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

For

B.E. Electrical Engineering

Group Members			
Azlaan Ranjha			
Laiba Jabbar			
Bilal Younas Chaudhary			
Rana Abdullah Azhar			
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	Continous 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 **mat**rix **lab**oratory. 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 . . . 0] with fifty 0's.
E= [1 1 1 . . . 1] with a hundred 1's.

(d) Generate the following vectors using the colon operator

F= [1 2 3 4 . . . 30]

G= [25 22 19 16 13 10 7 4 1]
H= [0 0.2 0.4 0.6 . . . 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 1^{st}, 3^{rd}, 5^{th}, 7^{th}, and 9^{th} 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:

Part a:

```
A=[1 0 4 5 3 9 0 2]
```

```
A = 1×8
     1     0     4     5     3     9     0     2
```

```
a=[4 5 0 2 0 0 7 1]
```

```
a = 1×8
     4     5     0     2     0     0     7     1
```

Part b:

```
B=[A a]
```

```
B = 1×16
     1     0     4     5     3     9     0     2     4     5     0     2     0     0     7     1
```

```
C=[a , A]
```

```
C = 1×16
     4     5     0     2     0     0     7     1     1     0     4     5     3     9     0     2
```

Part c:

```
D = zeros(1,50)
```

```
D = 1×50
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0   ...
```

```
E = ones(1,100)
```

```
E = 1×100
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1   ...
```

Part d:

```
F=1:30
```

```
F = 1×30
     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30
```

```
G = 25:-3:1
```

```
G = 1×9
    25    22    19    16    13    10     7     4     1
```

```
H = 0:0.2:2.0
```

```
H = 1×11
         0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

Task 2:

Variable Creation:

```
V1 = [1:9,0];
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];
```

Part a:

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

```
Addition = 1×10
    5.3000    7.2000    7.5000   10.1000    8.1000    9.4000   12.6000   14.2000   12.7000    1.9000
```

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:

```
V1 = [1:9,0];
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];
```

Part a:

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

Part b:

```
V1fifth = V1(1,5)
V2fifth = V2(1,5)
V3fifth = V3(1,5)
%V1(0)
%V1(11)
```

Part c:

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

Part d:

```
V20 = V2
V20(6) = []
V6 = V20
V7 = V2
V7(1,7) = 1.4
V8 = V2([1 3 5 7 9])
```

Part e:

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

```
9 - V1
```

Each element is multiplied by 5.

```
V1*5
```

Vector V1 is being added to Vector V2.

```
V1+V2
```

Vector V3 is subtracted from vector V1.

```
V1-V3
```

Element-wise multiplication of vector V1 and vector V2.

```
V1.*V2
```

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

```
%V1*V2
```

Each element is raised to the power of two.

```
V1.^2
```

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

```
V1.^V3
```

A matrix power cannot be another matrix.

```
%V1^V3
```

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.

```
V1 == V3
```

This element checks all elements in V1 and returns 1 for all those greater than 6. Otherwise it is a 0.

```
V1>6
```

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.

```
V1>V3
```

First, a vector is created with the answers of the operation (V1>2) and then they are subtracted from the vector V3.

```
V3-(V1>2)
```

It checks the elements of V1 so that they are greater than 2 but less than 6.

```
(V1>2) & (V1<6)
```

It checks the elements of V1 so that they are either greater than 2 or less than 6.

```
(V1>2) | (V1<6)
```

'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.

```
any(V1)
```

'all' function checks whether all elements in a vector or matrix are non-zero. If yes, then its returns a single 1. Otherwise 0.

```
all(V1)
```

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 7^{th} 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:

```
first_30 = [];

for i = 1:30
    first_30(i) = i;
end

display(first_30, 'first_30')
```

```
first_30 = 1×30
     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20    21    22    23    24    25    26    27    28    29    30
```

```
first_30_even = [];
a = 1;
for i = 0:100;
    if mod(i, 2) == 0
        first_30_even(a) = i;
        a = a+1;
    elseif size(first_30_even,2) == 30
        break
    end
end

display(first_30_even, 'first_30_even')
```

```
first_30_even = 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
```

```
first_30_even = [];
a = 1;
i = 0;

while i <= 100;
    if mod(i, 2) == 0
        first_30_even(a) = i;
        a = a+1;
    elseif size(first_30_even,2) == 30
        break
    end
    i = i+1;
end

display(first_30_even, 'first_30_even')
```

```
first_30_even = 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
```

```
first_30_odd = [];
a = 1;
for i = 0:100;
    if mod(i, 2) == 0
        first_30_odd(a) = i;
        a = a+1;
    elseif size(first_30_odd,2) == 30
        break
    end
end

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
```

```
first_30_odd = [];
a = 1;
for i = 0:100;
    if mod(i, 2) == 1
        first_30_odd(a) = i;
        a = a+1;
    elseif size(first_30_odd,2) == 30
        break
    end
end

display(first_30_odd, 'first_30_odd')
```

```
first_30_odd = 1×30
     1     3     5     7     9    11    13    15    17    19    21    23    25    27    29    31    33    35    37    39    41    43    45    47    49    51    53    55    57    59
```

```
first_30_odd = [];
a = 1;
i = 0;

while i <= 100;
    if mod(i, 2) == 1
        first_30_odd(a) = i;
        a = a+1;
    elseif size(first_30_odd,2) == 30
        break
    end
    i = i+1;
end

display(first_30_odd, 'first_30_odd')
```

```
first_30_odd = 1×30
     1     3     5     7     9    11    13    15    17    19    21    23    25    27    29    31    33    35    37    39    41    43    45    47    49    51    53    55    57    59
```

__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:__

Task 2:

```
for i = 1:20
    H(i) = i*5;
end

H
```

```
H = 1×20
     5    10    15    20    25    30    35    40    45    50    55    60    65    70    75    80    85    90    95   100
```

```
H = 1:20;
H = H * 5;

H
```

```
H = 1×20
     5    10    15    20    25    30    35    40    45    50    55    60    65    70    75    80    85    90    95   100
```

```
for n = 1:100
    x(n) = sin(n*pi/10);
end

x
```

```
x = 1×100
    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511    0.8090    0.5878    0.3090    0.0000   -0.3090   -0.5878   -0.8090   -0.9511   -1.0000   -0.9511   -0.8090   -0.5878   -0.3090   -0.0000 ...
```

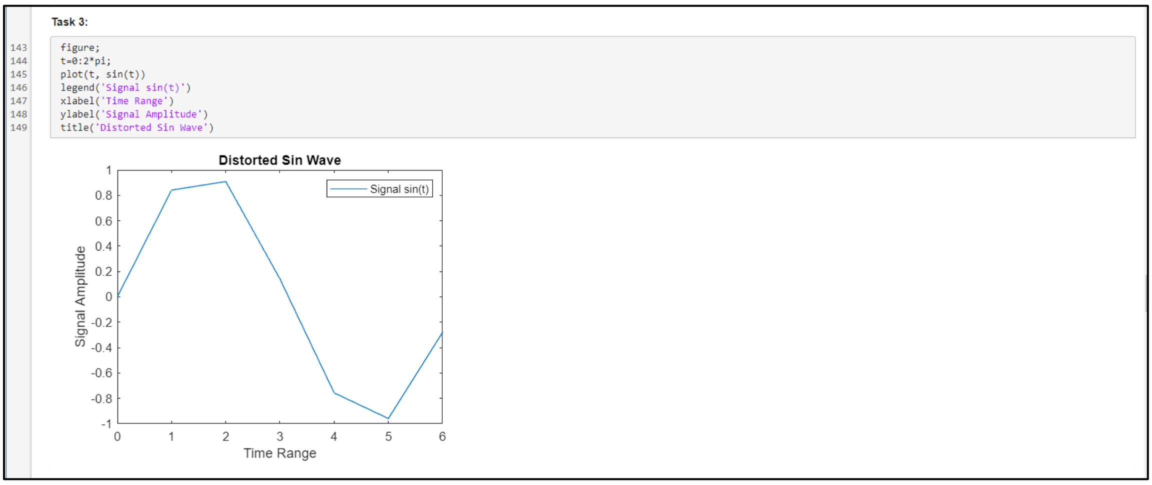
```
n = 1:100;
x = sin(n*pi/10);

x
```

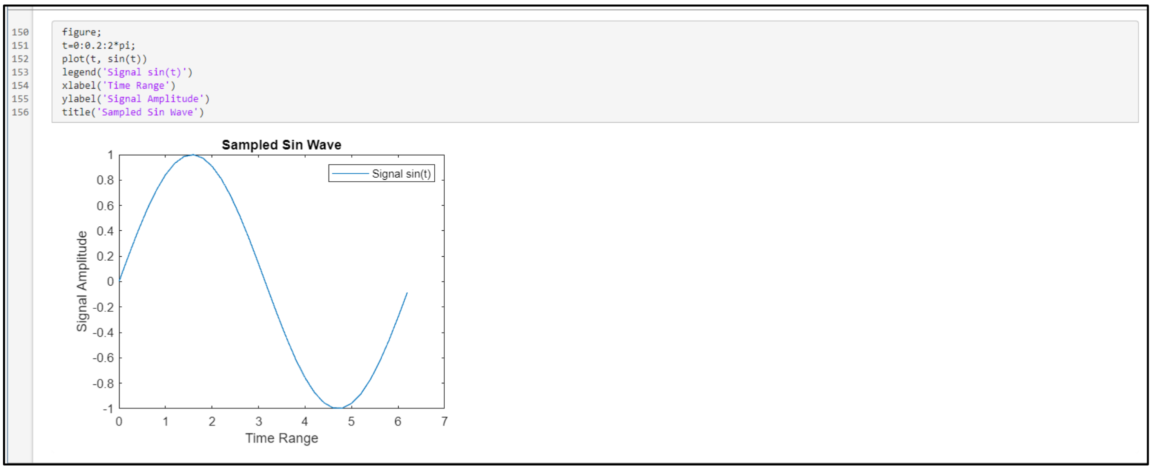
```
x = 1×100
    0.3090    0.5878    0.8090    0.9511    1.0000    0.9511    0.8090    0.5878    0.3090    0.0000   -0.3090   -0.5878   -0.8090   -0.9511   -1.0000   -0.9511   -0.8090   -0.5878   -0.3090   -0.0000 ...
```

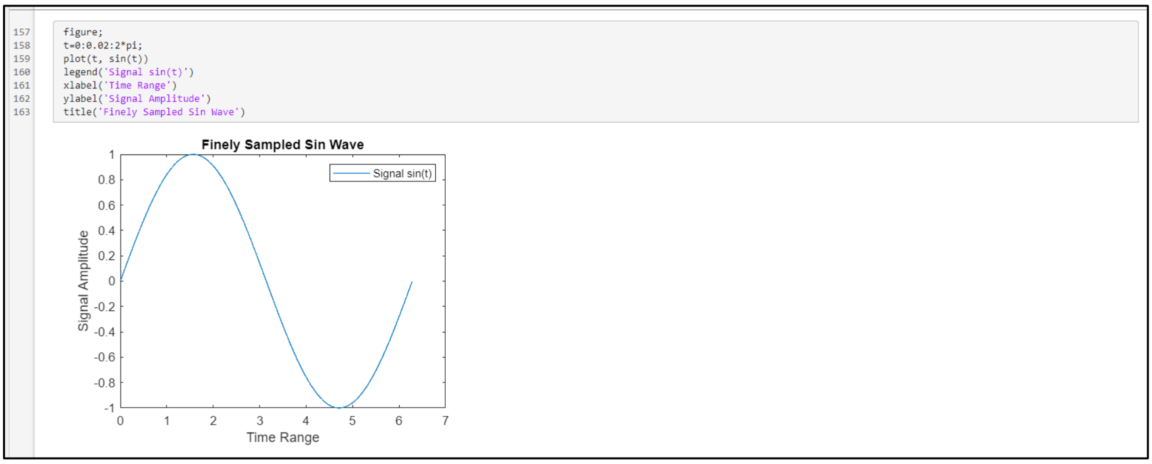
__Task 2 Description:__

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.

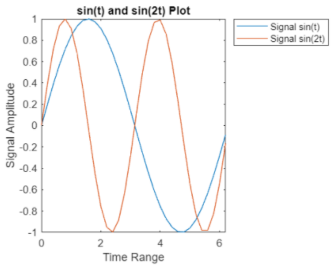
<u>Task 4:</u>

Task 4:

```
t = 0:0.2:2*pi
```

```
t = 1×32
        0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000    2.2000    2.4000    2.6000    2.8000    3.0000    3.2000    3.4000    3.6000    3.8000    4.0000    4.2000    4.4000    4.6000    4.8000 ...
```

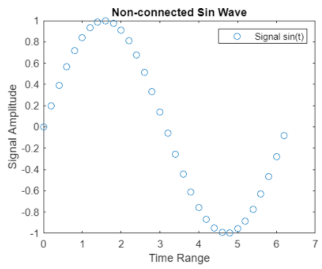
```
plot(t,sin(t))
hold on
plot(t, sin(2*t))
hold off
legend('Signal sin(t)','Signal sin(2t)','Location','northeastoutside')
xlabel('Time Range')
ylabel('Signal Amplitude')
title('sin(t) and sin(2t) Plot')
```



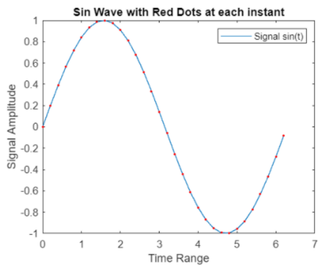

```
t = 0:0.2:2*pi
```

```
t = 1×32
        0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000    2.2000    2.4000    2.6000    2.8000    3.0000    3.2000    3.4000    3.6000    3.8000    4.0000    4.2000    4.4000    4.6000    4.8000 ...
```

```
plot(t,sin(t),'o')
legend('Signal sin(t)')
xlabel('Time Range')
ylabel('Signal Amplitude')
title('Non-connected Sin Wave')
```




```
t = 0:0.2:2*pi;
plot(t,sin(t), t, sin(t),'r.')
legend('Signal sin(t)')
xlabel('Time Range')
ylabel('Signal Amplitude')
title('Sin Wave with Red Dots at each instant')
```



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: ______________

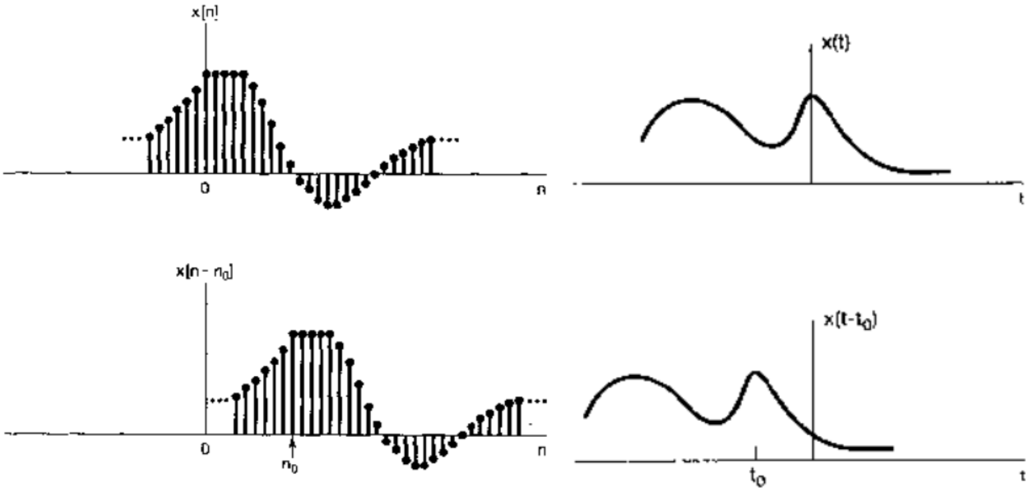
Experiment No. 3

Objective:

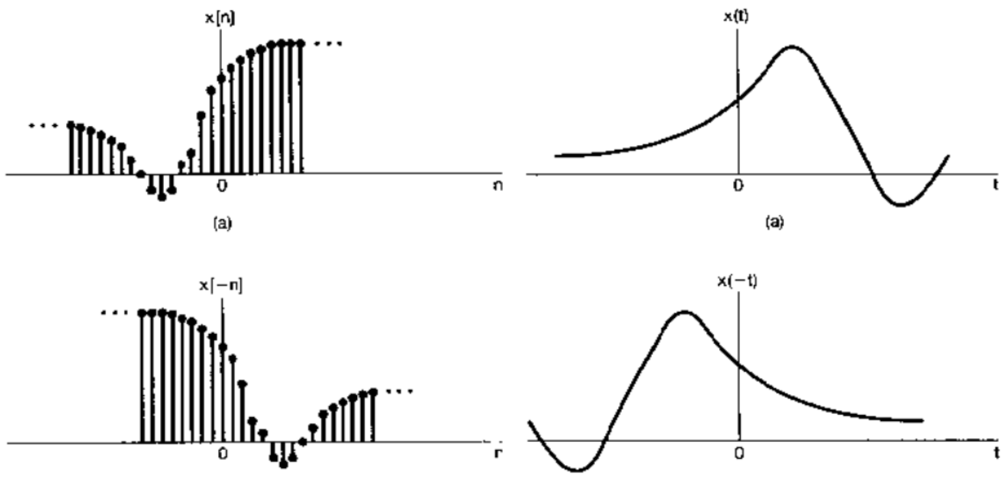
The objective of this lab is to practice some basic operations performed on a signal like shifting and flipping. Also, students will learn how to prove the stability and causality properties of a signal in MATLAB. An introduction to convolution is also included in this lab.

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_o, making the signal $x[n-n_o]$, 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.

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

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: ______________

Task 1:

Task 2:

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 2^{nd} order equations using symbolic variables and solve them simultaneously. Make five sets of equations.

Task 3 :

Declare five 5^{th} 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: ______________

Task 1:

Task 2:

Task 3:

Task 4:

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 δ[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: ______________

Task 1:

Task 2:

Task 3:

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.

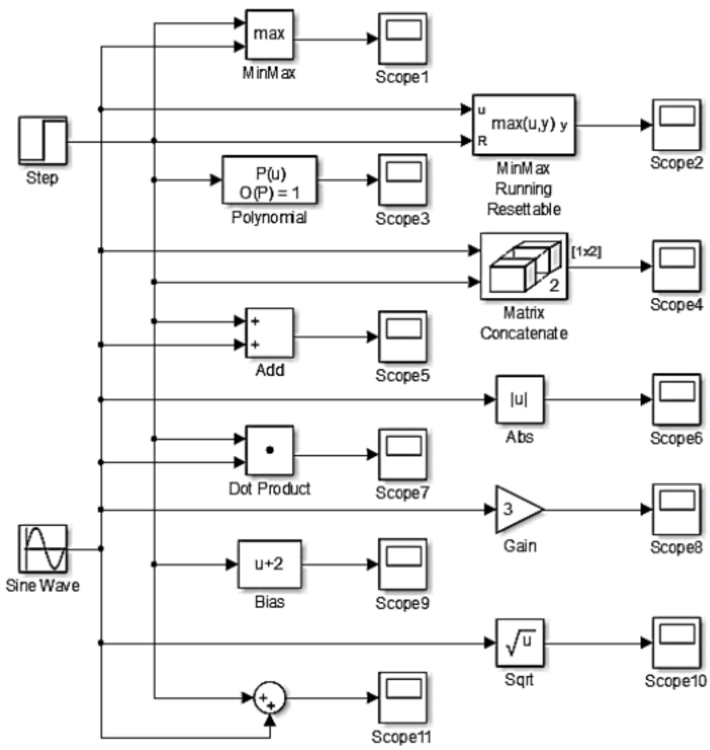
Theoretical Background:

Simulinkis 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.

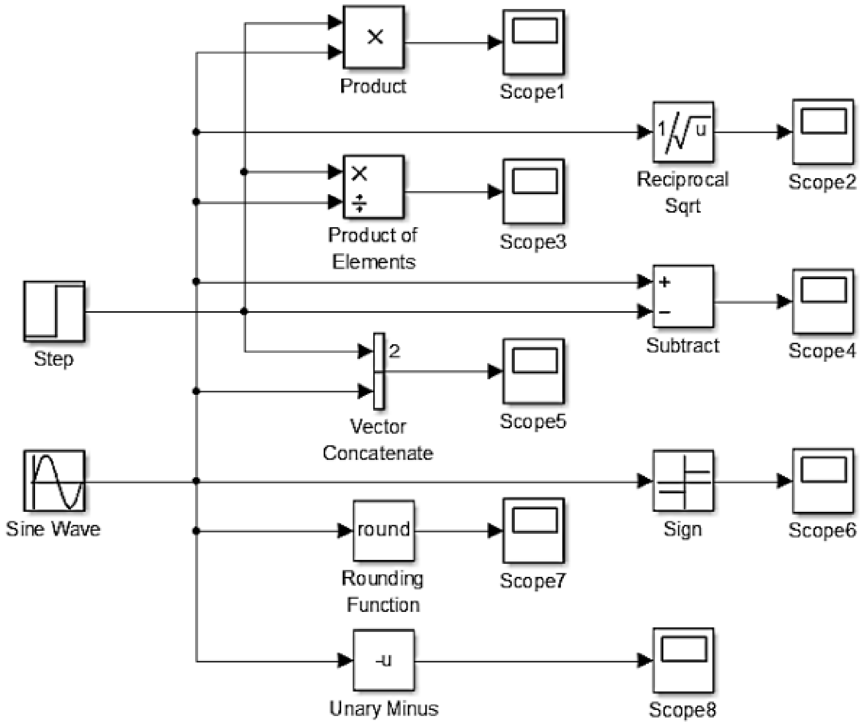
Tasks:

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.

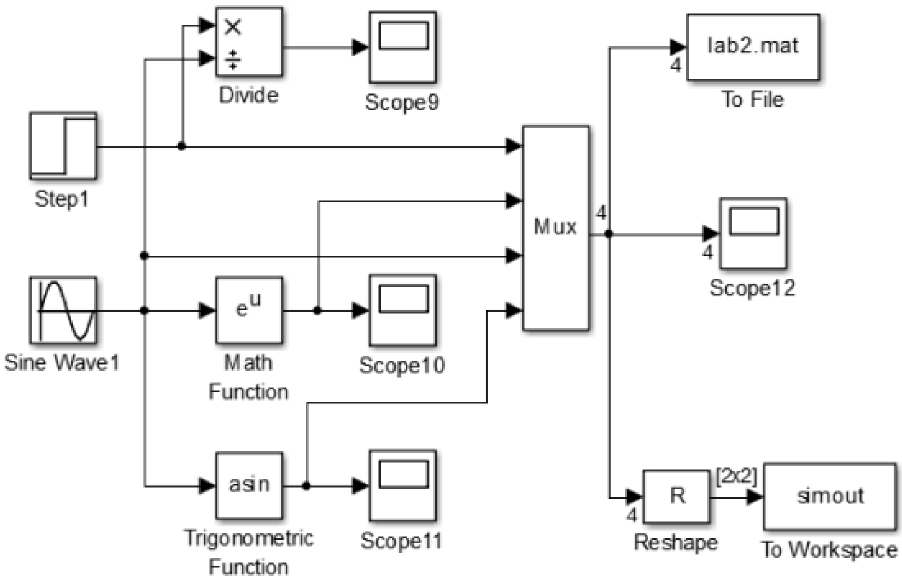
Task 1:



Task 2:


Product
Scope1
1/√u
Reciprocal Sqrt
Scope2
Product of Elements
Scope3
Step
+
−
Subtract
Scope4
2
Vector Concatenate
Scope5
Sine Wave
Sign
Scope6
round
Rounding Function
Scope7
-u
Unary Minus
Scope8

Task 3:


Divide
Scope9
lab2.mat
To File
Step1
Mux
4
Scope12
Sine Wave1
e^u
Math Function
Scope10
asin
Trigonometric Function
Scope11
R
Reshape
[2x2]
simout
To Workspace

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 2:

Task 3:

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 δ[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.

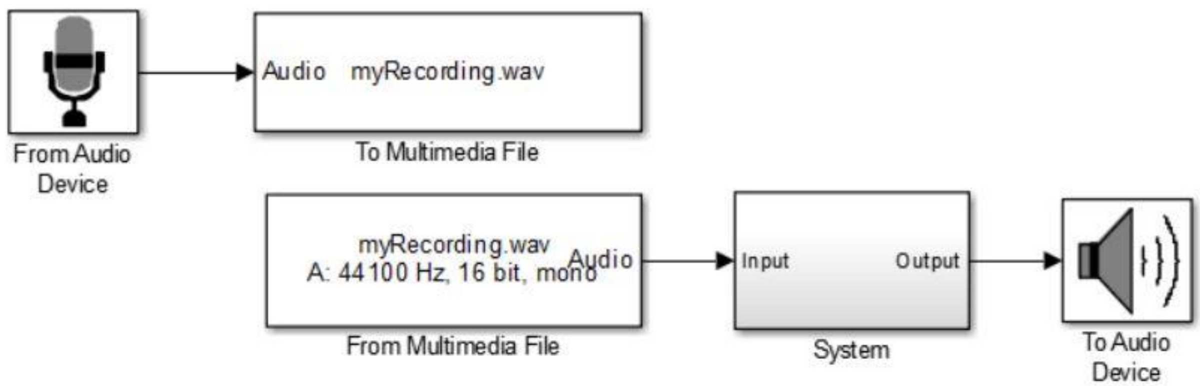
- Play the sound file in the following ways:
    - Complete file
    - First half and second half of the file separately
    - Middle one-third of the file

Task 3:

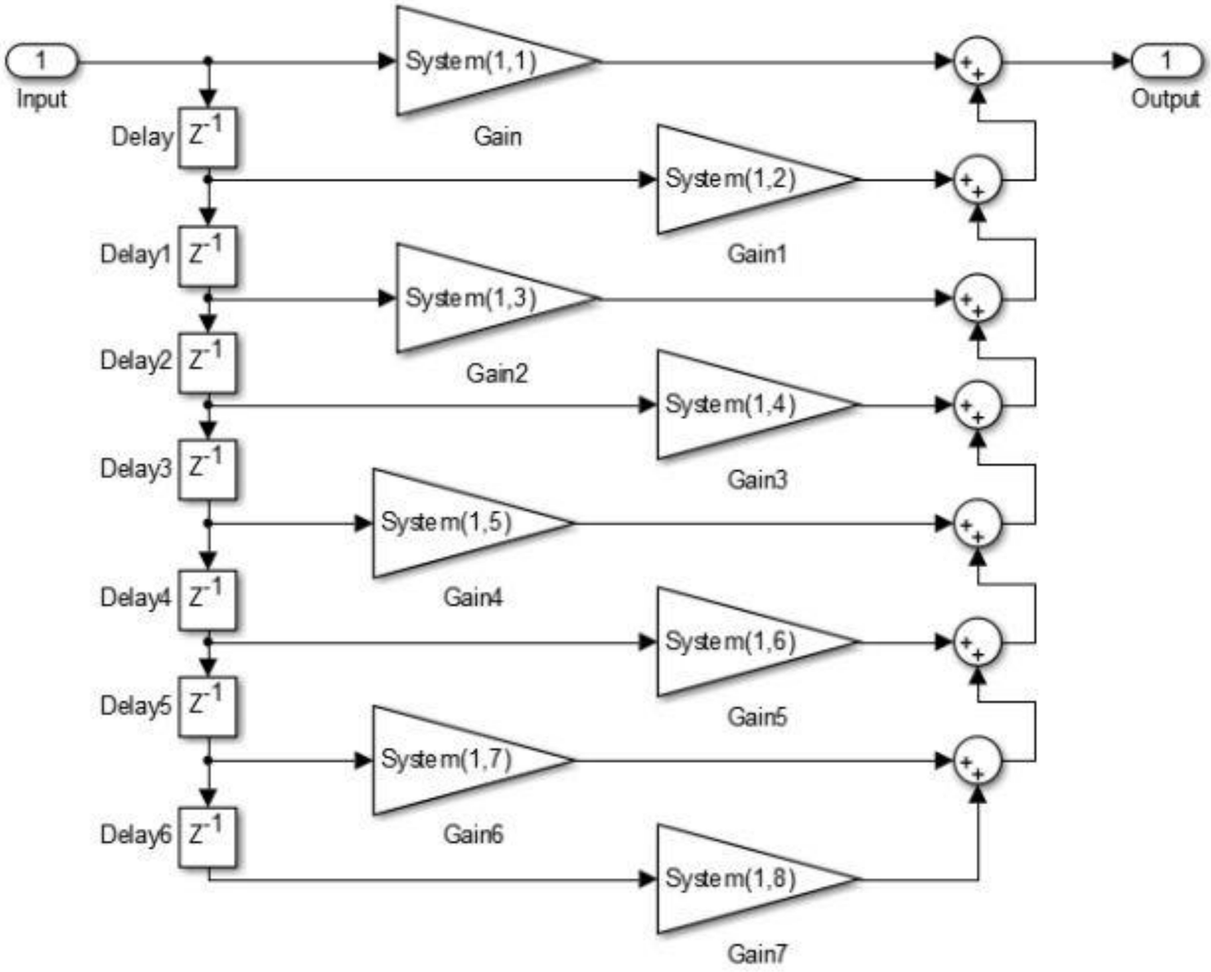
Perform convolution of an audio file imported in Simulink with a system defined below. Plot the audio signal before and after passing through the system using function callback (File -> Model properties -> Callbacks) The system is defined as:

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

The block diagram to be created by students are:



The system block diagram is to be created as:


1
Input
System(1,1)
Gain
1
Output
Delay
Z^{-1}
System(1,2)
Gain1
Delay1
Z^{-1}
System(1,3)
Gain2
Delay2
Z^{-1}
System(1,4)
Gain3
Delay3
Z^{-1}
System(1,5)
Gain4
Delay4
Z^{-1}
System(1,6)
Gain5
Delay5
Z^{-1}
System(1,7)
Gain6
Delay6
Z^{-1}
System(1,8)
Gain7

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 2:

Task 3:

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.

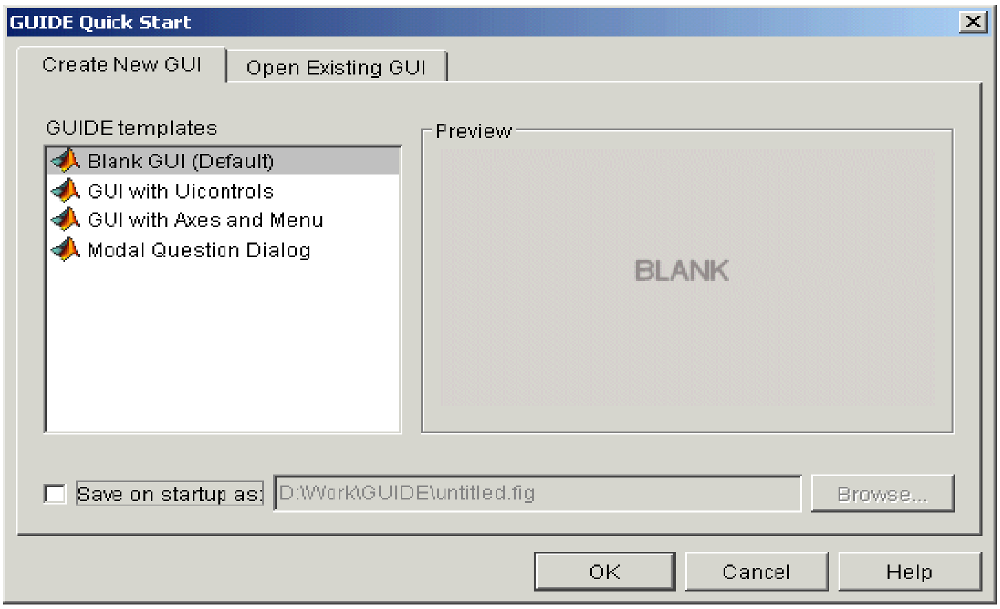
You can combine the two approaches to some degree. You can create a GUI with GUIDE and then modify it programmatically. However, you cannot create a GUI programmatically and later modify it with GUIDE.

Starting GUIDE

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

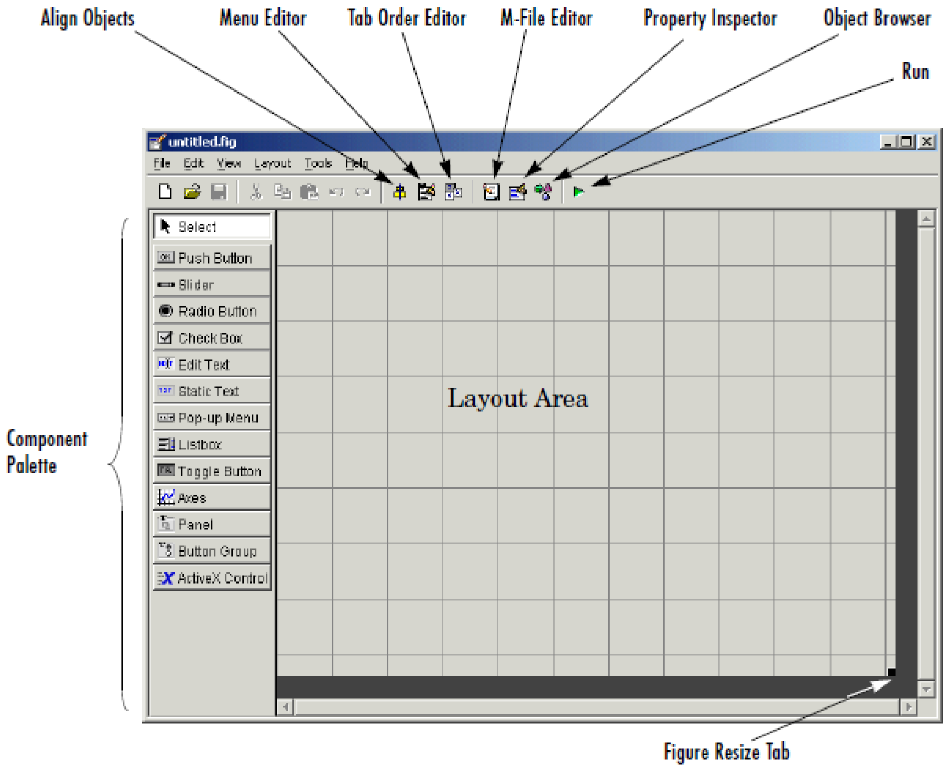
- Command line by typing guide
- **Start** menu by selecting **MATLAB** > **GUIDE (GUI Builder)**
- **MATLAB File** menu by selecting **New** > **GUI**
- MATLAB toolbar by clicking the **GUIDE** button

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



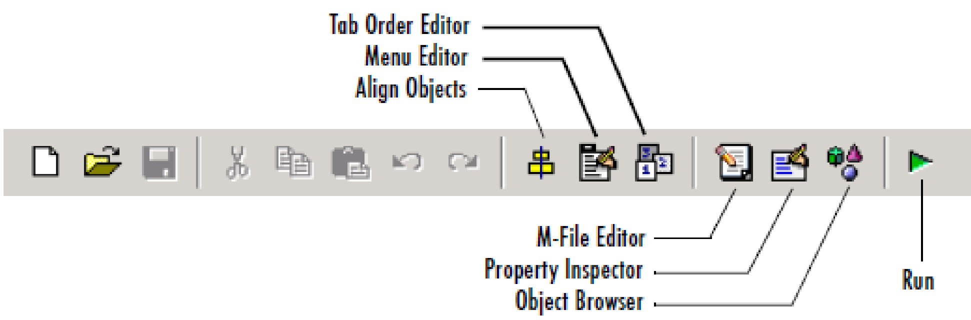
GUIDE Tools Summary

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.



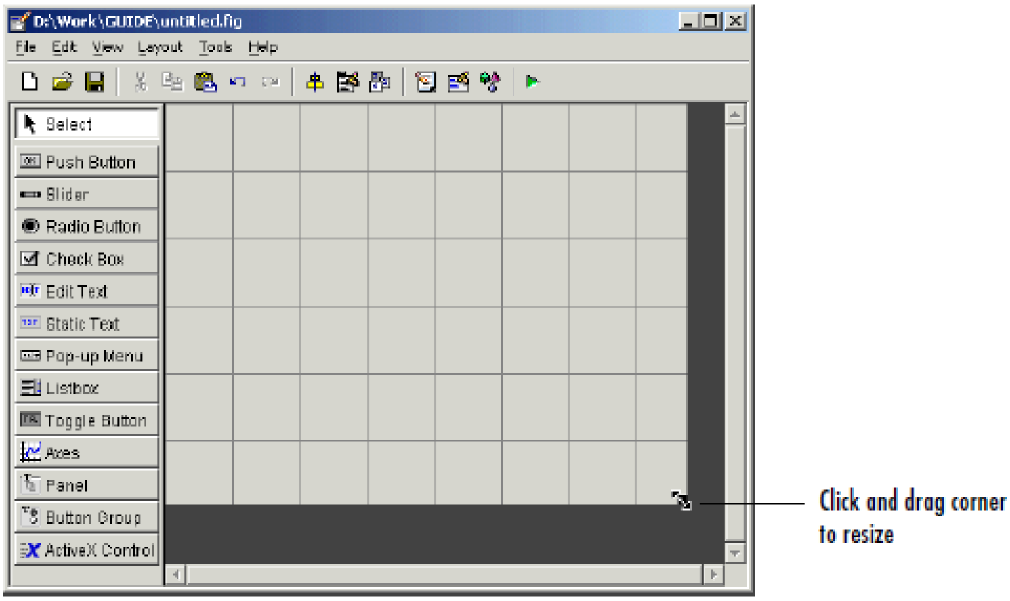
Show Toolbar

Displays the following toolbar in the Layout Editor window.



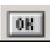
Setting the GUI Size

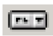
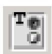
Set the size of the GUI by resizing the grid area in the Layout Editor. Click the lower-right corner and drag it until the GUI is the desired size. If necessary, make the window larger.



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. When you do so, the button returns to the raised
Radio Button		Radio buttons are similar to check boxes, but radio buttons are typically mutually exclusive within a ngroup of related radio buttons. That is, when you select one button the previously seleted button is deselected. To activate a radio button, click the mouse button on the object. The display indicates the state of
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

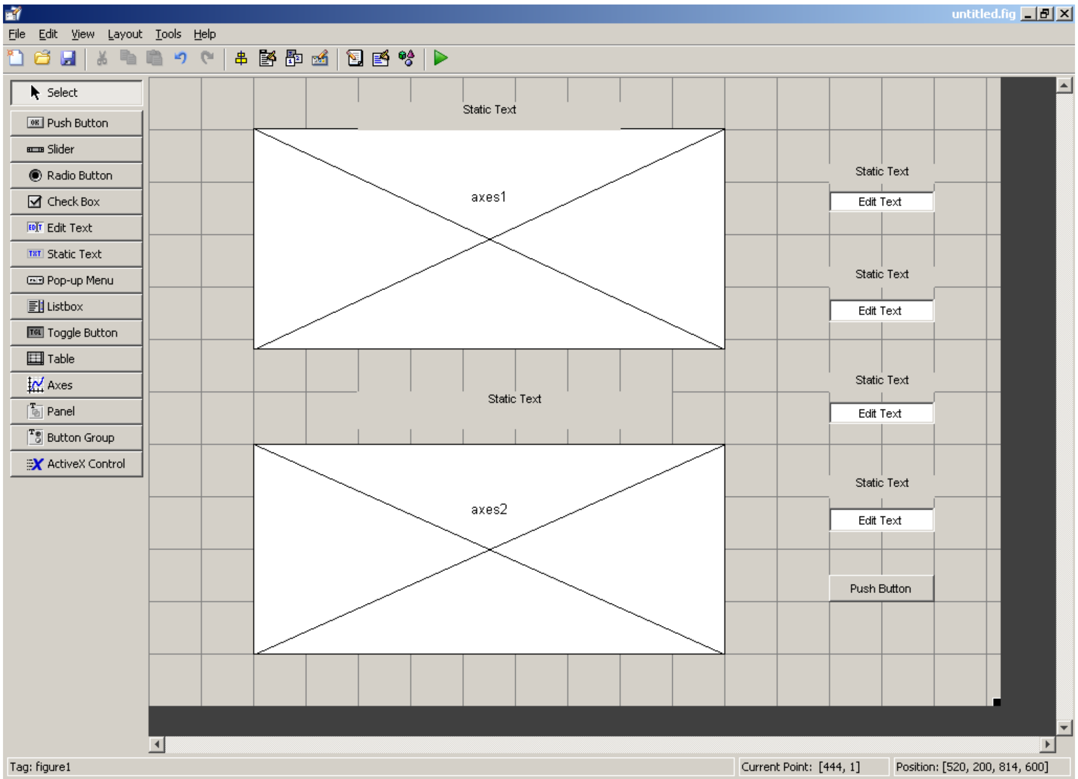
The GUI M-file that GUIDE generates is a function file. The name of the main function is the same as the name of the M-file. For example, if the name of the M-file is mygui.m, then the name of the main function is mygui. Each callback in the file is a subfunction of the main function. When GUIDE generates 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 theGUI M-file are ordered as shown in the following table.

Task 1

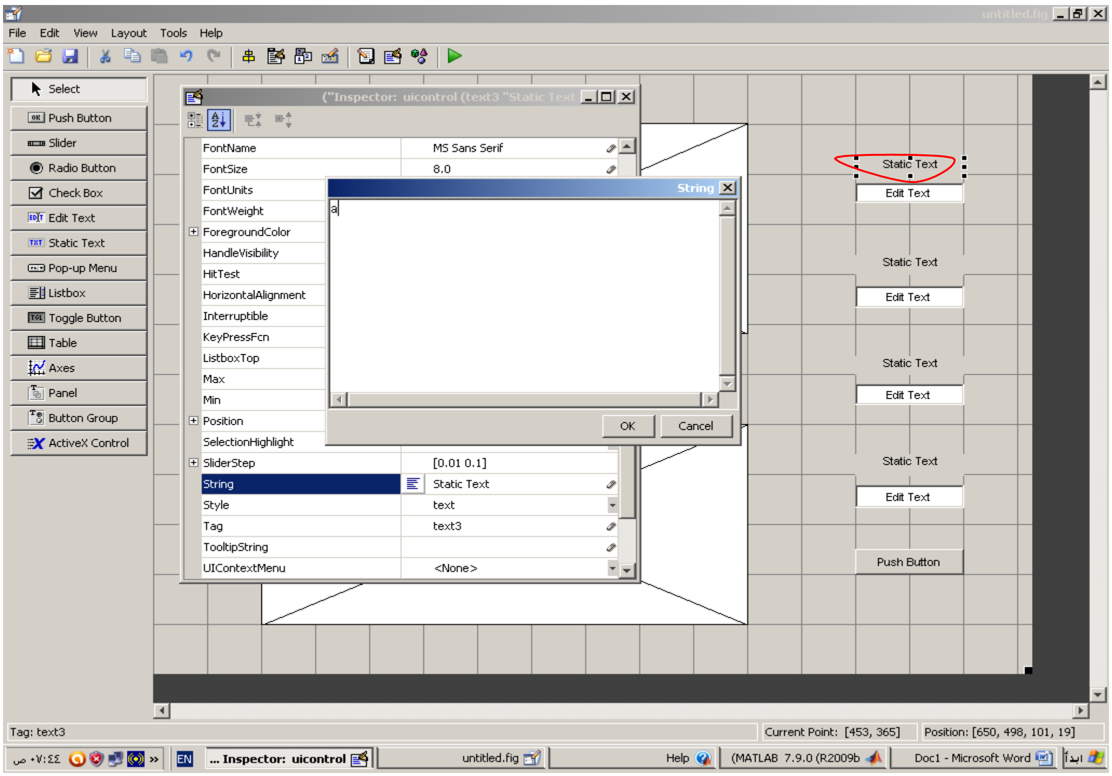
plot a function ***f(t)=t*[u(t+1)-u(t-1)]+u(t-1)-u(t-3)*** and plot a time shifted and time scaled version of f(t) which has the general form ***cf(at+b).*** The user can input variables values of a,b and c. The original function appears on GUI axis1 and the other on GUI axis2.

Designing steps

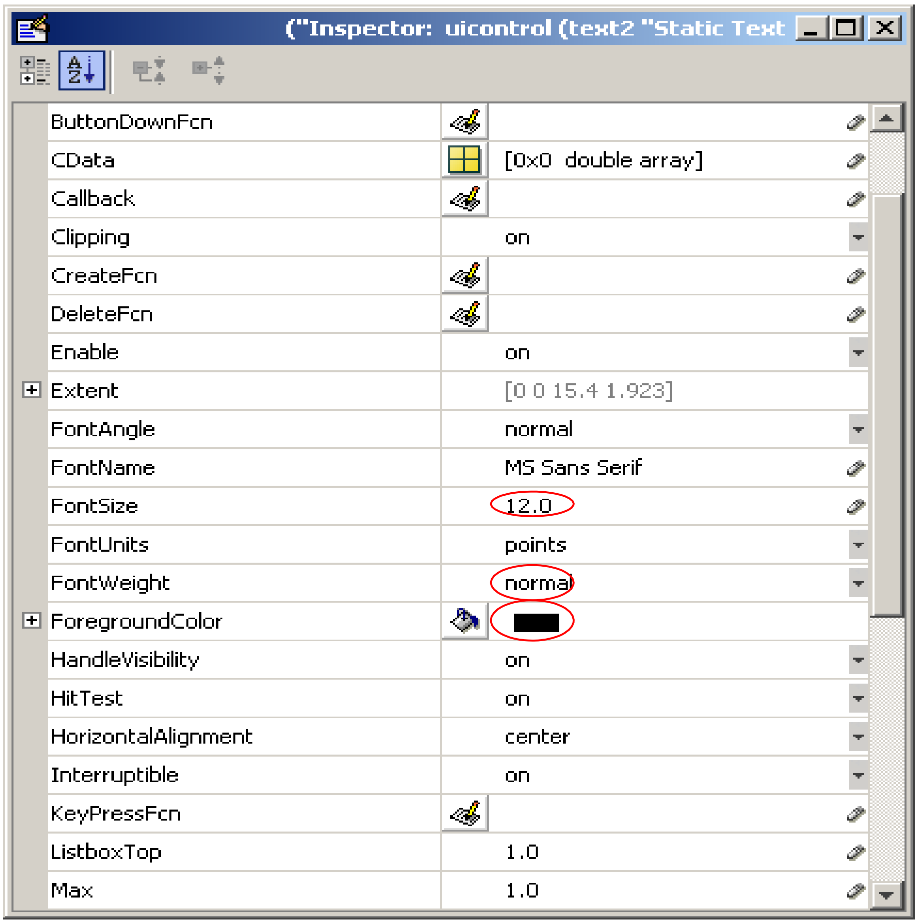
1-Put the following components in the figure.



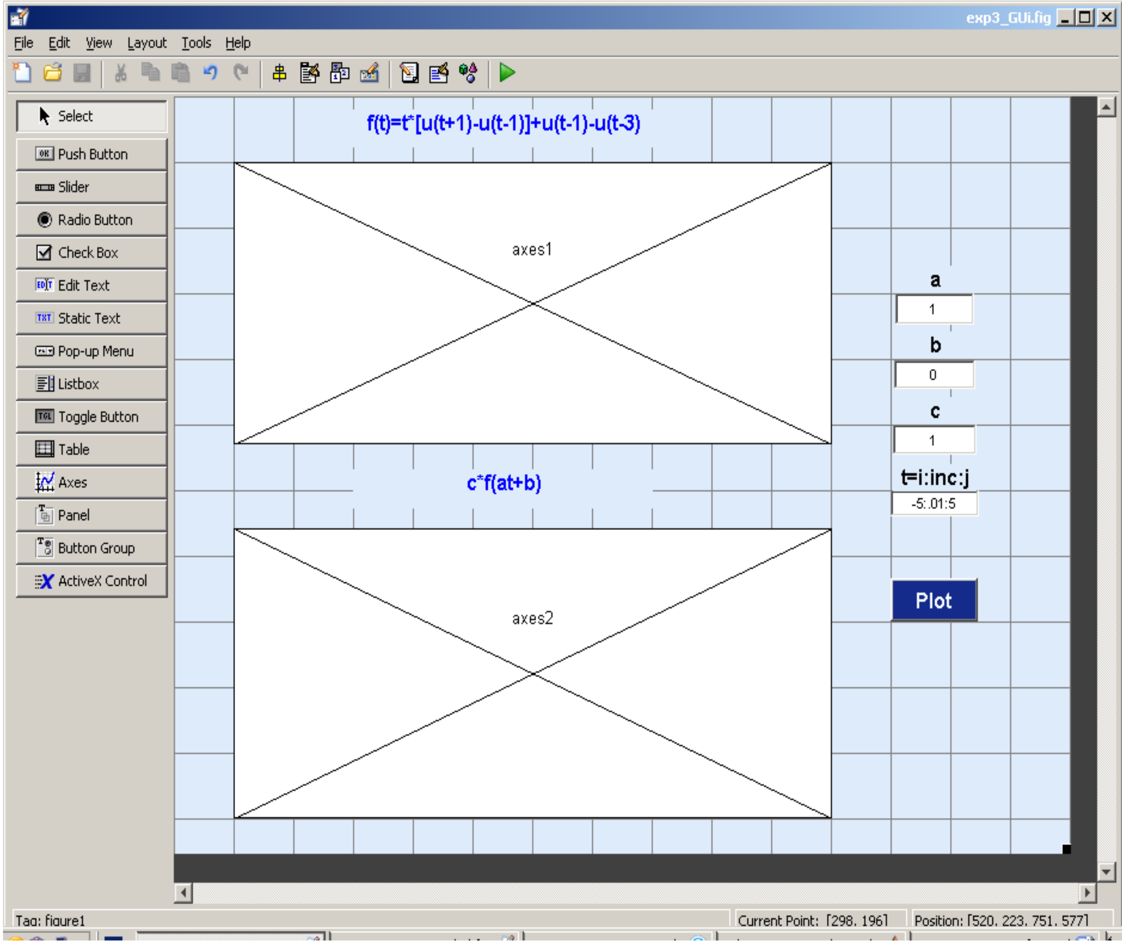
2. Change the name of the (*static text) by* double clicking on each one as follow



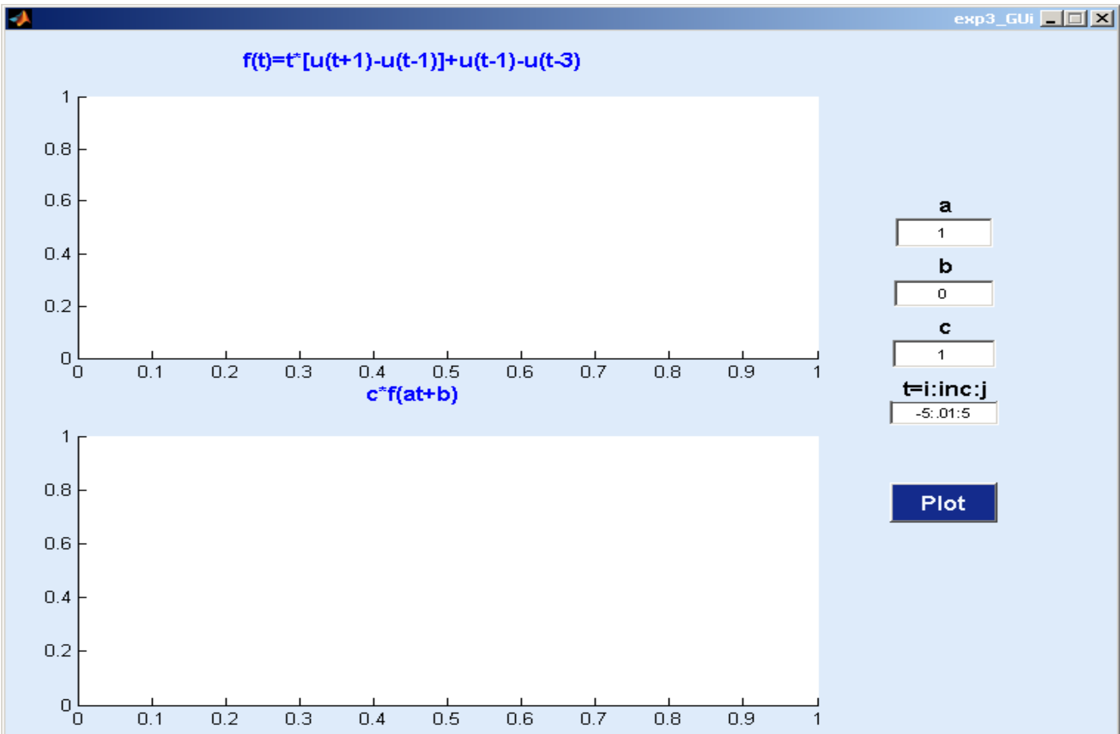
3-change the size ,colour and weight of the text as follow



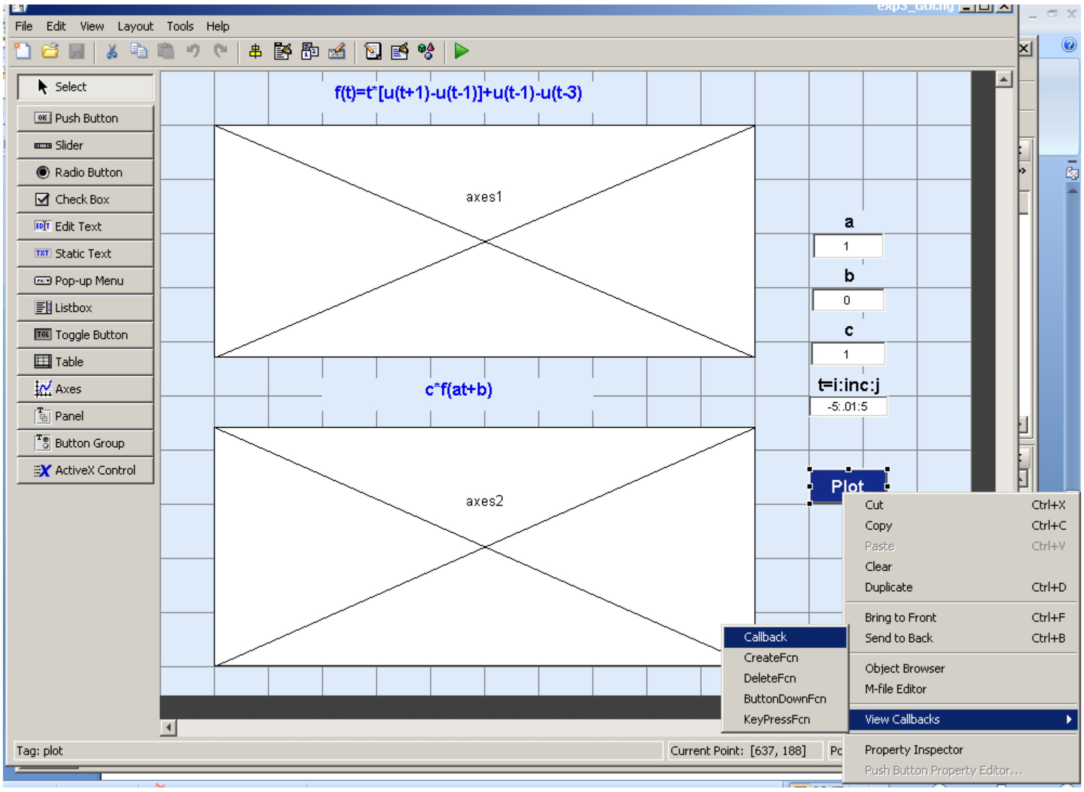
4- The final design will be as follow



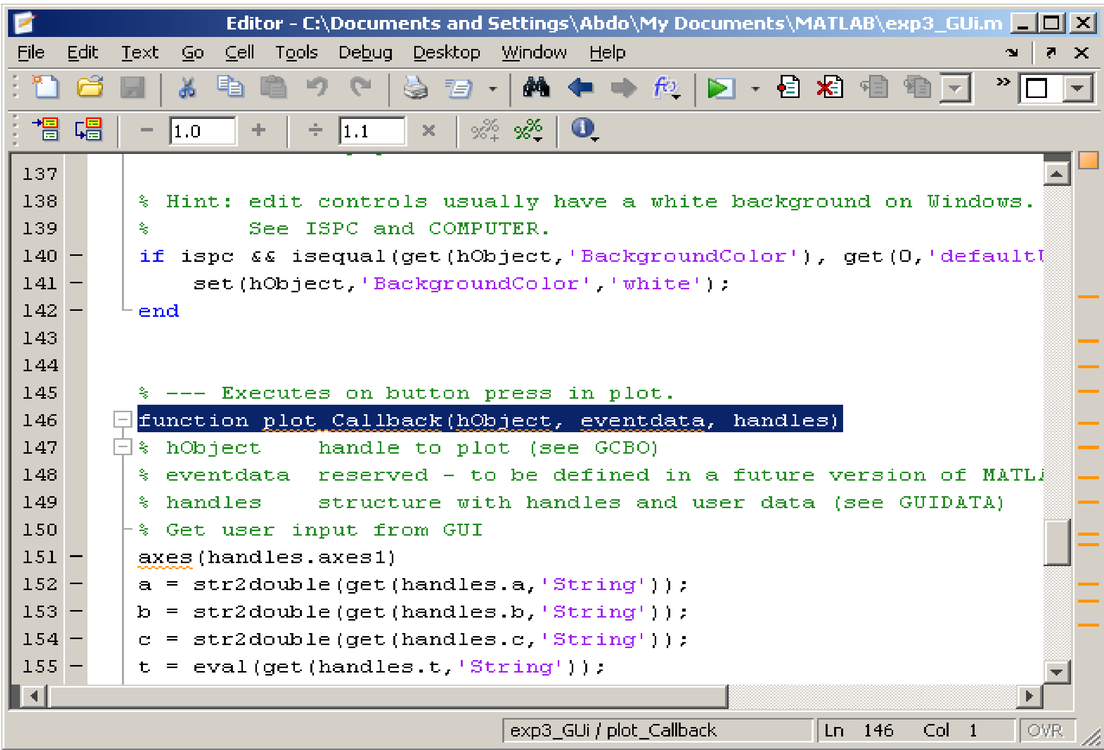
5- push on the green arrow (Run)   and save the design



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



7-The M file will open as follow



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:

Task 2:

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:

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

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 $w_o = 2\,\pi/\mathrm{T}$
- 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*(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 Tp=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 2:

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 x1[n] and x2[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 $w_o = 2\,\pi/\mathrm{T}$
- 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.

$$\underbrace{x[n] * y[n]}_{\text{Periodic convolution}} = \sum_{r=<N>} x[r]y[n-r] \stackrel{FS}{\longleftrightarrow} Na_k b_k$$

2. Frequency Shifting:

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

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

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:

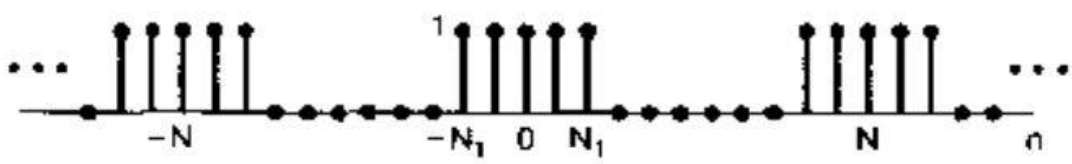
- Periodic Convolution
- Frequency Shifting

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 N1=5 and the number of zeros in each period is also equal to N1 (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:

Task 2:

Task 3:

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:

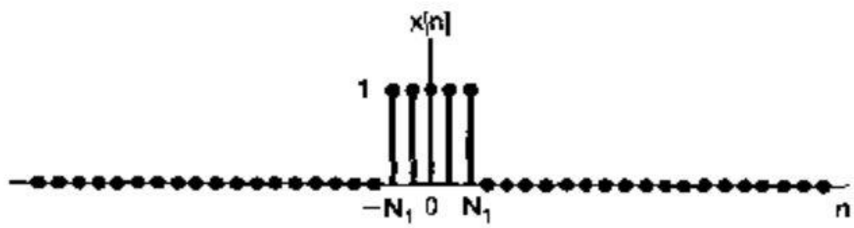
Create separate functions in MATLAB for Discrete Time Fourier Transform (DTFT), i.e. analysis equation, and Inverse Discrete Time Fourier Transform (IDTFT), i.e. synthesis equation.

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:

Task 2:

Task 3:

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

Task 2:

Task 3:

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:

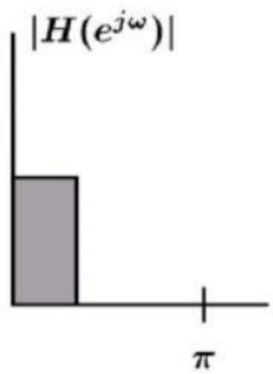
Filters:

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:

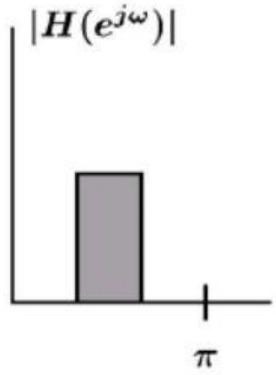
A low pass filter, as its name suggests, is used to pass only the low frequency components of a signal, and block all higher frequency components. The ideal low pass filter is shown below:



Ideal LPF

2. Band Pass Filter:

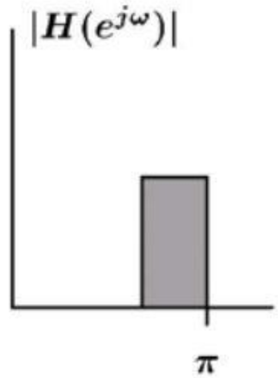
A band pass filter is a filter that passes all frequencies of a signal within a certain range, (which neither includes zero frequency nor pi), and block all other frequencies outside that range. An ideal band pass filter is shown below:



Ideal BPF

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:



Ideal HPF

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) A Low Pass Filter with pass band (f < 1500 Hz)
b) A Band Pass Filter with pass band (1500 < f < 3000 Hz)
c) 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:

Task 2:

Verified: _________________ Date/Time: _______________

Experiment No.14

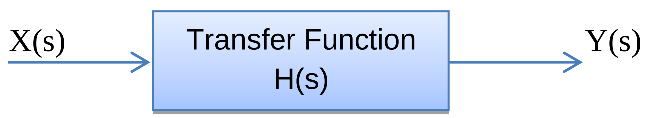
Objective:

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)

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:

Task 2:

Verified: _________________ Date/Time: _______________