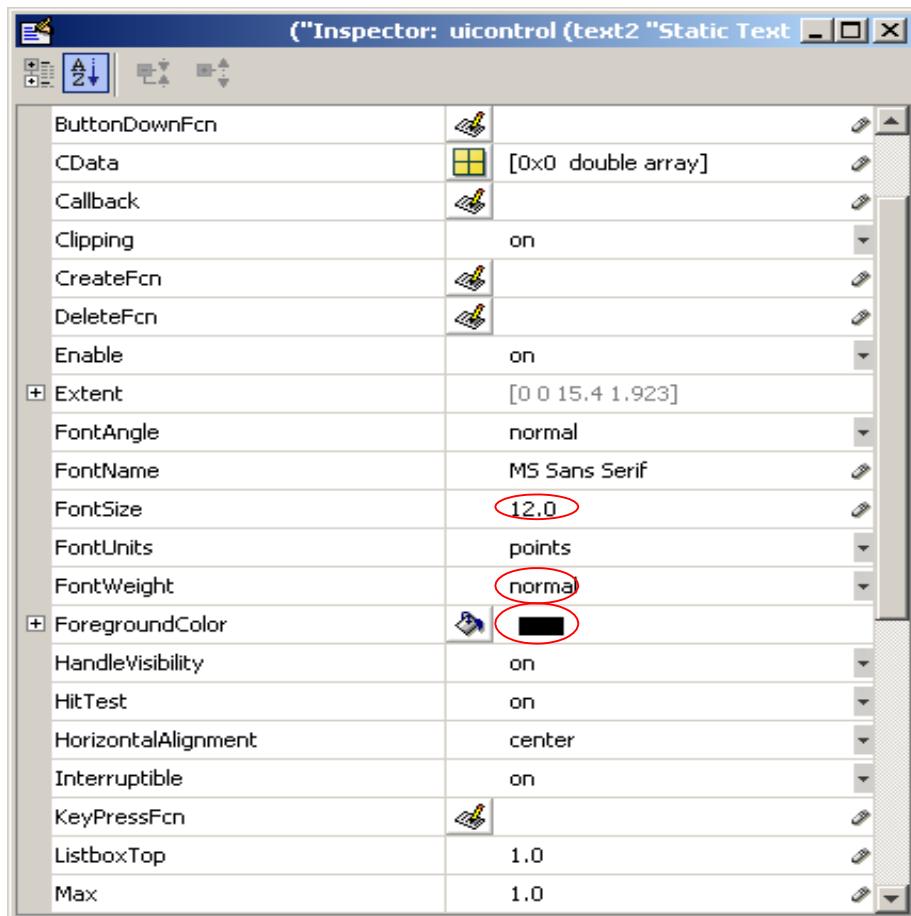
1-Put the following components in the figure.



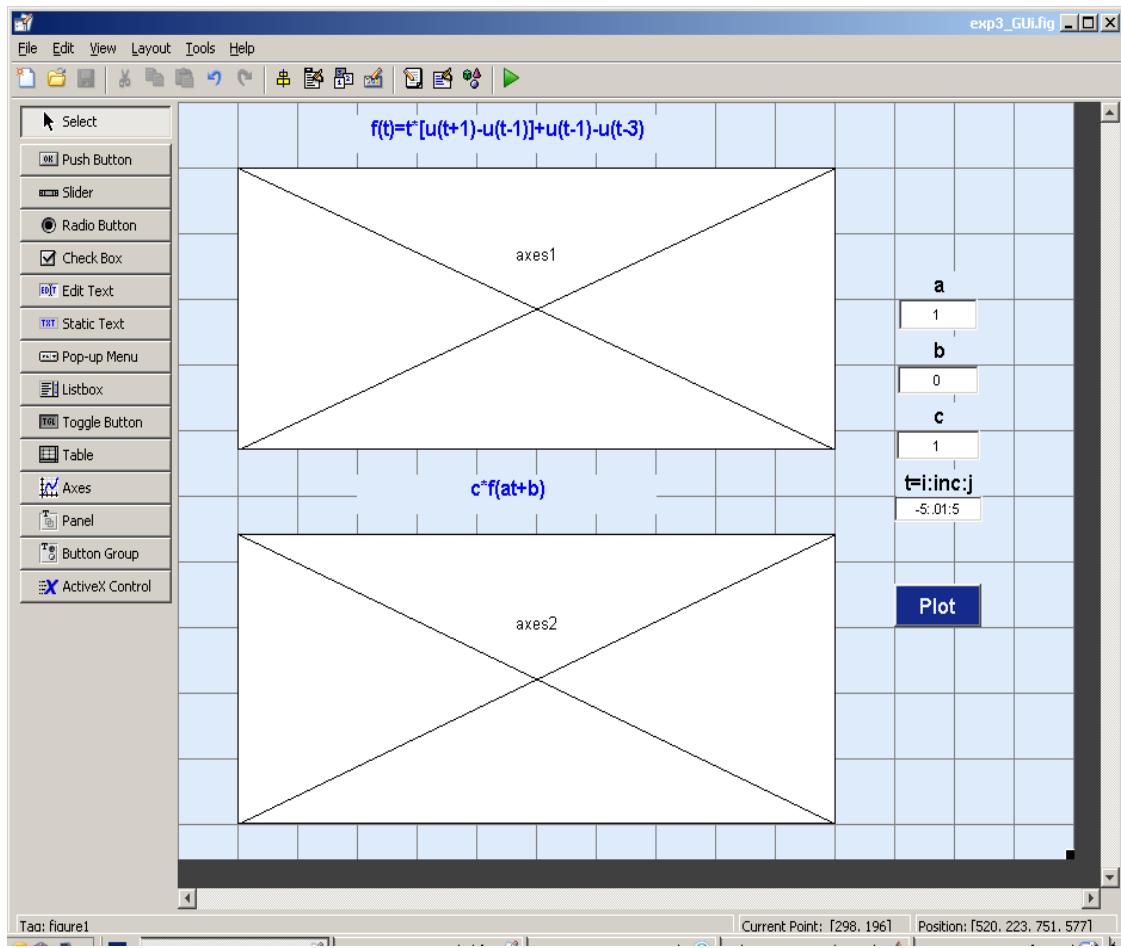
2. Change the name of the (*static text*) by double clicking on each one as follow



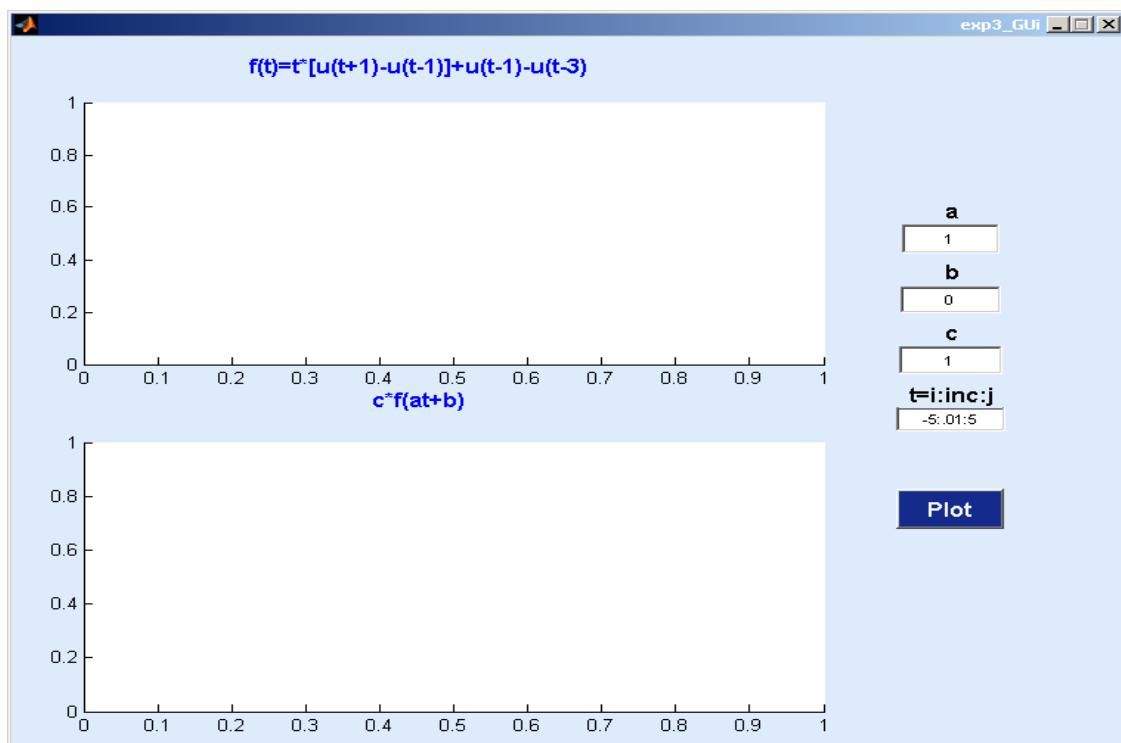
3-change the size ,colour and weight of the text as follow



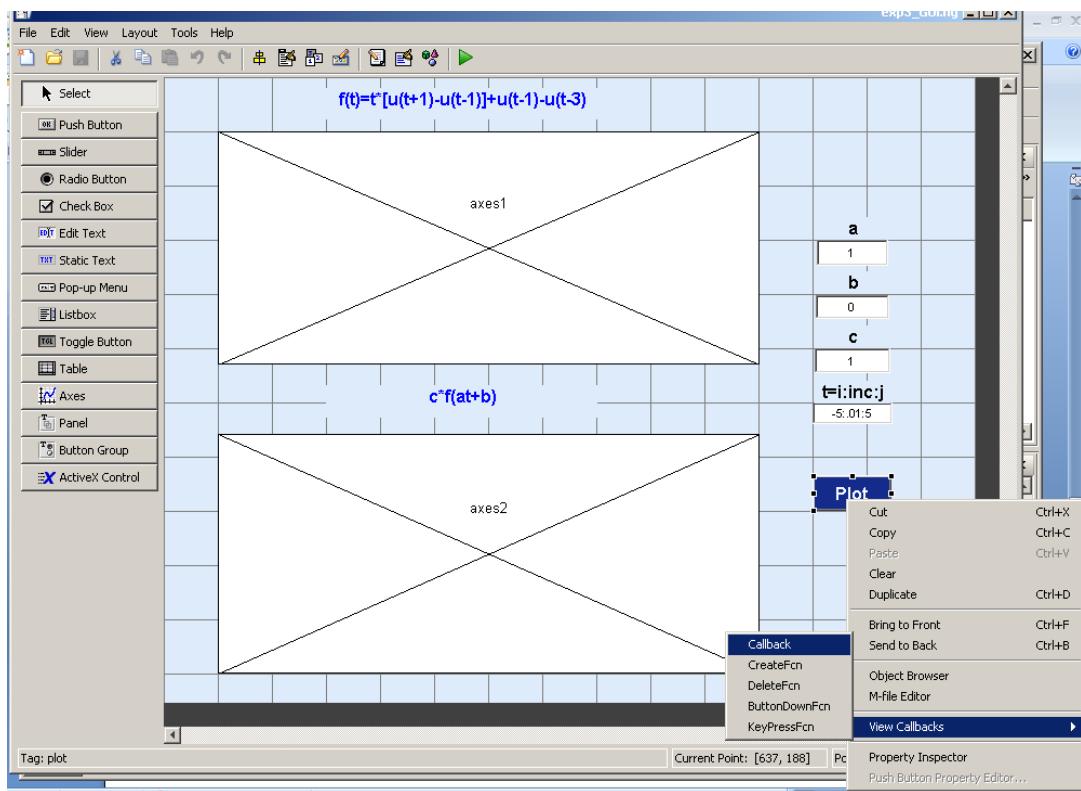
4- The final design will be as follow



5- push on the green arrow (Run) and save the design



6-Right click on the Plot button and select view callback and choose callback .



7-The M file will open as follow

The screenshot shows the MATLAB Editor window with the file name 'Editor - C:\Documents and Settings\Abdo\My Documents\MATLAB\exp3_Gui.m'. The code in the editor is:

```
137
138 % Hint: edit controls usually have a white background on Windows.
139 % See ISPC and COMPUTER.
140 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultBackgroundColor'))
141 set(hObject,'BackgroundColor','white');
142 end
143
144
145 % --- Executes on button press in plot.
146 function plot_Callback(hObject, eventdata, handles)
147 % hObject    handle to plot (see GCBO)
148 % eventdata reserved - to be defined in a future version of MATLAB
149 % handles    structure with handles and user data (see GUIDATA)
150 % Get user input from GUI
151 axes(handles.axes1)
152 a = str2double(get(handles.a,'String'));
153 b = str2double(get(handles.b,'String'));
154 c = str2double(get(handles.c,'String'));
155 t = eval(get(handles.t,'String'));
```

The code is a callback function for a plot button. It sets the background color of the plot to white if running on Windows. It then gets user input for coefficients a, b, c, and t from string inputs and evaluates them.

8-Write the following code under the Plot_callback function

```
axes(handles.axes1)
a = str2double(get(handles.a,'String'));
b = str2double(get(handles.b,'String'));
c = str2double(get(handles.c,'String'));
t = eval((get(handles.t,'String')));
%plot the first function
f=inline('((t>=1) & (t<3))','t');
plot(t,f(t))
ylim ([ min(f(t))- .2 max(f(t))+ .2])
grid on
%plot the second function
f1=c.*f(a*t+b);
axes(handles.axes2)
plot(t,f1)
ylim ([ min(f1)- .2 max(f1)+ .2])
grid on
```

9-Run to see the plot of the functions.

Task 2:

Build a GUI that makes plots of the following signals. Take input from the user for the range of n. The user should be able to input range of n for each function. Place a push button for producing plots of both functions simultaneously .Also insert a pop up menu for producing plots one by one and when one option from the menu is selected the other plot should disappear.

- $f(n)=u(n)-u(n-4)$
- $g(n)=n.u(n)-2(n-4)u(n-4)+(n-8)u(n-8)$

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

```
% Button pushed function: PlotButton
function PlotButtonPushed(app, event)
    % Get the values from the edit fields
    a = str2double(app.aEditField.Value);
    b = str2double(app.bEditField.Value);
    c = str2double(app.cEditField.Value);
    t = eval(app.tiincjEditField.Value);

    % Define the first function
    f = @(t) ((t>=1)&(t<3));

    % Plot the first function on UIAxes
    plot(app.UIAxes, t, f(t));
    ylim(app.UIAxes, [min(f(t))-2, max(f(t))+2]);
    grid(app.UIAxes, 'on');

    % Define and plot the second function on UIAxes2
    f1 = c.*f(a*t+b);
    plot(app.UIAxes2, t, f1);
    ylim(app.UIAxes2, [min(f1)-2, max(f1)+2]);
    grid(app.UIAxes2, 'on');

end
```

Figure 1: Code at the backend of the GUI for Task 1

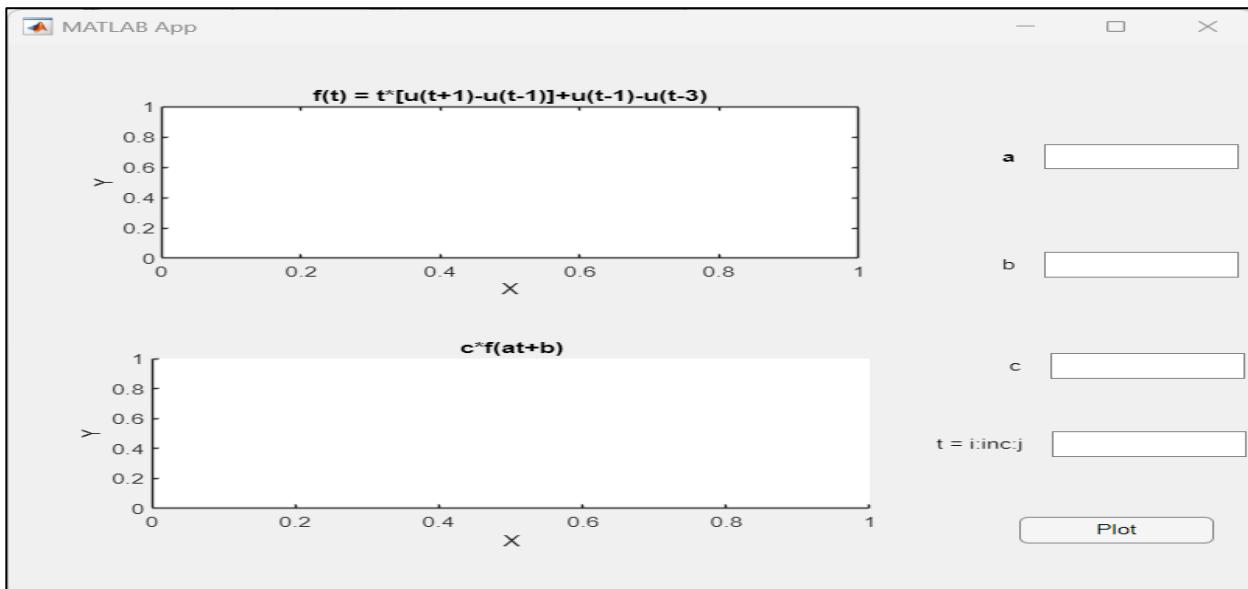


Figure 2: Graphical User Interface (GUI)

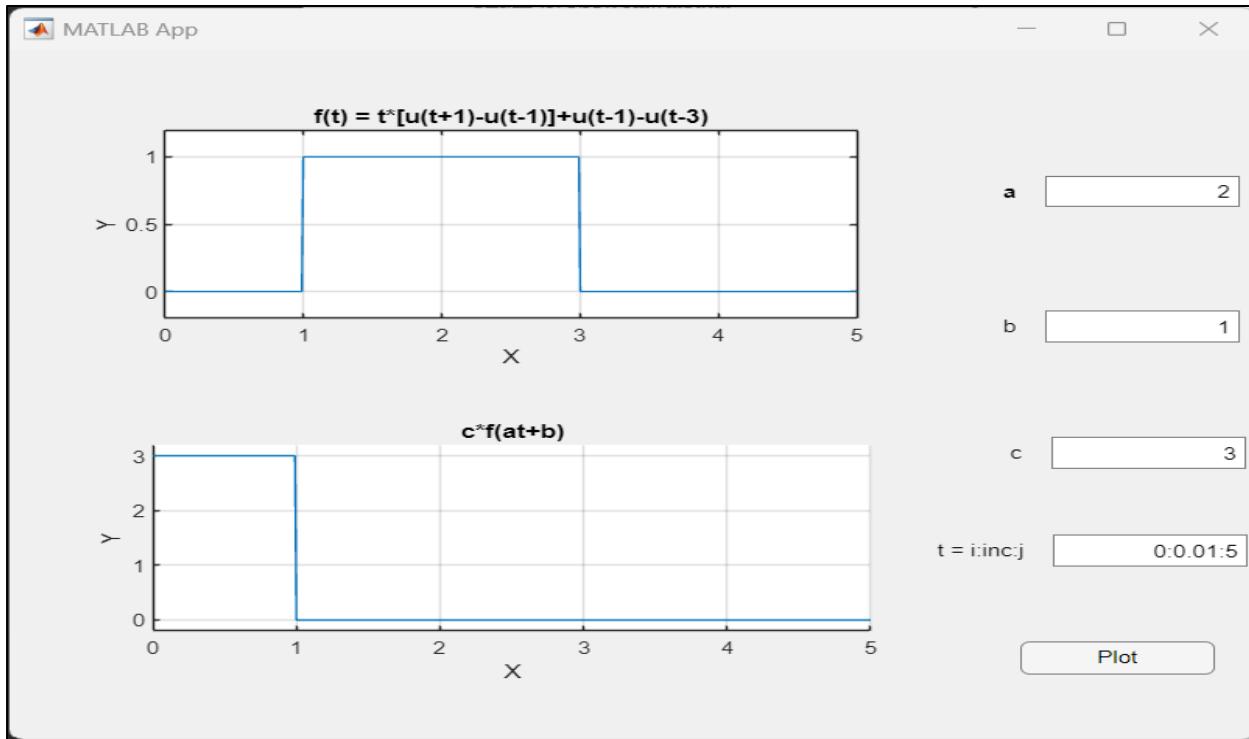


Figure 3: Working GUI. We can change the values of the variable a,b and c. The graph on the lower window will change accordingly. 'a' is the multiplier of the scaling, 'b' gives the time shift to the function 'f(t)' and 'c' is the weight of our function. Lastly, the interval is set by declaring an interval in the vector named 't'.

Task 1 Description:

In task 1, we have plotted a graph of function ' $f(t)$ ' that is the formulation of the addition and subtraction of several time shifted unit step impulse functions. By using our GUI, we can declare the value of our function weights, time scaling and time shift values for the function of ' $f(t)$ '. Additionally, we can also declare the interval for the function. Variable 'a' is for the time scaling, variable 'b' is the time shift value and vertical weight of our function is given by the variable 'c'.

Task 2:

```
% Button pushed function: PlotButton
function PlotButtonPushed(app, event)
    % Get the user-defined range of n from the Edit Field
    n = eval(app.nTincjEditField.Value);

    u_n = (n >= 0);
    result1 = u_n - u_n.*(n-4 >= 0);

    u_n = (n >= 0);
    result2 = n.*u_n - 2*(n-4).*u_n.*((n-4) >= 0) + (n-8).*u_n.*((n-8) >= 0);

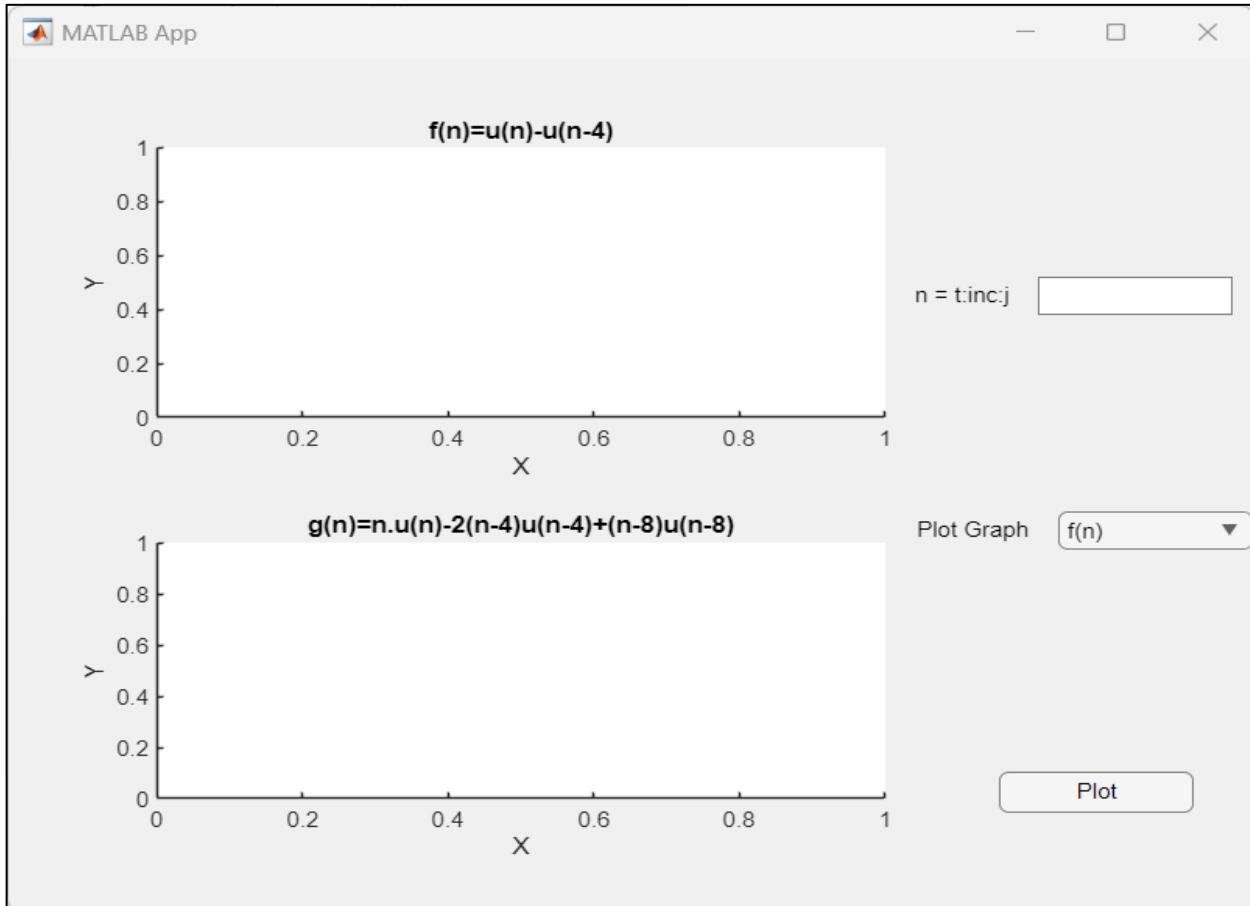
    % Check which option is selected in the drop-down menu
    selectedOption = app.PlotGraphDropDown.Value;

    if strcmp(selectedOption, 'f(n)')
        % Clear UIAxes2 and calculate/plot f(n) in UIAxes
        cla(app.UIAxes2);
        % Plot f(n)
        plot(app.UIAxes, n, result1);

    elseif strcmp(selectedOption, 'g(n)')
        % Clear UIAxes and calculate/plot g(n) in UIAxes2
        cla(app.UIAxes);
        % Plot g(n)
        plot(app.UIAxes2, n, result2);

    elseif strcmp(selectedOption, 'Both')
        plot(app.UIAxes, n, result1);
        plot(app.UIAxes2, n, result2);
    end
    hold(app.UIAxes, 'off');
    hold(app.UIAxes2, 'off');
end
end
```

Figure 4: GUI Code for Task 2

Figure 5: GUI Interface Task 2. Variable 'n' has the interval of the graph. Meanwhile the slidedown options are to select whether we want to plot $f(n)$, $g(n)$ or both plots.

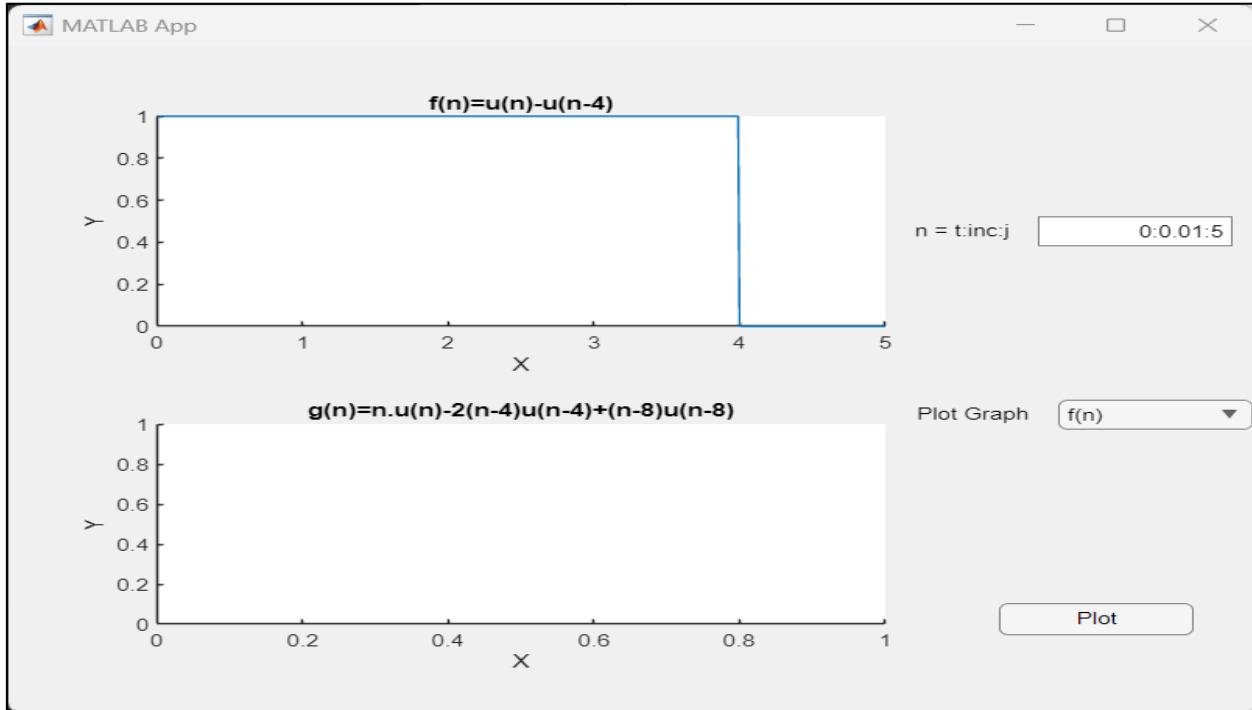


Figure 6: Only $f(n)$ is being plotted from the interval of 0-5 with a sampling of 0.01.

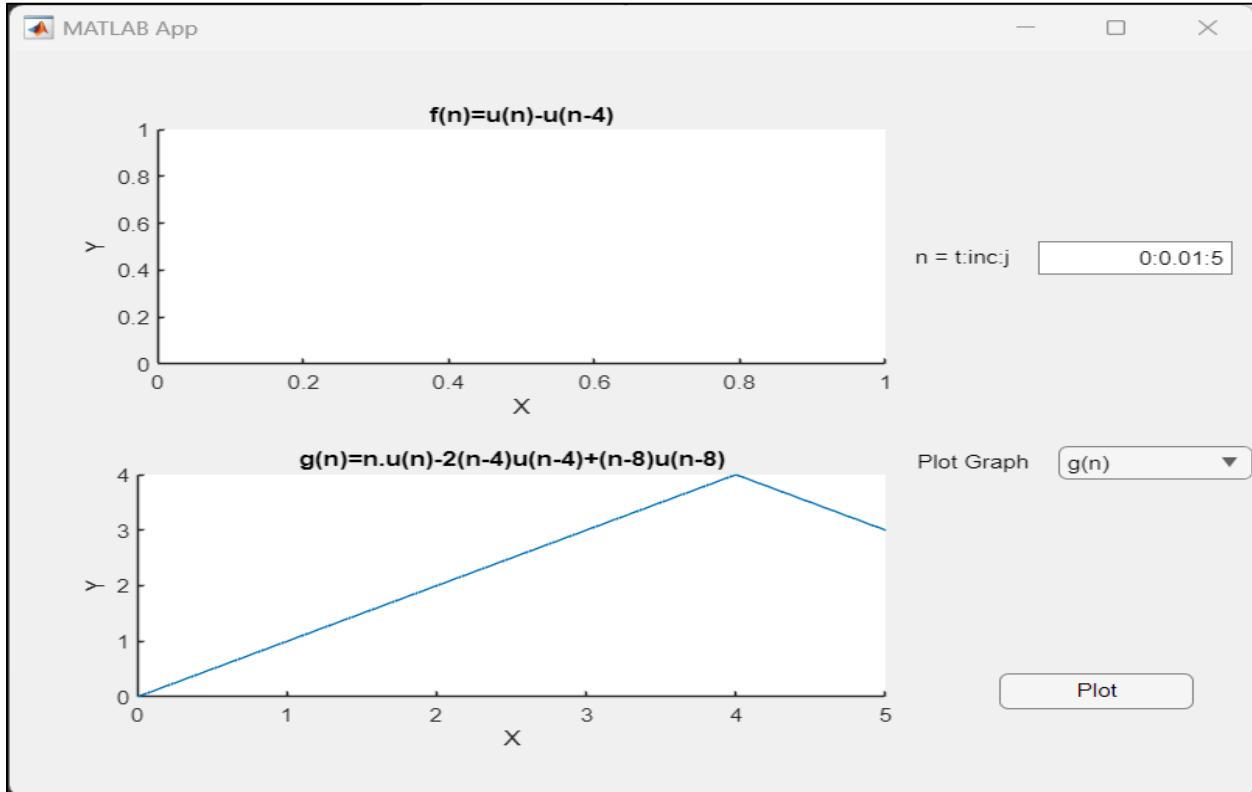


Figure 7: Only $g(n)$ is being plotted from the time interval of 0-5 with a sampling rate of 0.01

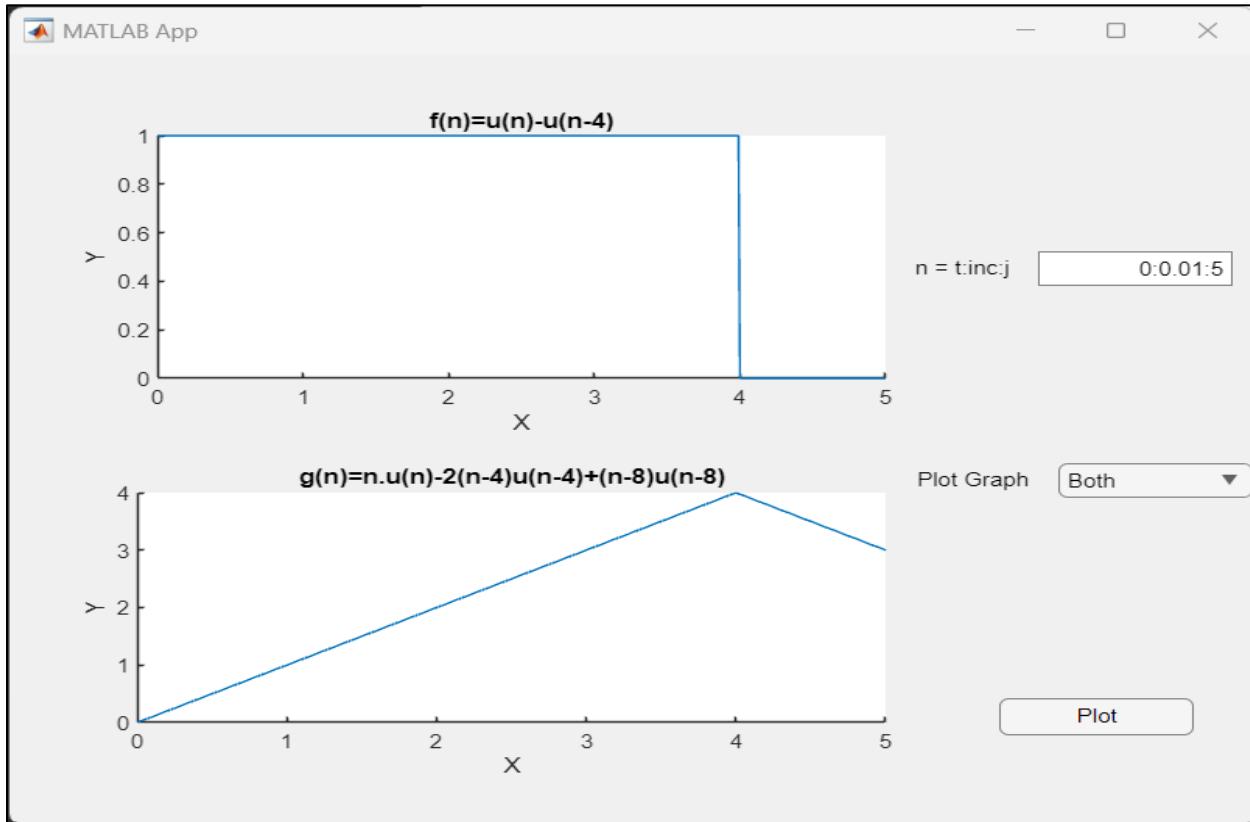


Figure 8: Graph of both $f(n)$ and $g(n)$

Task 2 Description:

Task 2 involved the development of a Graphical User Interface (GUI) to generate plots for two signals, $f(n)$ and $g(n)$. The user was prompted to input the range of values for variable 'n' for each function. The GUI included a push button enabling the simultaneous plotting of both functions. Additionally, a pop-up menu was integrated, allowing users to produce individual plots. When a specific option was selected from the menu, the corresponding plot would be generated, and the other plot would disappear, enhancing user interaction and visualization of the signals. This task aimed to create an intuitive and user-friendly interface for analyzing and visualizing the defined mathematical functions.

Conclusion:

In this experiment, we learned how we can create a Graphical User Interface (GUI) using MATLAB and then how can we use our GUI to be able to influence different parameters of our functions. We can easily declare the level of control that we can have over the process of graph plots.

Verified: _____

Date/Time: _____

Experiment No. 9

Objective:

The objective of this lab is to create a practical understanding of the Continuous time Fourier Series (Chapter 3 of textbook) and to prove various properties of the CTFS.

Theoretical Background:

The Continuous Time Fourier Series is used for representation of continuous-time periodic signals:

CTFS representation of a periodic signal:

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{j k w_o t} = \sum_{k=-\infty}^{\infty} a_k e^{j k \left(\frac{2\pi}{T}\right)t}$$

Fourier Series Coefficients of a periodic signal:

$$a_k = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-j k w_o t} dt = \frac{1}{T} \int_{-T/2}^{T/2} x(t) e^{-j k \left(\frac{2\pi}{T}\right)t} dt$$

Properties of the Continuous Time Fourier Series:

There are many properties associated with the CTFS, in this lab the students will prove the following two properties, where:

- $x(t)$ and $y(t)$ are two continuous time periodic signals with period T and fundamental frequency $w_o = 2\pi/T$
- a_k and b_k are the Fourier Series Coefficients

1. Multiplication:

The Fourier Series representation of the product of two continuous time periodic signals $x(t)$ and $y(t)$ is equal to the convolution of their Fourier Series coefficients

$$FS(x(t) y(t)) = \sum_{l=-\infty}^{\infty} a_l b_{k-l}$$

2. Differentiation:

The Fourier Series representation of differentiation of a periodic signal $x(t)$ is equal to $-jw^*(\text{Fourier Series of } x(t))$.

$$FS\left(\frac{d x(t)}{dt}\right) = jk w_o a_k = j k \frac{2\pi}{T} a_k$$

Tasks:

The following tasks are to be performed **individually** by each student:

Task 1:

Create separate functions in MATLAB for Continuous Time Fourier Series (CTFS), i.e. Fourier series coefficients of a signal and the Inverse Continuous Time Fourier Series (ICTFS), i.e. creating signal from Fourier Series Coefficients.

Task 2:

Using the functions for CTFS and ICTFS created in Task 1, prove the following properties (explained in theoretical background) of CTFS:

- Multiplication Property
- Differentiation Property

The specifications of the two signals should be:

- $x(t) = \sin(\pi t)$
- $y(t) = \cos(\pi t)$
- The signal period $T_p=2\pi$
- Number of coefficients $k= -10$ to 10

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

Task 1

```

syms t ak;
T = 2*pi;
w = (2*pi)/T;
x = sin(t);
ak = CTFS(x, T);
x1 = ICTFS(ak, T);

figure;
subplot(3,1,1);
fplot(x, 'blue', 'linewidth', 2);
title('Original Signal');

subplot(3,1,2);
stem(-10:10, abs(ak));
title('Magnitude of CTFS Coefficients');

subplot(3,1,3);
fplot(x1, 'r', 'linewidth', 2);
title('Reconstructed Signal');

sgtitle('Signal Analysis using CTFS');

```

Figure 1: Task 1 Code

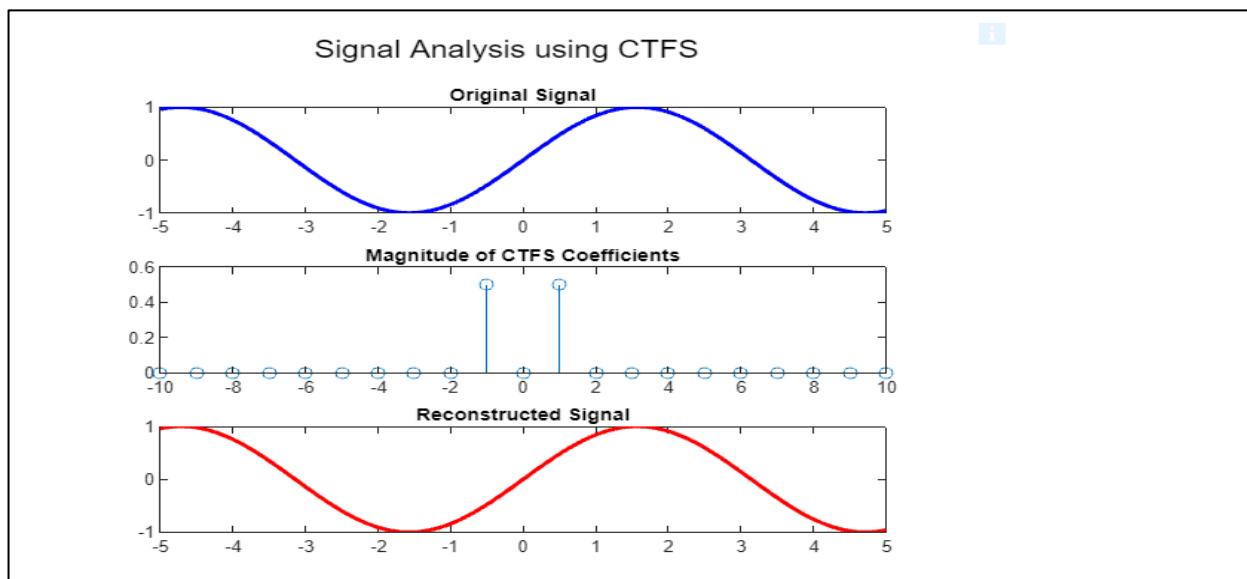


Figure 2: Output of Task 1. The Original Signal is plotted in blue colour. Meanwhile the graph of the CTFS coefficients is also drawn to show the spectrum of the co-efficients. The reconstructed signal then uses the CTFS coefficients to recreate the original signal.

Task 2:**Task 2**

```

syms t;

x = sin(pi*t);
T = 2*pi;
k = -10:10;
w = (2*pi)/T;

Dif = diff(x, t);

figure;

LHS = CTFS(Dif, T);
subplot(2,1,1);
stem(k, abs(LHS));
title('Magnitude of CTFS Coefficients for Differentiation');

ak = CTFS(x, T);
RHS = (1i*k*w .* ak);
subplot(2,1,2);
stem(k, abs(RHS));
title('1i*k*w * CTFS Coefficients');

sgtitle('Comparison of Differentiation and Frequency Modulation in CTFS');

```

Figure 3: Code for Task 2

Comparison of Differentiation and Frequency Modulation in CTFS

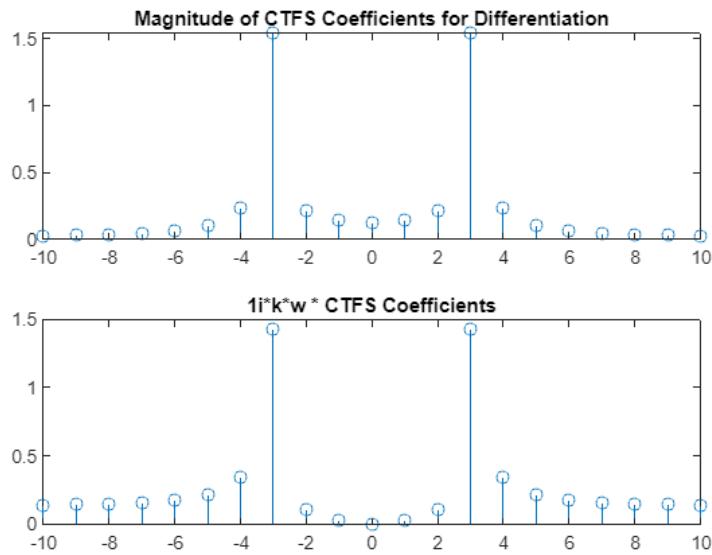


Figure 4: Graph of the CTFS Coefficients of the Differentiation and Frequency Modulation

Function to perform Continuous Time Fourier Series:**Function 1**

```

function ak = CTFS(x, T)
    syms t;
    w = (2*pi)/T;

    ak = zeros(1, 21); % Initializing ak to zeros

    for k = -10:10
        ak(k+11) = (1/T) * int(x * exp(-1i*k*w*t), t, 0, T);
    end
end

```

Figure 5: Function 1 to perform the Continuous Time Fourier Series. It is used to extract the fourier co-efficients from the signals.

Function to perform Inverse Continuous Time Fourier Series:**Function 2**

```

function x = ICTFS(ak, T)
    syms t;
    w = (2*pi)/T;

    x = 0;
    for k = -10:10
        x = x + ak(k+11) * exp(1j*k*w*t);
    end
end

```

Figure 6: Function 2 to perform the Inverse Continuous Time Fourier Series. It is used to reconstruct the signal by using the Fourier co-efficients extracted from the CTFS.

Task 1 Description:

In Task 1, students are required to create separate MATLAB functions for Continuous Time Fourier Series (CTFS) and Inverse Continuous Time Fourier Series (ICTFS). The functions should enable the computation of CTFS coefficients for a given signal and the reconstruction of the original signal from its Fourier Series coefficients. The provided example involves visualizing the original signal, the magnitude of CTFS coefficients, and the reconstructed signal.

Task 2 Description:

Task 2 builds on the functions created in Task 1, focusing on proving specific properties of Continuous Time Fourier Series (CTFS). Students utilize the functions to demonstrate the Multiplication and Differentiation properties, providing insights into how the CTFS behaves when applied to signals undergoing these operations. The visualizations showcase the comparison between the left-hand side (LHS) and right-hand side (RHS) of each property.

Function 1 Description:

Function 1, named CTFS, computes the Fourier Series coefficients of a given signal using the Continuous Time Fourier Series formula. The function takes the signal and its period as input and calculates the coefficients for a specified range of 'k', which is set from -10 to 10. The resulting coefficients are then used for signal analysis and reconstruction.

Function 2 Description:

Function 2, named ICTFS, is the Inverse Continuous Time Fourier Series function. This function takes the CTFS coefficients as input, along with the signal's period, and reconstructs the original signal by summing up the contributions from each coefficient. It plays a crucial role in validating the accuracy of the CTFS computations.

Conclusion:

The lab exercises provide hands-on experience in understanding and applying Continuous Time Fourier Series concepts. Through the implementation of MATLAB functions and visualization of signals, students gain practical insights into the Multiplication and Differentiation properties of CTFS. This comprehensive approach fosters a deeper comprehension of signal processing and lays a solid foundation for further exploration in the field of Fourier analysis.

Verified: _____

Date/Time: _____

Experiment No. 10

Objective:

The objective of this lab is to create a practical understanding of the Discrete Time Fourier Series (Chapter 3 of textbook) and to prove some properties of the DTFS.

Theoretical Background:

The Discrete Time Fourier Series is used for representation of discrete-time periodic signals:

DTFS representation of a periodic signal:

$$x[n] = \sum_{k=0}^{N-1} a_k e^{j k w_o n} = \sum_{k=0}^{N-1} a_k e^{j k \left(\frac{2\pi}{N}\right)n}$$

Fourier Series Coefficients of a periodic signal:

$$a_k = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j k w_o n} = \frac{1}{N} \sum_{n=0}^{N-1} x[n] e^{-j k \left(\frac{2\pi}{N}\right)n}$$

Periodic Convolution:

The convolution of two periodic sequences is calculated through periodic convolution. The basic idea is to take one period of each signal and perform periodic convolution to get one period of the resultant sequence. Then this one period is repeated over all time to create a periodic sequence. The formula for periodic convolution is:

$$y[n] = \sum_{m=0}^{N-1} x_1[m] x_2[n-m]$$

- where $x_1[n]$ and $x_2[n]$ are two discrete time periodic sequences and $y[n]$ is one period of a discrete time periodic sequence
- N is the length of each sequence

Properties of the Discrete Time Fourier Series:

There are many properties associated with the DTFS, in this lab the students will prove the following two properties, where:

- $x(t)$ and $y(t)$ are two continuous time periodic signals with period T and fundamental frequency $w_o = 2\pi/T$
- a_k and b_k are the Fourier Series Coefficients

1. Periodic Convolution:

This property states that the DTFS of periodic convolution of two discrete time periodic sequences is equal to multiplication of the DFS coefficients of the sequences.

$$\underbrace{x[n] * y[n]}_{\text{Periodic convolution}} = \sum_{r=-N}^{N-1} x[r]y[n-r] \xrightarrow{\text{FS}} N a_k b_k$$

2. Frequency Shifting:

The shifting of DFS coefficients is equivalent to multiplication of complex exponential to the actual periodic signal.

$$e^{jM(\frac{2\pi}{N})n} x[n] \xrightarrow{\text{FS}} a_{k-M}$$

Tasks:

The following tasks are to be performed by each student:

Task 1:

Create separate functions in MATLAB for Discrete Time Fourier Series (DTFS), i.e. Fourier series coefficients of a signal and the Inverse Discrete Time Fourier Series (IDTFS), i.e. creating signal from Fourier Series Coefficients.

Task 2:

a). Create a function that performs periodic convolution on two discrete time periodic sequences of same length.

b). Using the functions for DTFS and IDTFS created in Task 1 and the function for periodic convolution, prove the following properties (explained in theoretical background) of DTFS:

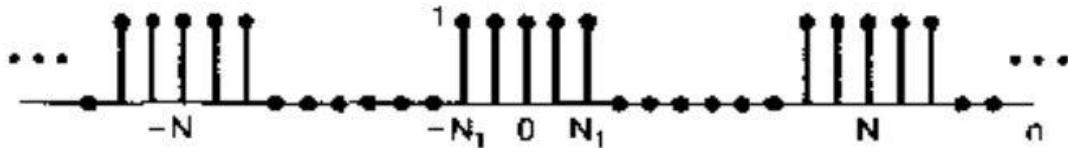
- Periodic Convolution
- Frequency Shifting

The specifications of the two signals are:

- $x[n]=[1 2 1 2 1 2 1]$
- $y[n]=[1 0 1 0 1 0 1]$
- The period N is the length of the signal

Task 3:

Take the discrete time periodic square wave as shown below:



Take $N_1=5$ and the number of zeros in each period is also equal to N_1 (Example 3.12 of textbook)

Calculate the DTFS of this square wave, then calculate its inverse DTFS using the IDTFS function. Plot the original signal, the DTFS and the IDTFS results in the same figure using subplot.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

```
function ak = DTFS(x, N)
ak = zeros(1,N);
for k = 1:N
    for n = 1:N
        ak(k) = ak(k) + (1/N) * x(n) * exp(-1i*2*pi*(k)*(n)/N);
    end
end
end
```

Figure 1: Discrete Time Fourier Series Function Code

```
function xn = IDTFS(ak,N)
xn = zeros(1,N);
for n = 1:N
    for k = 1:N
        xn(n) = xn(n) + ak(k)*exp(1i*2*pi*(k)*(n)/N);
    end
end
end
```

Figure 2: Inverse Discrete Time Fourier Series Code

```

x = [1 2 1 2 1 2 1];
N = length(x);
ak = DTFS(x, N);
x1 = IDTFS(ak, N);
subplot(3,1,1);
stem(abs(x));
title('Original Discrete-Time Signal');
subplot(3,1,2);
stem(abs(ak));
title('Coefficients of Fourier Series');
subplot(3,1,3);
stem(abs(x1));
title('Reconstructed Discrete-Time Signal');

```

Figure 3: Task 1 Code

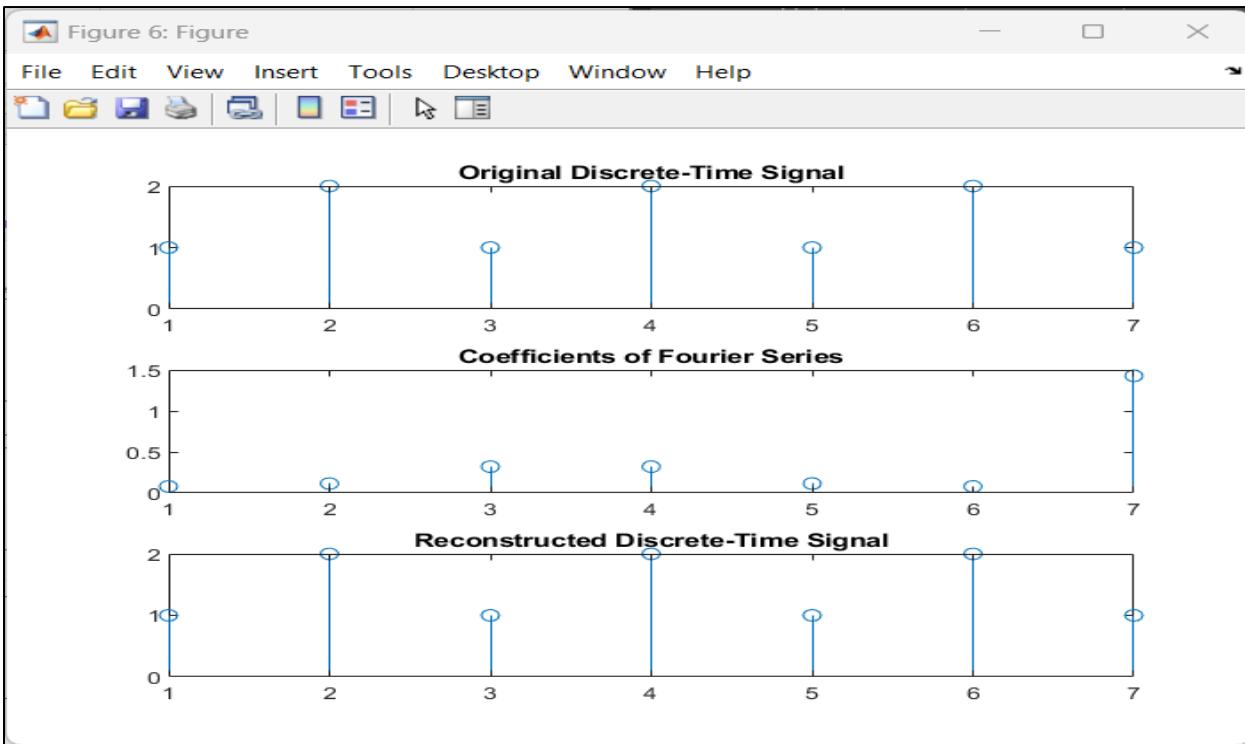


Figure 4: Task 1 Output

Task 1 Description:

This MATLAB code implements the Discrete Time Fourier Series (DTFS) and its inverse, Inverse Discrete Time Fourier Series (IDTFS), through distinct functions. The 'DTFS' function computes the Fourier series coefficients (ak) of a given discrete-time signal, while the 'IDTFS' function reconstructs the original signal from these coefficients. The code then applies these functions to process a sample discrete-time signal, visualizing the original signal, its Fourier series coefficients, and the reconstructed signal using stem plots. This modular approach enhances code readability and promotes reusability, providing a clear understanding of the DTFS and IDTFS processes in the context of signal analysis.

Task 2:Part (a):

```

function result = periodic_convolution(x, y)
    N = length(x);
    result = zeros(1, N);
    for n = 1:N
        for m = 1:N
            result(n) = result(n) + x(m) * y(mod((n - m), N) + 1);
        end
    end
end

```

Figure 5: Code for Periodic Convolution Function

Part(b):

```

x=[1 2 1 2 1 2 1];
y=[1 0 1 0 1 0 1];

z = periodic_convolution(x,y);
LHS = DTFS(z,N);

figure;
subplot(2,1,1);
stem(abs(LHS));
title('DTFS of Periodic Convolution of Two Signals');

ak = DTFS(x,N);
bk = DTFS(y,N);
N = length(x);

RHS = N.*ak.*bk;
subplot(2,1,2);
stem(abs(RHS));
title('Multiplication of the DFS coefficients of the Sequences');

```

Figure 6: Code for Task 2 Periodic Convolution

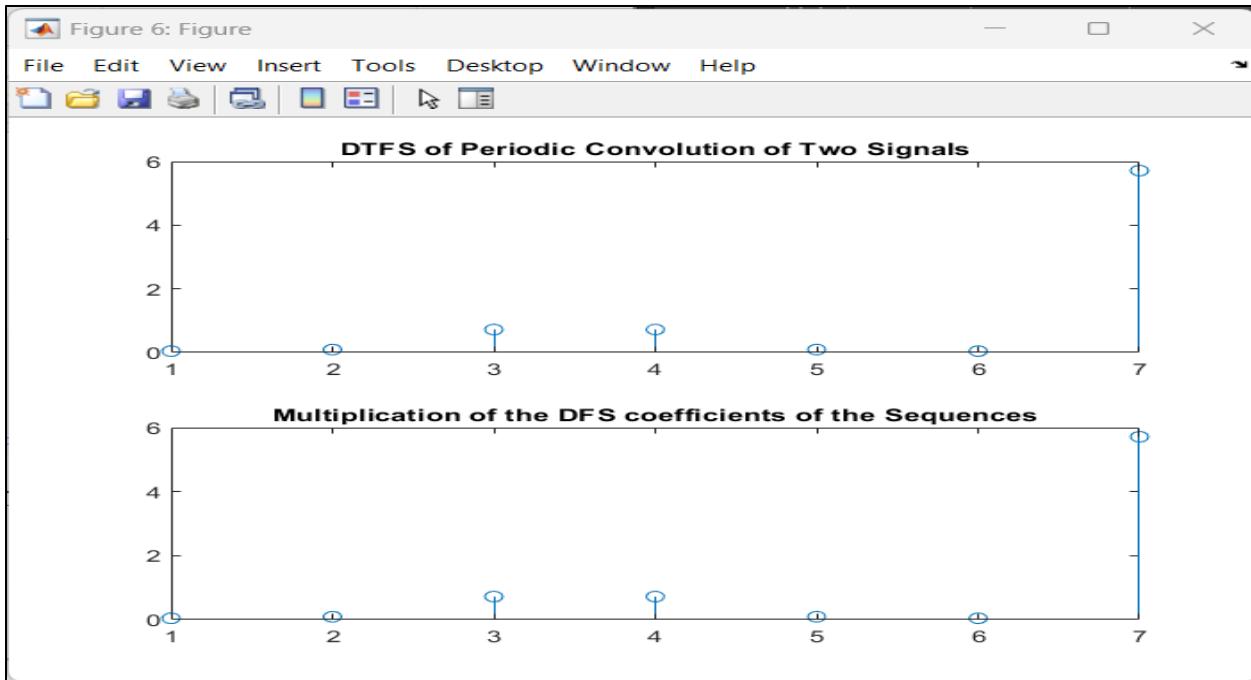


Figure 7: Task 2 Periodic Convolution Output

```

a = zeros(1:N);
%M = input("Enter value of shift: ");
M = 9;
% LHS
for n = 1:N
    a(n) = exp(1i * 2 * pi * M * (n/ N)) * x(n);
end

LHS = DTFS(a,N);

figure;
subplot(2,1,1);
stem(abs(LHS));
title('Shifting of DT Fourier Series Coefficients');

% RHS
z = DTFS(x,N);
RHS = circshift(z, [0 M]);

subplot(2,1,2);
stem(abs(RHS));
title('Multiplication of Complex Exponential With x[n]')

```

Figure 8: Task 2 Frequency Shift Code

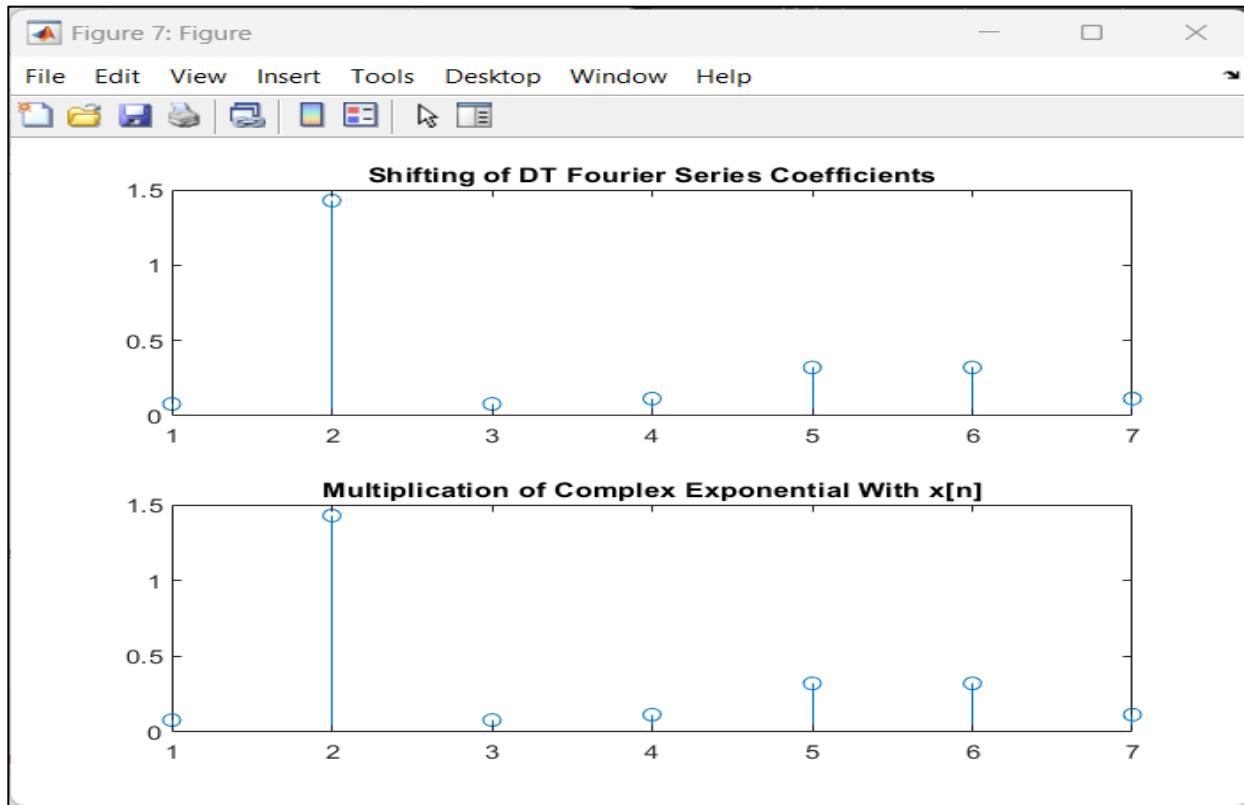


Figure 9: Task 2 Frequency Shift Output

Task 2 Description:

This MATLAB code for Task 2 efficiently verifies essential properties of Discrete Time Fourier Series (DTFS). It begins by implementing a periodic convolution function, subsequently demonstrating the periodic convolution property by comparing DTFS results with coefficient multiplications of individual sequences. Additionally, the frequency-shifting property is validated by comparing DTFS coefficients of a shifted sequence with those obtained by multiplying the original coefficients with a complex exponential term. The visualizations offer a succinct and compelling confirmation of these fundamental DTFS properties, aligning with the specified theoretical background.

Task 3:

```

A = [ones(1,5) zeros(1,5)];
x = [A A];
N = length(x);

x_DTFs = DTFs(x,N);
x_IDTFS = IDTFS(x_DTFs,N);

figure;
subplot(3,1,1);
stem(abs(x));
title('Original Discrete-Time Signal x');

subplot(3,1,2);
stem(abs(x_DTFs));
title('Coefficients of Fourier Series of x');

subplot(3,1,3);
stem(abs(x_IDTFS));
title('Reconstructed Discrete-Time Signal of x');

```

Figure 10: Task 3 Code

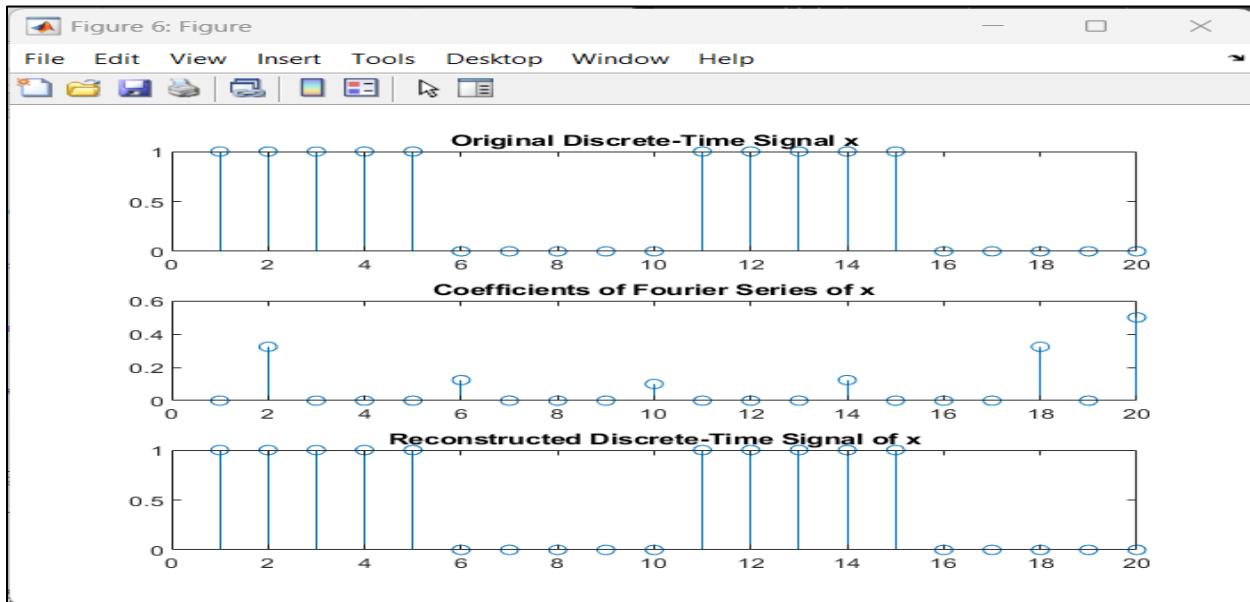


Figure 11: Task 3 Output

Task 3 Description:

This MATLAB code addresses Task 3 by generating a discrete-time periodic square wave and calculating its Discrete Time Fourier Series (DTFS). The Inverse Discrete Time Fourier Series (IDTFS) function is then employed to reconstruct the original signal. The resulting signals—the original, DTFS coefficients, and the reconstructed signal—are efficiently visualized in a single figure using subplots. This concise presentation offers a clear illustration of the DTFS and IDTFS processes applied to the periodic square wave, highlighting the transformation between the original signal and its Fourier series components.

Verified: _____

Date/Time: _____

Experiment No. 11

Objective:

The objective of this lab is to create a practical understanding of the Discrete Time Fourier Transform (Chapter 5 of textbook) and to prove some properties of the DTFT.

Theoretical Background:

The Discrete Time Fourier Transform is used for representation of discrete-time a-periodic signals:

DTFT representation of a finite discrete signal (Analysis equation):

$$X(e^{jw}) = \sum_{n=0}^{N-1} x[n]e^{-jwn}$$

Calculation of a signal from its DTFT (Synthesis equation):

$$x[n] = \frac{1}{2\pi} \int_0^{2\pi} X(e^{jw})e^{jwn} dw$$

Properties of the Discrete Time Fourier Transform:

There are many properties associated with the DTFT, in this lab the students will prove the following two properties, where:

- $x[n]$ and $y[n]$ are two discrete time a-periodic signals
- $X(e^{jw})$ and $Y(e^{jw})$ are the DTFT representation of $x[n]$ and $y[n]$ respectively

1. Convolution Property:

This property states that the DTFT of convolution of two discrete time sequences is equal to multiplication of the DTFTs of the sequences.

$$x[n] * y[n] = IDTFT \left(X(e^{jw})Y(e^{jw}) \right)$$

2. Multiplication Property:

The DTFT of multiplication of two discrete time a-periodic signals is equal to the periodic convolution of the DTFT of the individual signals.

$$x[n]y[n] = IDTFT \left(\frac{1}{2\pi} \int_0^{2\pi} X(e^{j\theta})Y(e^{j(w-\theta)}) d\theta \right)$$

Tasks:

The following tasks are to be performed by each student.

Task 1:

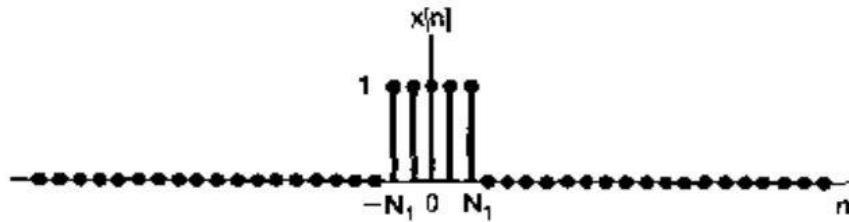
Create separate functions in MATLAB for Discrete Time Fourier Transform (DTFT), i.e. analysis equation, and Inverse Discrete Time Fourier Transform (IDTFT), i.e. synthesis equation.

Task 2:

Consider the rectangular pulse:

$$x[n] = \begin{cases} 1, & |n| \leq N_1 \\ 0, & |n| > N_1 \end{cases}$$

which is illustrated below for $N_1 = 2$.



Find the DTFT of $x[n]$ using the DTFT function created in Task 1. Then, find the IDFT of this result using the IDTFT function, also created in Task 1. Using subplot, display the input signal $x[n]$, and the output of the IDTFT function. The result in both subplots should be same. This task is an implementation of Example 5.3 of your textbook.

Task 3:

Using the functions created in task 1, prove the convolution and multiplication properties of the DTFT in separate codes. Display the time domain (n - domain) results in each case using the subplot command.

The specifications of the two signals are given below for both properties separately.

For Convolution Property:

- $x[n]=[1\ 0\ 1\ 0\ 1]$
- $y[n]=[1\ 1\ 0\ 1\ 0]$
- N is the length of the signal.

For Multiplication Property:

- $x[n]=[1 \ 2 \ 3 \ 1 \ 3]$
- $y[n]=[3 \ 4 \ 3 \ 3 \ 2]$
- N is the length of the signal.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

```
function [X] = DTFT(x,N)
syms w
X = 0;
for n = 0:N-1
X = X + (x(n+1) .* exp(-1j .* w .* n));
end
end
```

Figure 1: Discrete Time Fourier Transform Function Code

```
function [x] = IDTFT(X, N)
syms w
for n = 0:N-1
x(n+1) = 1/(2.*pi) .* int(X .* exp(1i .* w .* n), w, 0, 2*pi);
end
end
```

Figure 2: Inverse Discrete Time Fourier Transform Function Code

Task 1 Description:

This MATLAB code fulfills Task 1 by providing separate functions for the Discrete Time Fourier Transform (DTFT) and the Inverse Discrete Time Fourier Transform (IDTFT). The `DTFT` function computes the DTFT coefficients using an analysis equation, incorporating a summation over discrete sequence elements multiplied by complex exponential terms. In contrast, the `IDTFT` function synthesizes a sequence from its DTFT coefficients using a synthesis equation, involving an integral expression to compute each sequence element. These functions enable the transformation between time and frequency domains, offering a versatile toolset for signal analysis and synthesis in the context of Discrete Time Fourier Transform.

Task 2:

```

N1 = 2;
n = -10:10;
x_n = (abs(n) <= N1);
N = length(n);

% Find DTFT of x[n]
X_w = DTFT(x_n, N);

% Find IDFT of X(w)
x_reconstructed = IDTFT(X_w, N);

% Plotting
figure;

% Plot input signal x[n]
subplot(3,1,1);
stem(n, x_n, 'b', 'LineWidth', 1.5);
title('Input Signal x[n]');
xlabel('n');
ylabel('x[n]');

% Plot of DTFT function output
subplot(3,1,2);
ezplot(abs(X_w));
title('Discrete Time Fourier Transform');

% Plot output of IDTFT function
subplot(3,1,3);
stem(n, abs(x_reconstructed), 'r', 'LineWidth', 1.5);
title('Output of IDTFT function');
xlabel('n');
ylabel('IDTFT(X[\omega])');

% Adjust subplot layout
sgtitle('Rectangular Pulse DTFT and IDTFT');

% Ensure the plots are on the same scale
axis tight;

```

Figure 3: Task 2 Code

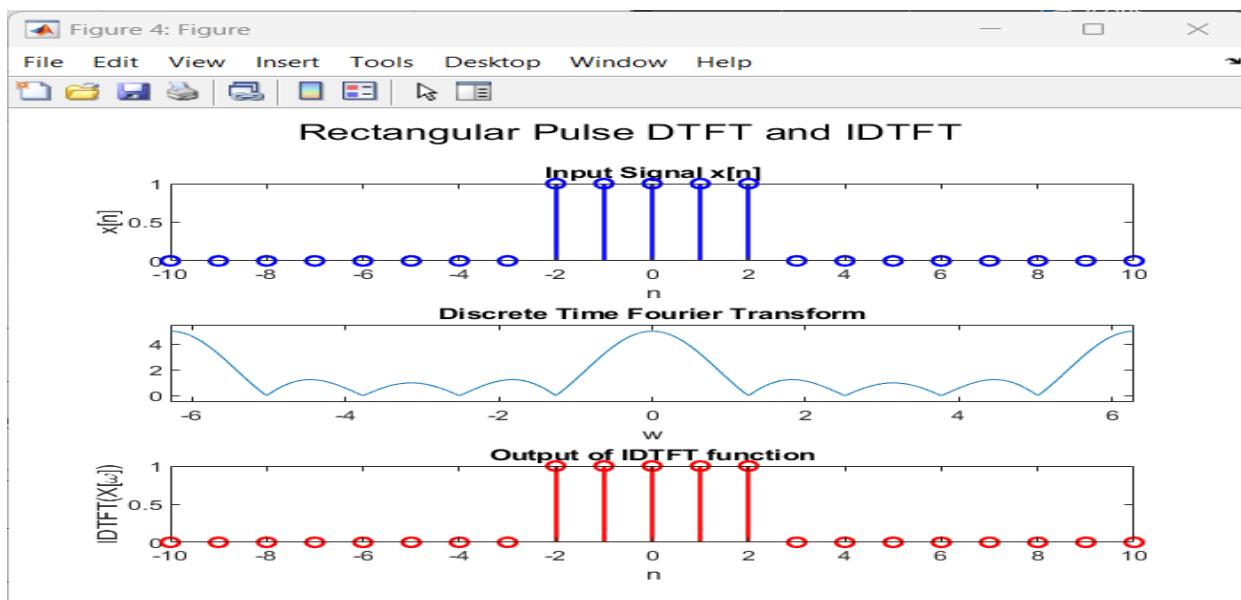


Figure 4: Task 2 Output

Task 2 Description:

This MATLAB code, designed for Task 2, implements a rectangular pulse signal with $N_1 = 2$, where the signal is 1 for $|n| < N_1$ and 0 otherwise. Leveraging the Discrete Time Fourier Transform (DTFT) function from Task 1, the code calculates the DTFT of the rectangular pulse. Subsequently, the Inverse Discrete Time Fourier Transform (IDTFT) function is employed to reconstruct the original signal from its DTFT coefficients. The resulting signal is then plotted alongside the input signal in a subplot arrangement. The visual confirmation of an identical input and output signal in the subplots verifies the effectiveness of the IDTFT process, aligning with Example 5.3 in the textbook and illustrating the practical application of these signal processing functions.

Task 3:

```

x = [1 0 1 0 1];
N1 = length(x);
y = [1 1 0 1 0];
N2 = length(y);

figure;
% LHS
LHS = conv(x,y);
subplot(2,1,1);
stem(abs(LHS));
title('Convolution of Two Discrete Time Signals');

% RHS
A = DTFT(x,N1);
B = DTFT(y,N2);
C = A .* B;
RHS = IDTFT(C, length(LHS));

subplot(2,1,2);
stem(abs(RHS));
title('IDTFT of Product of DTFTs of Two Discrete Time Signals');

```

Figure 5: Task 3 Convolution Code



Figure 6: Task 3 Convolution Output

```

x = [1 2 3 1 3];
N1 = length(x);
y = [3 4 3 3 2];
N2 = length(y);

figure;
% LHS
LHS = x .* y;
subplot(2,1,1);
stem(abs(LHS));
title('Product of Two Discrete Time Signals');

% RHS
syms w theta
A = DTFT(x, N1);
C = DTFT(y, N2);
B = subs(A, w, theta);
D = subs(C, w, w-theta);
E = B * D;
F = 1/(2.*pi) .* int(E, theta, 0, 2*pi);
RHS = IDTFT(F,length(LHS));

subplot(2,1,2);
stem(abs(RHS));
title('Periodic Convolution of DTFT of Two Discrete Time Signals');

```

Figure 7: Task 3 Multiplication Property Code

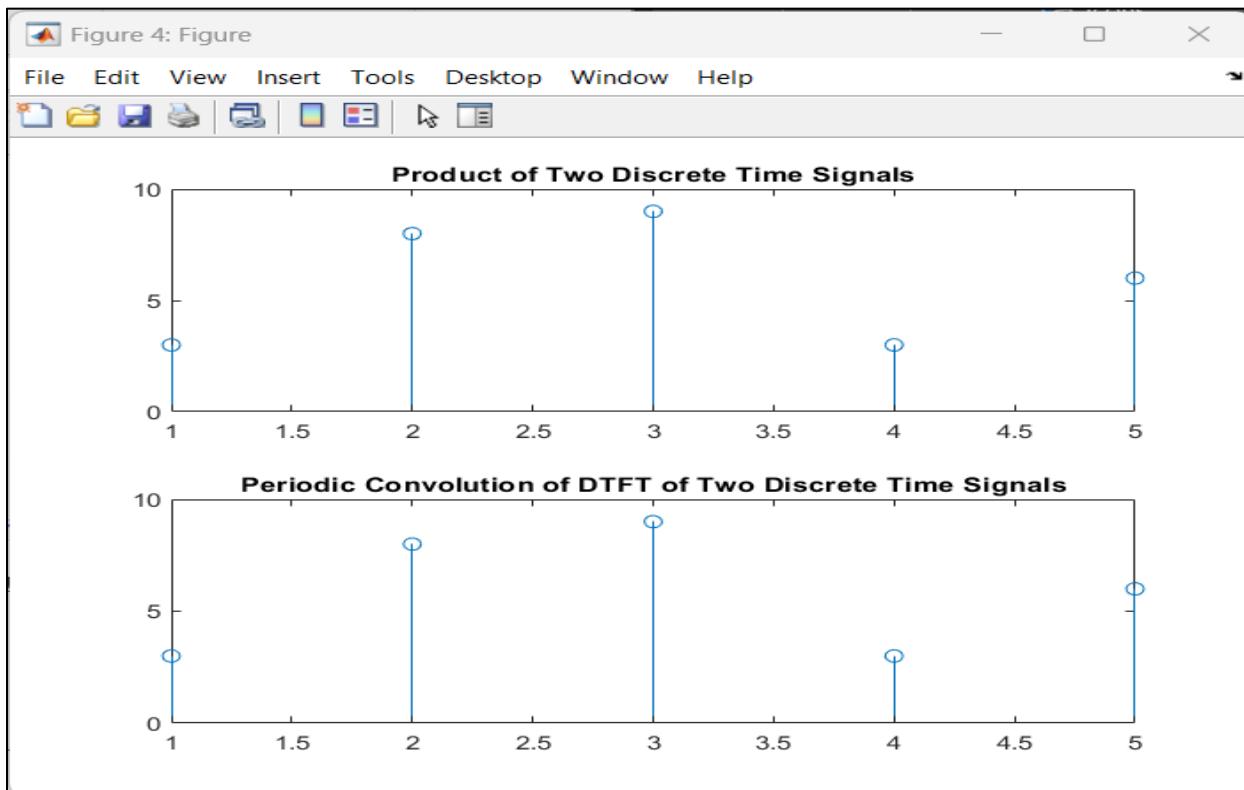


Figure 8: Task 3 Multiplication Property Output

Task 3 Description:

This MATLAB code, designed for Task 3, effectively validates the convolution and multiplication properties of the Discrete Time Fourier Transform (DTFT) using functions created in Task 1. For the convolution property, signals $x[n] = [1 \ 0 \ 1 \ 0 \ 1]$ and $y[n] = [1 \ 1 \ 0 \ 1 \ 0]$ are processed, displaying both time and frequency domain results in subplots. Similarly, for the multiplication property, signals $x[n] = [1 \ 2 \ 3 \ 1 \ 3]$ and $y[n] = [3 \ 4 \ 3 \ 3 \ 2]$ are employed, and the code verifies the DTFT multiplication property by comparing time domain product with the periodic convolution of DTFTs in the frequency domain. The visualizations provide a concise and compelling demonstration of the convolution and multiplication properties of DTFT, aligning with the specified theoretical background in the task.

Verified: _____

Date/Time: _____

Experiment No. 12

Objective:

The objective of this lab is to create a practical understanding of the Continuous Time Fourier Transform (Chapter 4 of textbook) and to prove some properties of the CTFT. Also, students will learn implementation of the Laplace Transform (Chapter 9 of textbook) and prove some of its properties.

Theoretical Background:

Continuous Time Fourier Transform:

The Continuous Time Fourier Transform is used for representation of continuous-time a-periodic signals.

Fourier Transform of a continuous time signal (Analysis equation):

$$X(jw) = \int_{-\infty}^{\infty} x(t)e^{-jwt} dt$$

To calculate the Inverse Fourier Transform (Synthesis equation):

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(jw)e^{jwt} dw$$

Properties of the Continuous Time Fourier Transform:

There are many properties associated with the CTFT; in this lab the students will prove the following two properties:

1. Time Shifting Property:

This property states that the CTFT of a time shifted continuous time signal is equal to multiplication of the Fourier Transform of the original signal by a complex exponential.

$$CTFT\{x(t - t_o)\} = e^{-jw t_o} X(jw)$$

2. Differentiation Property:

This property states that the CTFT of the differentiation of a continuous time signal is equal to multiplication of the CTFT of the original signal with (jw).

$$CTFT\left\{\frac{d x(t)}{dt}\right\} = jw X(jw)$$

where,

- $x(t)$ is a continuous time a-periodic signal
- $X(jw)$ is the CTFT representation of $x(t)$

Tasks:

The following tasks are to be performed by each student:

Task 1:

Using symbolic variables, calculate the Fourier transform of a signal $x = e^{-t^2}$. Also calculate the Inverse Fourier Transform to get the original signal. Plot all three signals in a subplot figure.

Task 2:

- a) Prove the time shifting property for the CTFT using the signal $x = te^{-t^2}$. The time shift given to this signal is $t_0 = 3$. Plot the signals in time domain using subplot.
- b) Using the input signal $y = e^{-t^2}$, prove the differentiation property of the CTFT. Plot the signals in time domain using subplot.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

```
function X = CTFT(x)
    syms w t
    X = int(x * exp(-1i * w * t), t, -Inf, Inf);
end
```

Figure 1: Continuous Time Fourier Transform Function Code

```
function x = ICTFT(X)
    syms w t
    x = 1/(2*pi) * int(X * exp(1i * w * t), w, -Inf, Inf);
end
```

Figure 2: Inverse Continuous Time Fourier Transform Function Code

```
% Define symbolic variables
syms t w

% Define the signal x(t)
x_t = exp(-t^2);

% Calculate the Fourier transform of x(t)
X_w = CTFT(x_t);

% Calculate the Inverse Fourier Transform to get the original signal
x_reconstructed_t = ICTFT(X_w);

% Plot the original signal, its Fourier transform, and the reconstructed signal
figure;

% Original signal x(t)
subplot(3, 1, 1);
fplot(x_t, [-5, 5]);
title('Original Signal x(t) = e^(-t^2)');
xlabel('t');
ylabel('x(t)');

% Fourier transform X(jw)
subplot(3, 1, 2);
fplot(abs(X_w), [-10, 10]);
title('Fourier Transform |X(jw)|');
xlabel('w');
ylabel('|X(jw)|');

% Reconstructed signal from Inverse Fourier Transform
subplot(3, 1, 3);
fplot(x_reconstructed_t, [-5, 5]);
title('Reconstructed Signal from Inverse Fourier Transform');
xlabel('t');
ylabel('Reconstructed x(t)');

% Adjust subplot layout
sgtitle('Continuous-Time Fourier Transform of x(t) = e^(-t^2)');
```

Figure 3: Task 1 Code

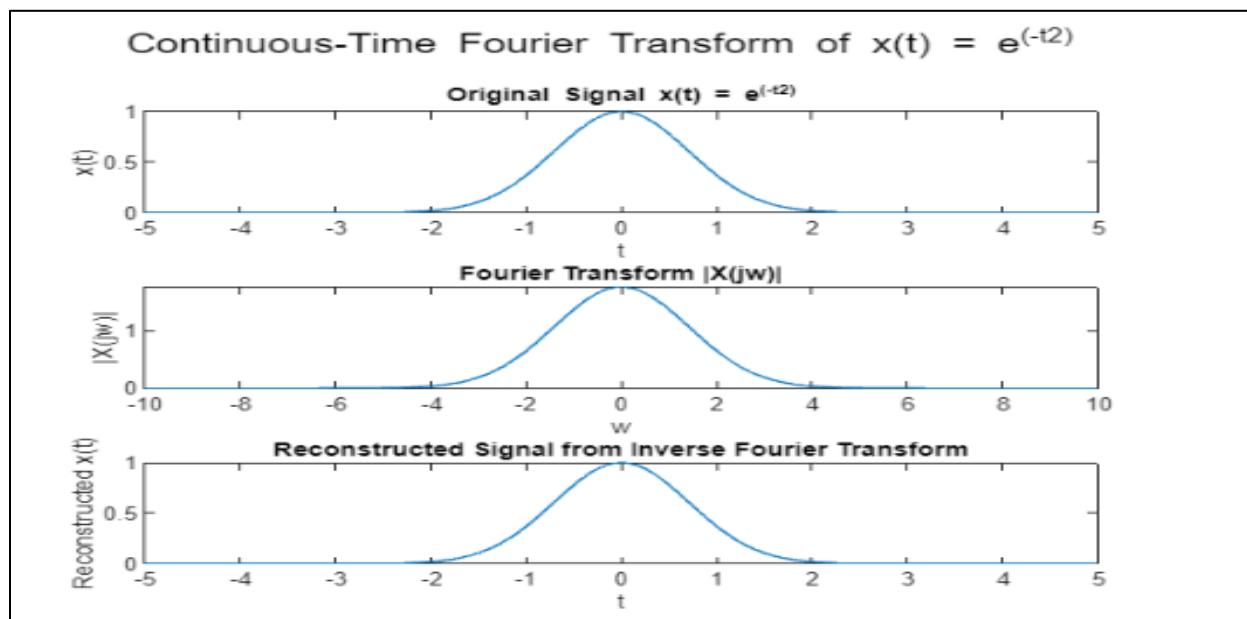


Figure 4: Task 1 Output

Task 2

```
% Symbolic variables
syms w t

to = 3;
x = t * exp(-t^2);

figure;

% LHS
LHS = subs(x, t, t-to);
subplot(2,1,1);
ezplot(LHS);

title('Time Shifted Continuous Time Signal');

% RHS
X = CTFT(x);
Z = X .* exp(-li .* w .* to);

RHS = ICTFT(Z);
subplot(2,1,2);
ezplot(RHS);

title('ICTFT of Product of Complex Exponential with CTFT of the Original Signal')
```

Figure 5: Task 2 Time Shifting Property Code

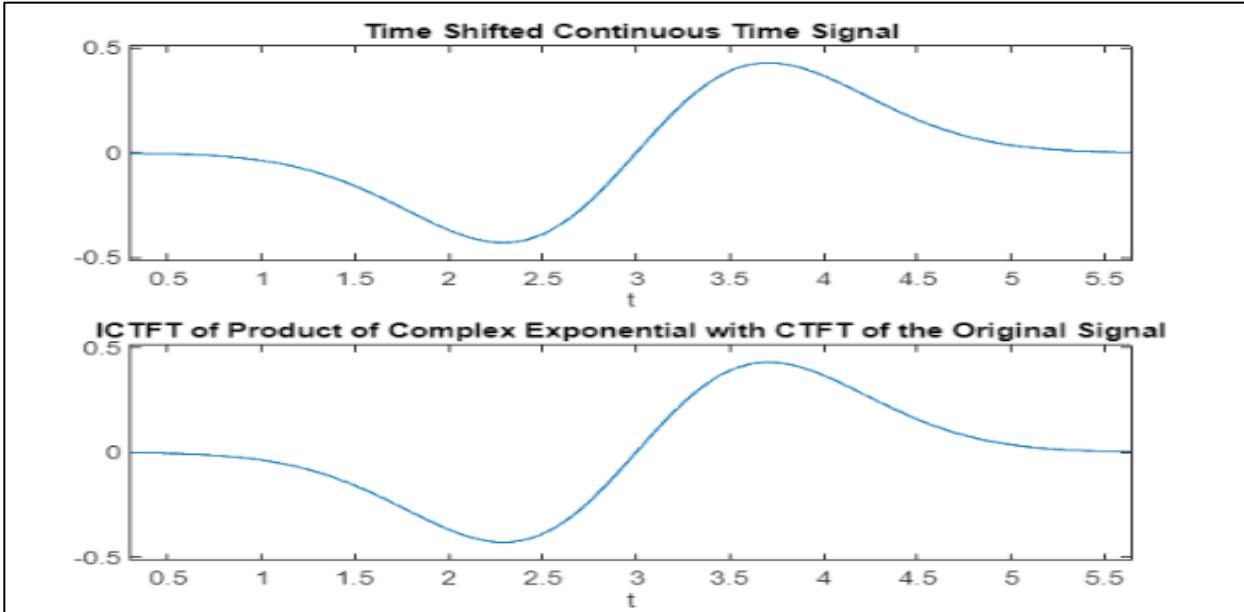


Figure 6: Task 2 Time Shifting Property Output

```

syms w t
y = exp(-t^2);

figure;

% LHS
LHS = diff(y,t);

subplot(2,1,1);
ezplot(LHS);

title('Differentiation of Continuous Time Signal');

% RHS
Y = CTFT(y);
Z = (1i .* w) * (Y);

RHS = ICTFT(Z);

subplot(2,1,2);
ezplot(RHS);

title('ICTFT of Product of jw and CTFT of the Original Signal');

```

Figure 7: Task 2 Differentiation Property Code

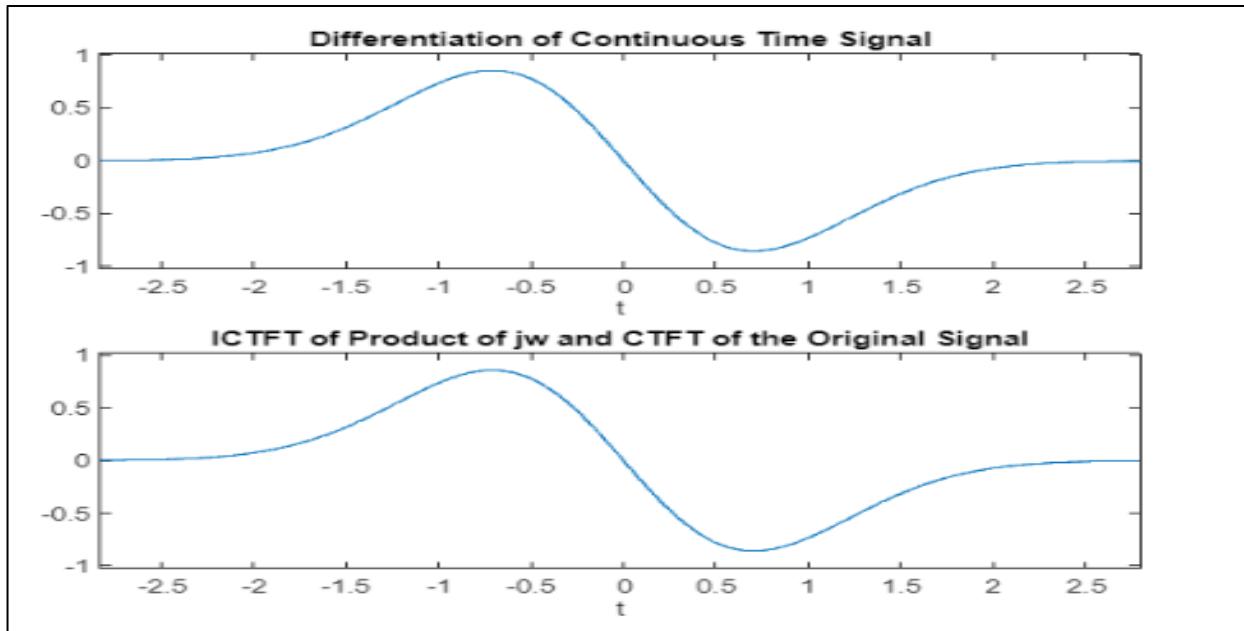


Figure 8: Task 2 Differentiation Property Output

Task 1 Description:

This MATLAB code explores the Continuous-Time Fourier Transform (CTFT) and its inverse, as outlined in the provided functions. It begins by defining a symbolic variable and creating a signal $x(t) = e^{-t^2}$. The CTFT function computes the Fourier transform of $x(t)$, and the Inverse Continuous-Time Fourier Transform (ICTFT) function reconstructs the original signal from its Fourier transform. Subsequently, the code plots three subplots: the original signal $x(t)$, its Fourier transform, and the reconstructed signal. The visualizations illustrate the transformation between time and frequency domains, highlighting the effectiveness of the CTFT and its inverse.

in analyzing and reconstructing signals in the continuous-time domain. This task serves as an insightful demonstration of Fourier transforms and their applications in signal processing.

Task 2 Description:

This MATLAB code addresses two tasks related to Continuous-Time Fourier Transform (CTFT) properties. In the first task, it illustrates the time-shifting property using a signal $x(t) = t \cdot e^{-t^2}$, showcasing the time-shifted signal and the ICTFT of the product of the CTFT and a complex exponential term. In the second task, the code demonstrates the differentiation property using $y(t) = e^{-t^2}$ and displays the derivative. The right subplot exhibits the ICTFT of the product of $j\omega$ and the CTFT of the original signal, confirming the differentiation property. These visualizations offer a concise and intuitive exploration of key CTFT properties.

Verified: _____

Date/Time: _____

Experiment No.13

Objective:

The objective of this lab is to learn how to implement the basic types of filters in MATLAB and to apply them on an audio signal.

Theoretical Background:

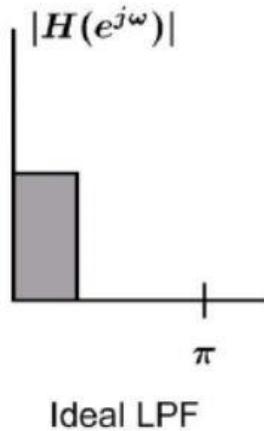
Filters:

Filters, in general, are used to block some part of a signal and to pass some particular part of a signal. In frequency domain, we say that a filter will block a certain range of frequencies, while passing a specific range of frequencies. The three most common types of filters are:

1. Low pass Filters (LPF)
2. Band pass Filters (BPF)
3. High pass Filters (HPF)

1. Low Pass Filter:

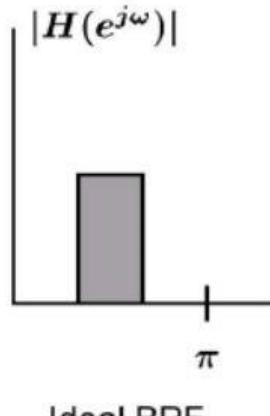
A low pass filter, as its name suggests, is used to pass only the low frequency components of a signal, and block all higher frequency components. The ideal low pass filter is shown below:



Ideal LPF

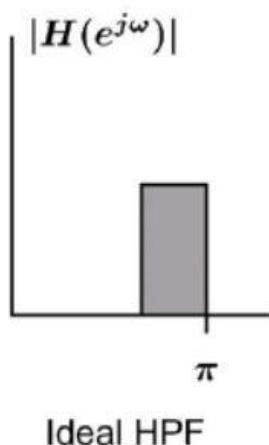
2. Band Pass Filter:

A band pass filter is a filter that passes all frequencies of a signal within a certain range, (which neither includes zero frequency nor pi), and block all other frequencies outside that range. An ideal band pass filter is shown below:



3. High Pass Filter:

A high pass filter is used to pass all frequency components of a signal higher than a cutoff frequency, and stop all other frequency components which are lower than the cutoff frequency. An ideal high pass filter is shown below:



Tasks:

In order to create an understanding of passing signals through filters, the following tasks are to be performed by the students:

Task 1:

Using the filter design toolbox, (FDA tool), design three filters with the following specifications:

- A Low Pass Filter with pass band ($f < 1500$ Hz)
- A Band Pass Filter with pass band ($1500 < f < 3000$ Hz)
- A High Pass Filter with pass band ($f > 3000$ Hz)

Task 2:

- Using audiorecorder, record a 5 second audio in MATLAB
- Export the filters into workspace, and save the filters and recorder object in a ‘.mat’ file.
- In a new MATLAB script, load the ‘.mat’ file into workspace. Now pass the signal through each of the three filters separately.
- Reconstruct the original signal by adding the outputs of all three filters
- Using subplot, show the original signal, the outputs of the three filters, and then the reconstructed signal.

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

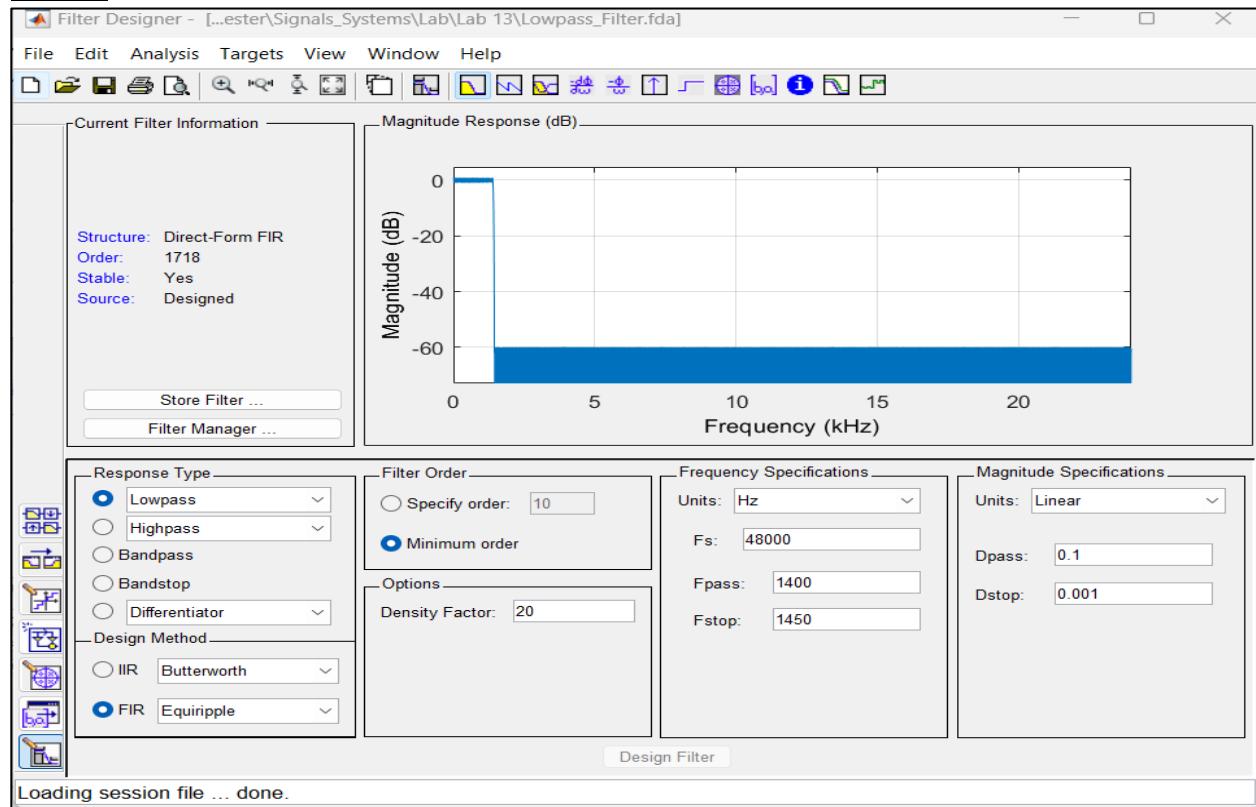


Figure 1: Filter Designer Interface of Low Pass Filter

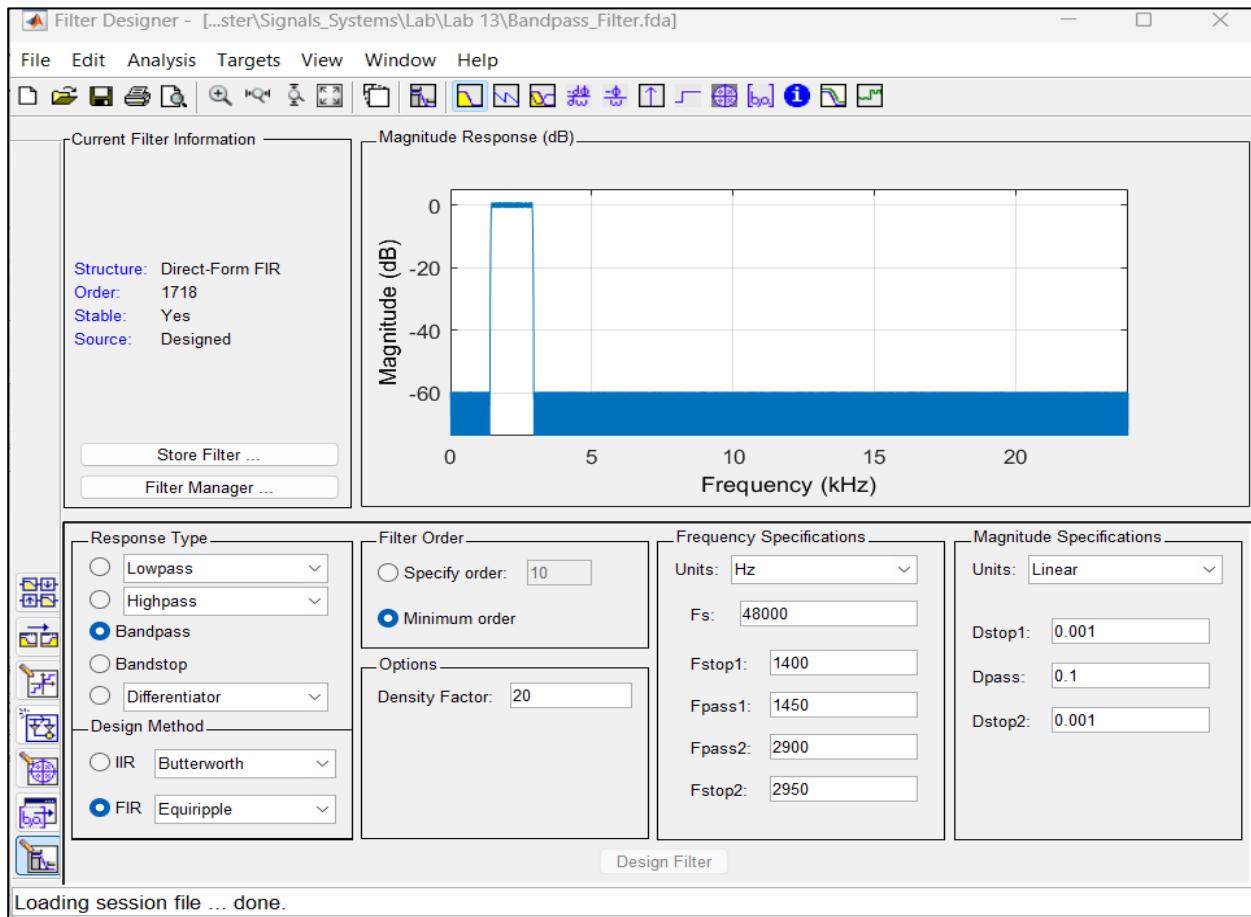


Figure 2: Filter Designer Interface of Band Pass Filter

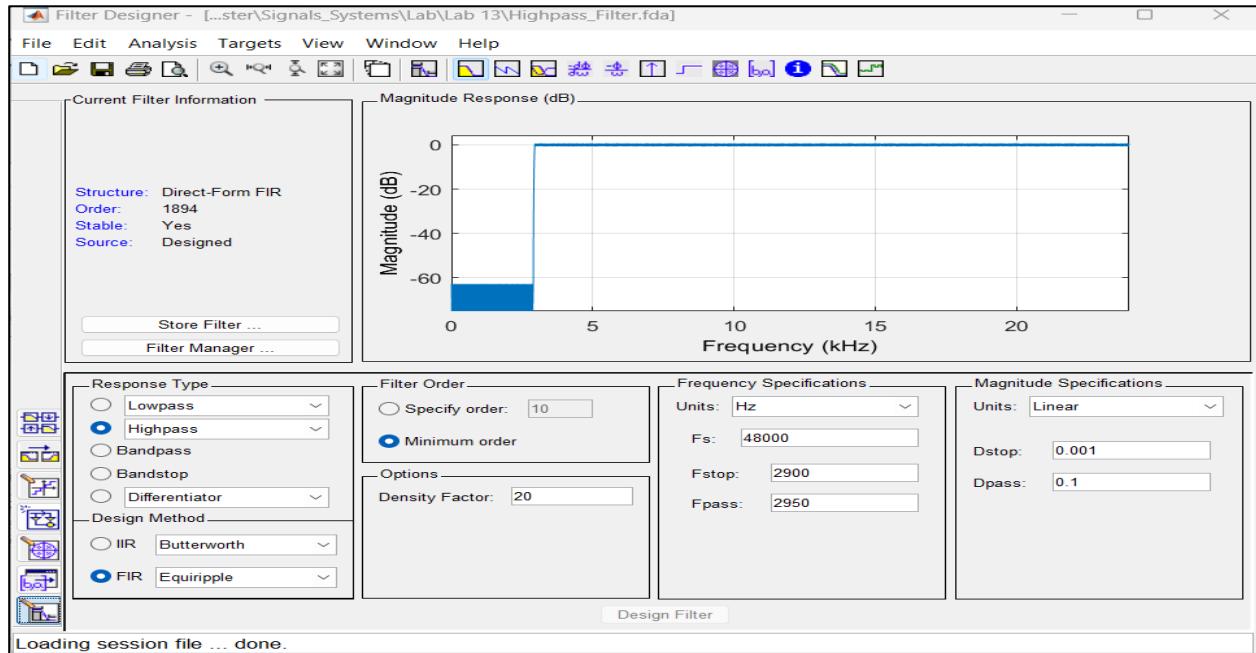


Figure 3: Filter Designer Interface of High Pass Filter

Task 2:

```
% Create an audiorecorder object
recObj = audiorecorder;

% Record 5 seconds of audio
disp('Start speaking.');
Start speaking.

recordblocking(recObj, 2);
disp('End of Recording.');
End of Recording.

% Save the recorded audio to a variable
audioData = getaudiodata(recObj);

% Save the filters and recorder object in a MAT file
save('audio_processing_data.mat', 'LPF', 'BPF', 'HPF', 'recObj');

% Load the MAT file into the workspace
load('audio_processing_data.mat');

% Pass the signal through each filter separately
outputLPF = filter(LPF, audioData);
outputBPF = filter(BPF, audioData);
outputHPF = filter(HPF, audioData);

% Reconstruct the original signal by adding the outputs of all three filters
reconstructedSignal = outputLPF + outputBPF + outputHPF;

% Plot the signals using subplot
figure;

% Original Signal
subplot(4,1,1);
plot(audioData);
title("Original Signal");

% Filtered Signals
subplot(4,1,2);
plot(outputLPF);
title("LPF Output");

subplot(4,1,3);
plot(outputBPF);
title("BPF Output");

subplot(4,1,4);
plot(outputHPF);
title("HPF Output");
```

Figure 4: Task 2 Code to Pass Audio Signal through Lowpass, Bandpass and High pass Filters

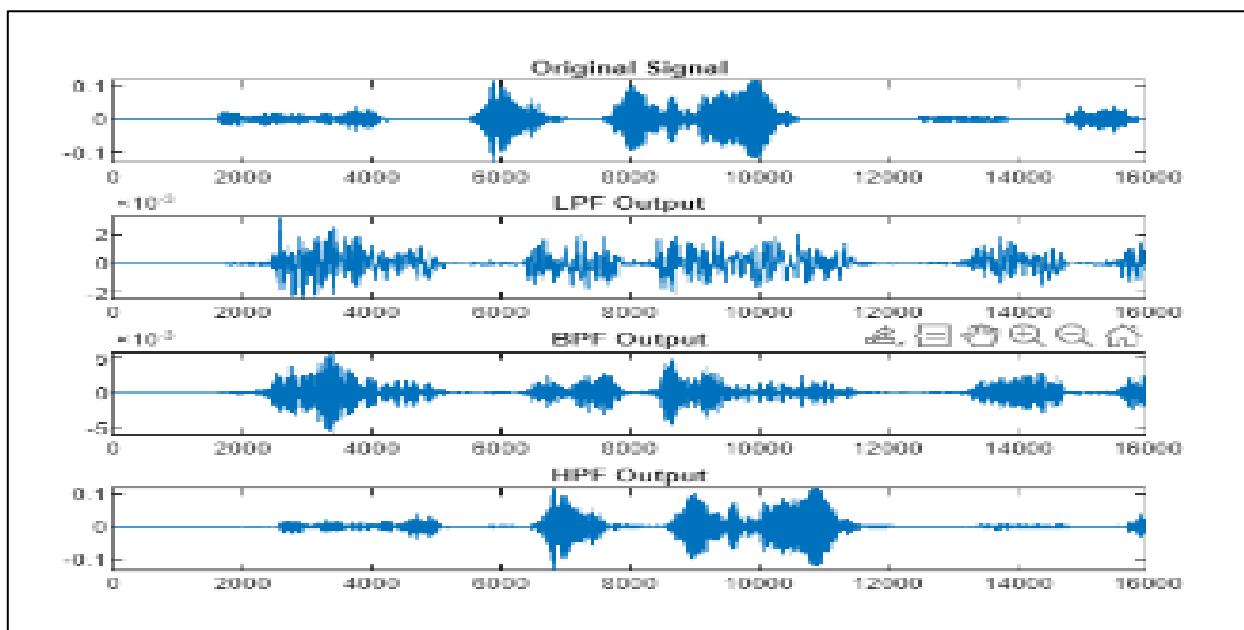


Figure 5: Task 2 Output of Audio Signal and Lowpass, Bandpass and Highpass Filter Outputs

```
% Reconstructed Signal
figure;
subplot(2,1,1);
plot(audioData);
hold on;
plot(reconstructedSignal, 'r');
title('Original Signal and Reconstructed Signal');

subplot(2,1,2);
plot(reconstructedSignal - audioData);
title('Difference (Reconstructed - Original)');
```

Figure 6: Code to plot the Original and Reconstructed Signal and the difference between these signals

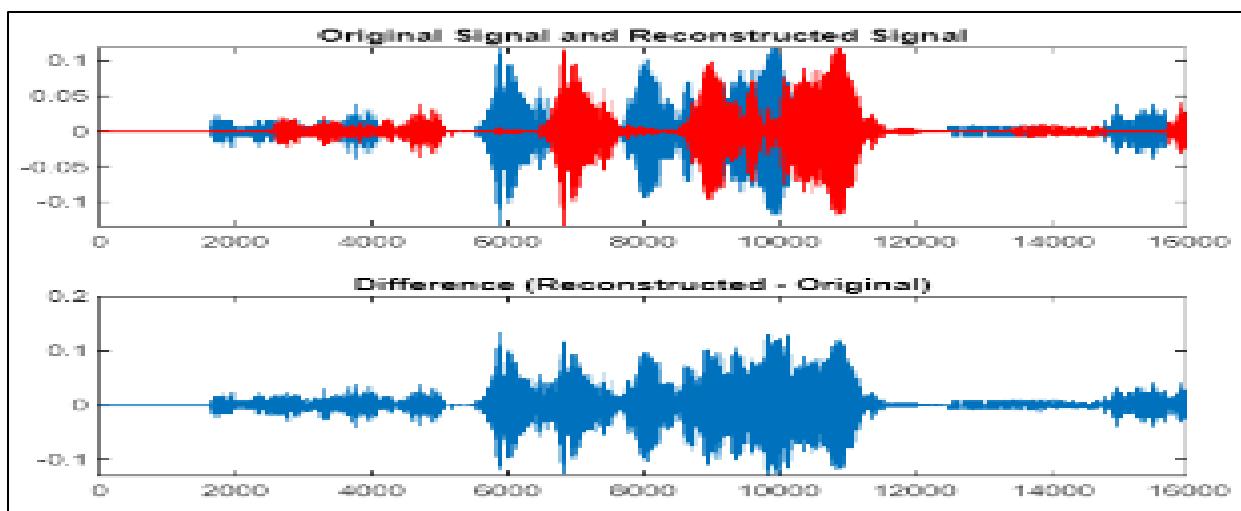


Figure 7: Output of the Original and Reconstructed Signals and their Difference

Task 1 Description:

Instructing students to utilize the Filter Design Toolbox (FDA tool), this task involves the design of three filters to understand signal processing through frequency band specifications. The first filter is a Low Pass Filter allowing frequencies below 1500 Hz, followed by a Band Pass Filter with a pass band between 1500 Hz and 3000 Hz. Lastly, a High Pass Filter is designed to permit frequencies above 3000 Hz. This hands-on exercise provides practical experience in filter design and underscores the importance of tailoring filters to specific frequency bands in signal processing applications.

Task 2 Description:

In Task 2, students record a 5-second audio clip using MATLAB's audiorecorder and export filters and the recorder object to a '.mat' file. In a subsequent script, they load this file, pass the audio through three filters (Low Pass, Band Pass, High Pass), and reconstruct the original signal by summing the filter outputs. Subplot is then employed to visually present the original audio, individual filter outputs, and the reconstructed signal. This hands-on exercise provides a practical understanding of audio signal processing and filter applications.

Verified: _____

Date/Time: _____

Experiment No.14

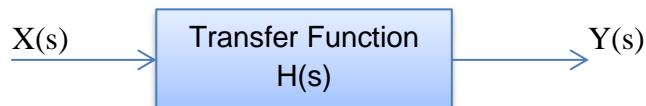
Objective:

In this lab, students will learn how to create transfer functions in MATLAB and Simulink, and to use them further to find step and impulse response of a signal.

Theoretical Background:

Transfer Functions:

A transfer function is a mathematical function relating the output or response of a system, such as a filter, to the input. For example:



Where,

$$Y(s) = H(s) X(s)$$

$$\text{and } H(s) = \frac{Y(s)}{X(s)}$$

The transfer function may be written in z-domain or in s-domain (or Laplace domain).

Tasks:

The following tasks are to be performed by the students:

Task 1:

Using the ‘TF’ and ‘ZPK’ commands, create two transfer functions. Plot the step and impulse response of each transfer function. The transfer functions to be generated are:

Using ‘tf’ command:

$$\frac{s + 1}{6s^5 + 5s^4 + 4s^3 + 3s^2 + 2s + 1}$$

Using ‘zpk’ command:

$$\frac{2(s - 1)(s - 2)}{(s + 1)(s + 2)(s + 3)(s + 4)(s + 5)}$$

Task 2:

In Simulink, take three step functions. Add them and apply a transfer function to the result. View the input and output in scope, and also export the result to workspace. Also view the result in case of one step function only. The transfer function to be applied to the input is:

$$\frac{s + 1}{6s^5 + 5s^4 + 4s^3 + 3s^2 + 2s + 1}$$

INSTRUCTOR VERIFICATION SHEET

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or Professors might ask.

Name: _____

Date of Lab: _____

Task 1:

```
% Transfer function using 'tf' command
num_tf = [1, 1];
denum_tf = [6, 5, 4, 3, 2, 1];
TF_tf = tf(num_tf, denum_tf);

% Transfer function using 'zpk' command
zeros_zpk = [1, 2];
poles_zpk = [-1, -2, -3, -4, -5];
gain_zpk = 2;
TF_zpk = zpk(zeros_zpk, poles_zpk, gain_zpk);

% Plot step and impulse responses for the transfer function created using 'tf'
figure;
subplot(2,2,1);
step(TF_tf);
title('Step Response (TF using tf)');

subplot(2,2,2);
impulse(TF_tf);
title('Impulse Response (TF using tf)');

% Plot step and impulse responses for the transfer function created using 'zpk'
subplot(2,2,3);
step(TF_zpk);
title('Step Response (TF using zpk)');

subplot(2,2,4);
impulse(TF_zpk);
title('Impulse Response (TF using zpk)');
```

Figure 1: Task 1 Code

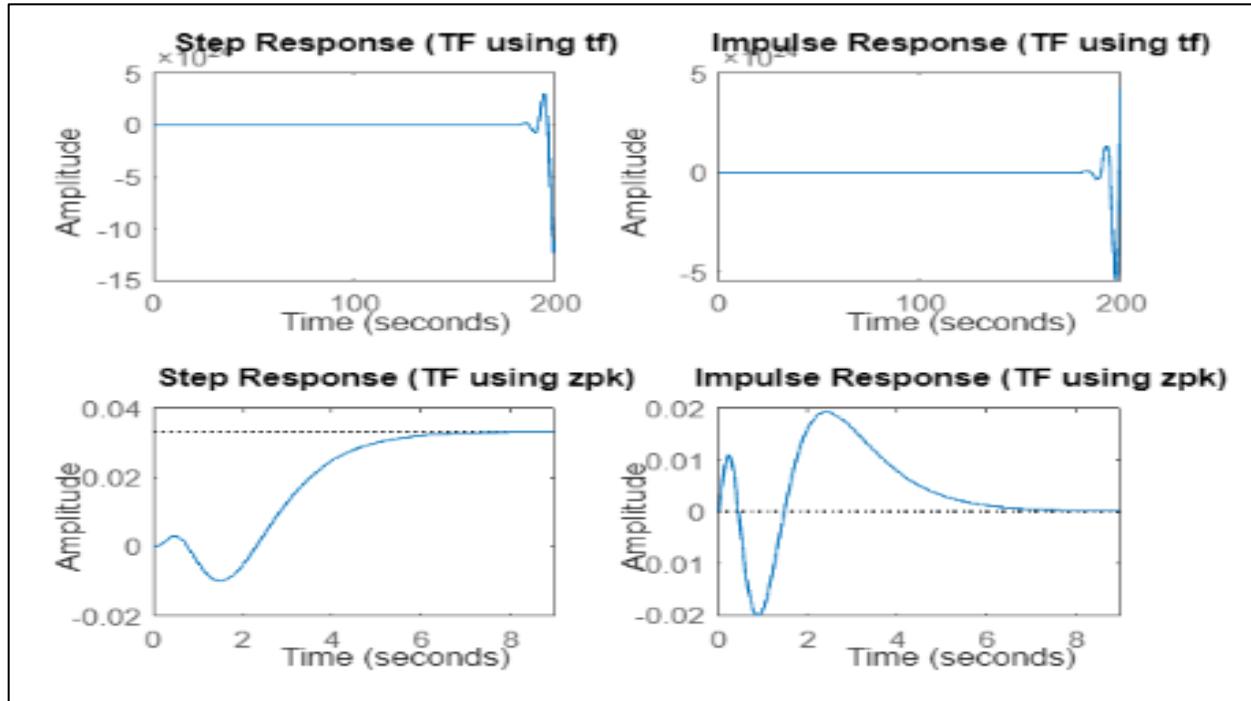


Figure 2: Task 1 Output

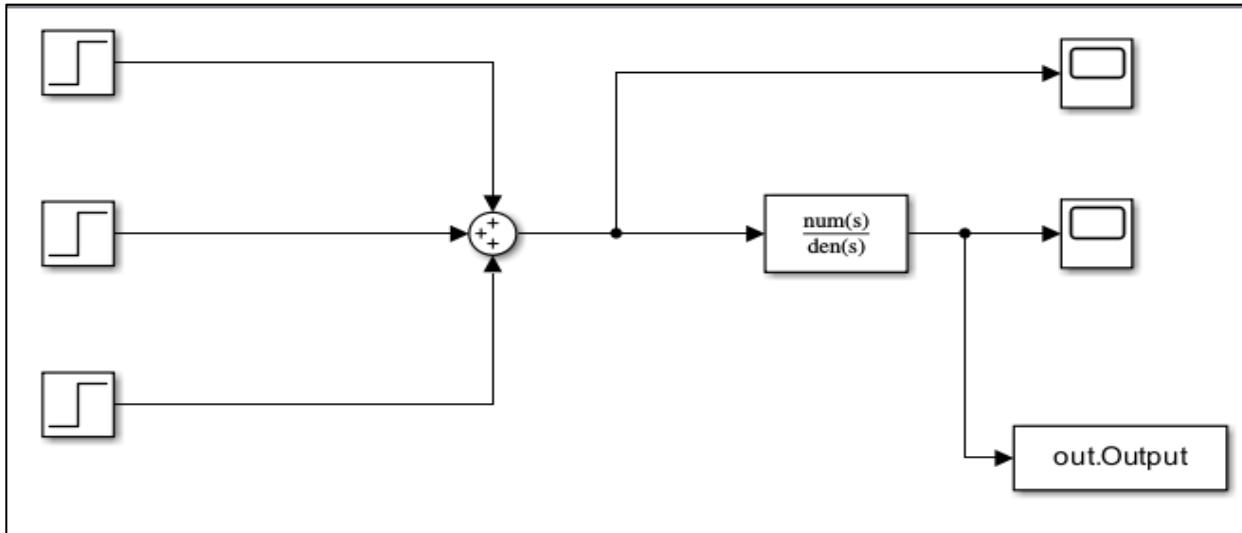
Task 2:

Figure 3: Simulink Model to pass 3 step functions through transfer function

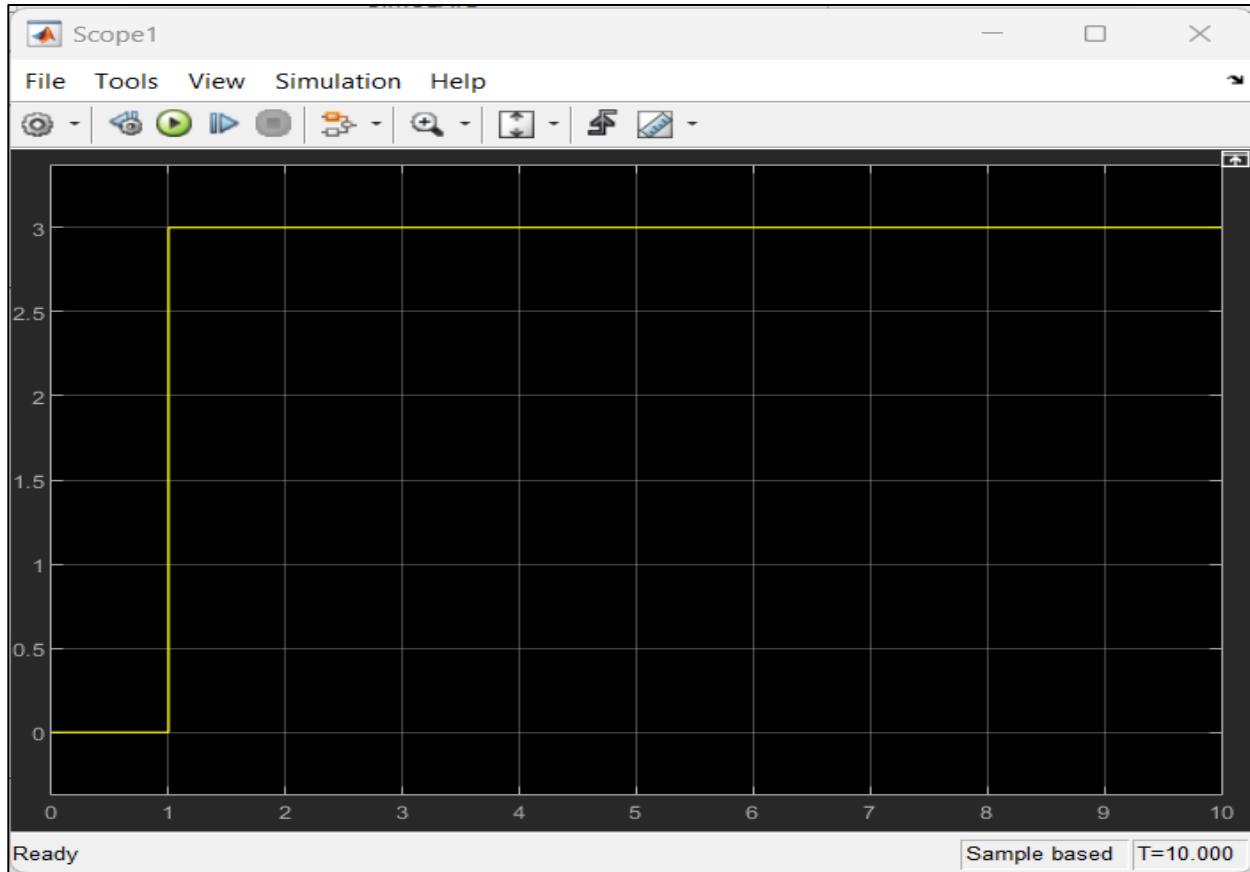


Figure 4: Scope for 3 step functions

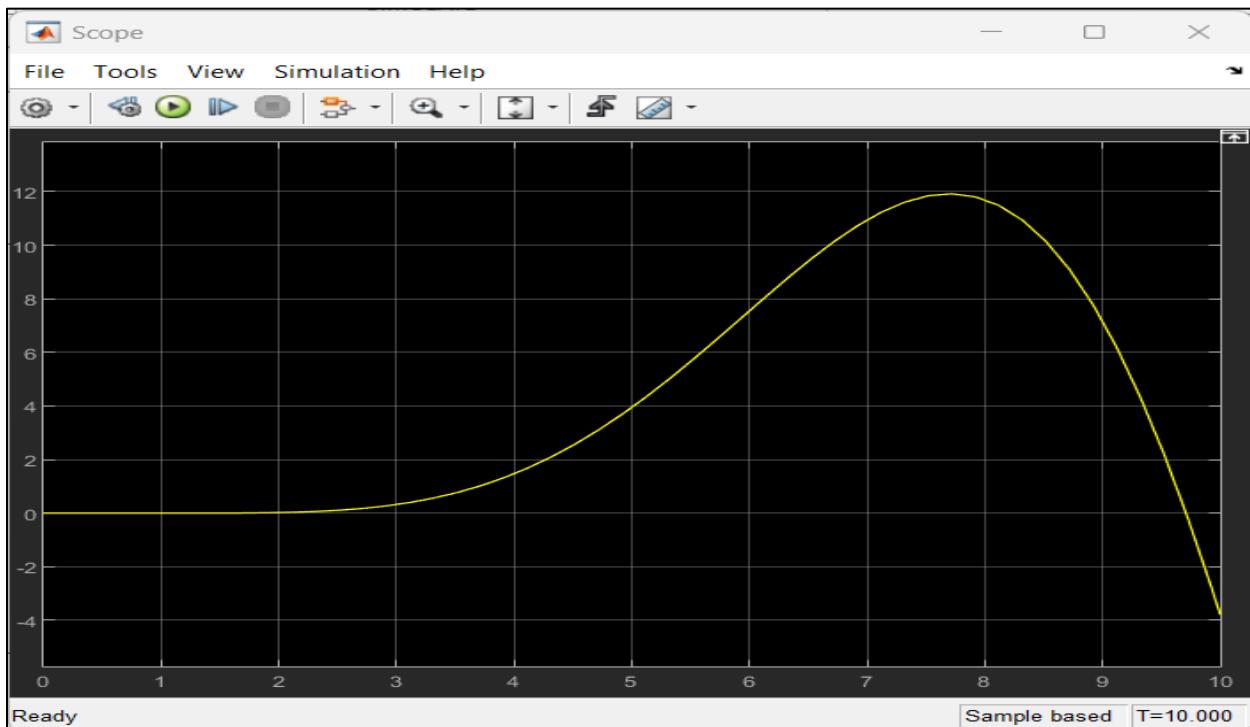


Figure 5: Output of passing 3 step functions through transfer function

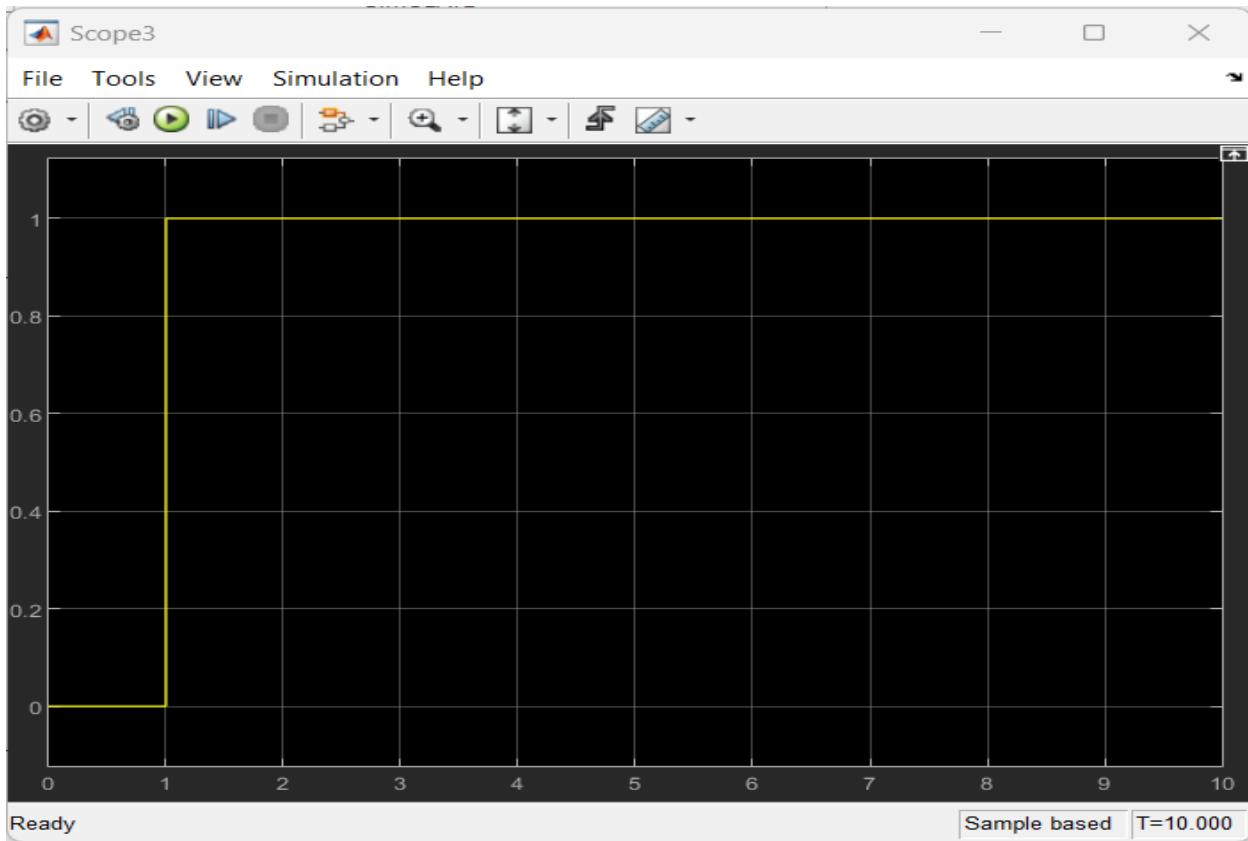


Figure 6: Step Function Scope

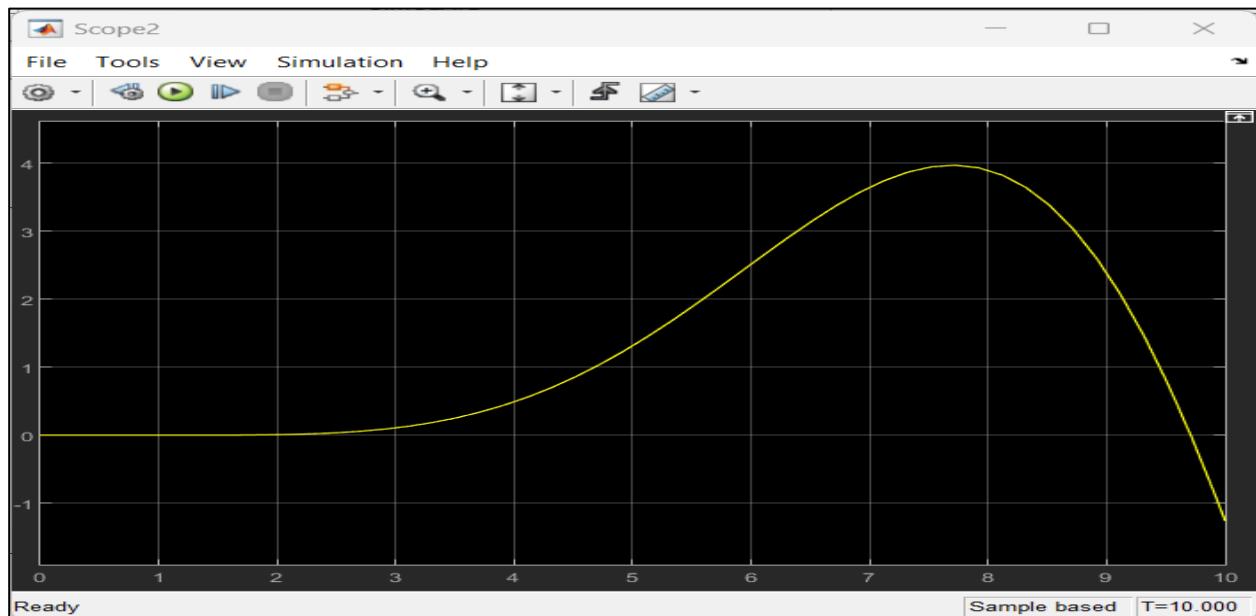


Figure 7: Output of passing step function through transfer function

Task 1 Description:

In Task 1, students are tasked with creating two transfer functions using the 'TF' (Transfer Function) and 'ZPK' (Zero-Pole-Gain) commands in MATLAB. The transfer functions to be generated are specified for both commands. Using 'tf', the task requires the creation of a transfer function represented by the expression:

$$\frac{s + 1}{6s^5 + 5s^4 + 4s^3 + 3s^2 + 2s + 1}$$

Similarly, using 'zpk', students must create a transfer function expressed as:

$$\frac{2(s - 1)(s - 2)}{(s + 1)(s + 2)(s + 3)(s + 4)(s + 5)}$$

The subsequent step involves plotting the step and impulse responses for both transfer functions using the 'step' and 'impulse' functions in MATLAB. This task provides students with hands-on experience in creating and analyzing transfer functions using different commands in MATLAB, fostering a deeper understanding of system dynamics.

Task 2 Description:

Task 2 instructs students to use Simulink for signal processing by combining three step functions and applying a transfer function to the result. The specified transfer function,

$$\frac{s + 1}{6s^5 + 5s^4 + 4s^3 + 3s^2 + 2s + 1}$$

is implemented in Simulink, and both the input (sum of step functions) and output are visualized in the Simulink scope. The task further requires exporting the result to the MATLAB workspace for additional analysis. Students are also prompted to examine the result when only one step function is applied, providing a comprehensive hands-on experience in signal processing using Simulink.

Verified: _____

Date/Time: _____