Contents

EXPERIMENT # 01 2
EXPERIMENT # 02 10
EXPERIMENT # 03 18
EXPERIMENT # 04 23
EXPERIMENT # 05 35
EXPERIMENT # 06 49
EXPERIMENT # 07 55
EXPERIMENT # 08 69
EXPERIMENT # 09 83
EXPERIMENT # 10 96
EXPERIMENT # 11 126

ISRA University Islamabad campus

Signals & Systems Lab

EXPERIMENT # 01*: Introduction to MATLAB*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……...

Lab Experiment 1: Introduction to MATLAB

Objective: This lab provides an introduction to MATLAB in the first part. The lab also provides tutorial of definitions and Operations on Scalars, Vectors and Matrix in MATLAB and how can we plot Complex Numbers and Vectors using MATLAB.

Part I: Introduction to MATLAB

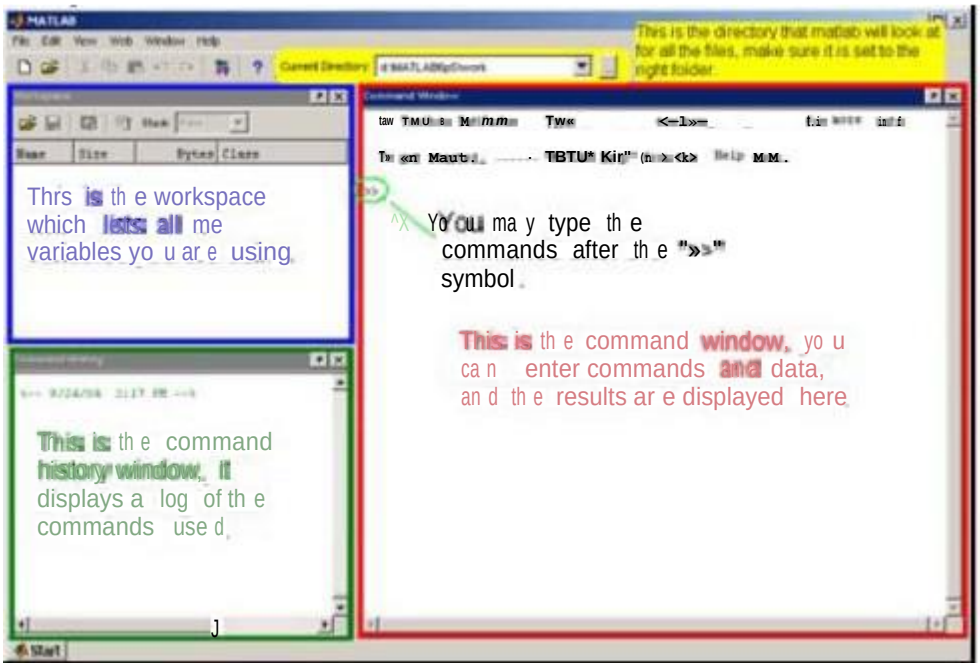
Objective: The objective of this exercise will be to introduce you to the concept of mathematical programming using the software called MATLAB. We shall study how to define variables, matrices etc, see how we can plot results and write simple MATLAB codes.

Introduction:

- What is MATLAB ?
  • MATLAB is a computer program that combines computation and visualization power that makes it particularly useful tool for engineers.
  • MATLAB is an executive program, and a script can be made with a list of MATLAB commands like other programming language.
- MATLAB Stands for MATrix LABoratory.
  • The system was designed to make matrix computation particularly easy.
- The MATLAB environment allows the user to:
  • manage variables
  • import and export data
  • perform calculations
  • generate plots
  • develop and manage files for use with MATLAB.

Display Window:

- Graphic (Figure) Window
  - Displays plots and graphs
  - Created in response to graphics commands.
- M-file editor/debugger window
  - Create and edit scripts of commands called M-files.



Getting Help:

- type one of following commands in the command window:
  - **help** – lists all the help topic
  - **help** *topic* – provides help for the specified topic
  - **help** *command* – provides help for the specified command
  - **help help** – provides information on use of the help command
  - **helpwin** – opens a separate help window for navigation
  - **lookfor** *keyword* – Search all M-files for *keyword*

Variables:

- Variable names:
  - Must start with a letter
  - May contain only letters, digits, and the underscore "_"
  - Matlab is case sensitive, i.e. one & OnE are different variables.
  - Matlab only recognizes the first 31 characters in a variable name.
- Assignment statement:
  - *Variable = number;*
  - *Variable = expression;*
- Example:

```
>> tutorial = 1234;
>> tutorial = 1234
tutorial =
1234
```

***Note*:** When a semi-colon ";" is placed at the end of each command, the result is not displayed.

- Special variables:
  - **ans** : default variable name for the result
  - **pi**: π = 3.1415926……
  - **eps**: ε = 2.2204e-016, smallest amount by which 2 numbers can differ.
  - **Inf** or **inf** : ∞, infinity
  - **NaN** or **nan**: not-a-number
- Commands involving variables:
  - **who**: lists the names of defined variables
  - **whos**: lists the names and sizes of defined variables
  - **clear**: clears all varialbes, reset the default values of special variables.
  - **clear** *name*: clears the variable *name*
  - **clc**: clears the command window
  - **clf**: clears the current figure and the graph window.

Vectors:

- A row vector in MATLAB can be created by an explicit list, starting with a left bracket, entering the values separated by spaces (or commas) and closing the vector with a right bracket.
- A column vector can be created the same way, and the rows are separated by semicolons.
- For example:

```
>> x = [ 0 0.25*pi 0.5*pi 0.75*pi pi ]          //x is row vector
x =
0 0.7854 1.5708 2.3562 3.1416
>> y = [ 0; 0.25*pi; 0.5*pi; 0.75*pi; pi ]      //y is column vector
y =
0
0.7854
1.5708
2.3562
3.1416
```

- Vector Addressing – A vector element is addressed in MATLAB with an integer index enclosed in parentheses.
- Example:

```
>> x(3)        //3rd element of vector x
ans =
1.5708
```

- The colon notation may be used to address a block of elements.
  (start : increment : end)
  start is the starting index, increment is the amount to add to each successive index, and end is the ending index. A shortened format (start : end) may be used if increment is 1.
- Example:

```
>> x(1:3)      //1st to 3rd elements of vector x
ans =
0 0.7854 1.5708
```

Note: MATLAB index starts at 1.

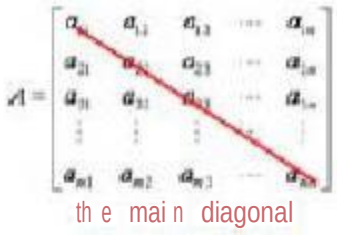
Some useful commands:

x = start:end	create row vector x starting with start, counting by one, ending at end
x = start:increment:end	create row vector x starting with start, counting by increment, ending at or before end
linspace(start,end,number)	create row vector x starting with start, ending at end, having number elements
length(x)	returns the length of vector x
y = x'	transpose of vector x
dot (x, y)	returns the scalar dot product of the vector x and y.

Matrices:

- A Matrix array is two-dimensional, having both multiple rows and multiple columns, similar to vector arrays:
  - it begins with [, and end with]
  - spaces or commas are used to separate elements in a row
  - semicolon or enter is used to separate rows.

Example: Let A is an m x n matrix.



- Example:

```
>> f = [ 1 2 3; 4 5 6]
f =
1       2       3
4       5       6
>> h = [ 2 4 6
1 3 5]
h =
2       4       6
1       3       5
```

- Magic Function

  For example you can generate a matrix by entering

  >> m=magic(4)

  It generates a matrix whose elements are such that the sum of all elements in its rows, columns and diagonal elements are same

- Sum Function

  You can verify the above magic square by entering

  >> sum(m)

  For rows take the transpose and then take the sum

  >> sum(m')

- Diag

  You can get the diagonal elements of a matrix by entering
  >> d=diag(m)
  >> sum(d)

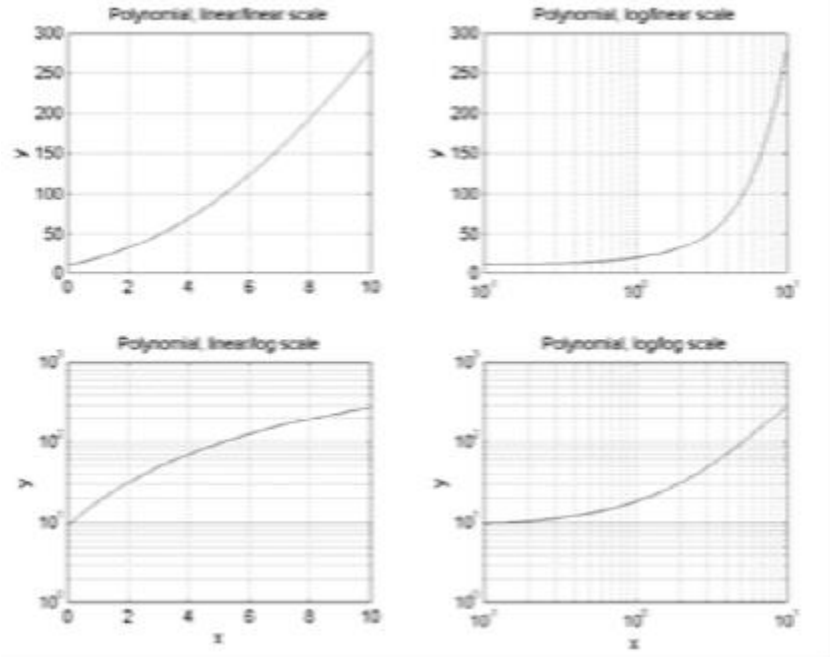
- Matrix Addressing:

  -- *matrixname(row, column)*
  -- **colon** may be used in place of a row or column reference to select the entire row or column.
  Example:
  >> f(2,3)
  ans =
  6
  >> h(:,1)
  ans =
  2
  1

  Where
  f =
  1 2 3
  4 5 6
  h =
  2 4 6
  1 3 5

zeros(n)	returns a n x n matrix of zeros
zeros(m,n)	returns a m x n matrix of zeros
ones(n)	returns a n x n matrix of ones
ones(m,n)	returns a m x n matrix of ones
rand(n)	returns a n x n matrix of random number
rand(m,n)	returns a m x n matrix of random number
size (A)	for a m x n matrix A, returns the row vector [m,n] containing the number of rows and columns in matrix.
length(A)	returns the larger of the number of rows or columns in A.
Transpose	B = A’
Identity Matrix	eye(n) → returns an n x n identity matrix eye(m,n) → returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) → returns the rank of the matrix A.

Matlab Ouput:



Adding new curves to the existing graph, use the **hold** command to add lines/points to an existing plot.

- hold on – retain existing axes, add new curves to current axes. Axes are rescaled when necessary.
- hold off – release the current figure window for new plots

Grids and Labels:

Command	Description
grid on	Adds dashed grids lines at the tick marks
grid off	removes grid lines (default)
grid	toggles grid status (off to on, or on to off)
title (‘text’)	labels top of plot with text in quotes
xlabel (‘text’)	labels horizontal (x) axis with text is quotes
ylabel (‘text’)	labels vertical (y) axis with text is quotes
text (x,y,’text’)	Adds text in quotes to location (x,y) on the current axes, where (x,y) is in units from the current plot.

Additional commands for plotting

Color of the point or curve

Symbol	Color
y	Yellow
m	Magenta
c	Cyan
r	Red
g	Green
b	Blue
w	White
k	Black

Marker of the data

Symbol	Marker
.	●
o	○
x	×
+	+
*	*
s	□
d	◊
v	∇
^	Δ
h	⧆

Plot line styles

Symbol	Line Style
–	Solid line
:	Dotted line
–.	Dash-dot line
- -	Dashed line

Saving Your Work:

All the commands, variables created in the Matlab can be saved using following procedure.

- File -> Save Workspace As -> Choose Directory->Filename.mat -> Click on Save
- In Command Window:

  >> save filename.mat % save all the variables in filename.mat file in Current Directory

  >> save filename.mat s % saves only variable s in filename.mat file.

  >> diary filename.mat %will save all commands and variables in filename file

  >> diary off will close the file and save it as text file containing all the commands
- Loading file

  We can load file by double click on the saved file or in command window type

  >> load filename.mat % all variables will be loaded as saved in previous session.

Exercises:

1.) Set x = 4, y = 12, z = -2 and find the following

(a) y/x*z (b) y/(x*z)

Are these different? Why or why not?

2.) Find the square root of x, y, and z above using the built in matlab function. Print you answer in format long and then again in format short e.

3.) Evaluate the following using matlab:

$\sin(\pi / 3), \cos(\pi / 4), \tan(\pi / 2), \arcsin(0), |z|, \ln(x), e^{y}$

Anything interesting happen?

4.) Create the following row vector a = [1,2,3,4,5,6] two different ways.

5.) Make a column vector *b* that is the transpose of *a* two different ways.

6.) Create a vector *c* that is the same as *a* but whose second component is the last component of *b*.

7.) Create a 4x4 array $A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ -1 & 1 & 1 & 0 \end{bmatrix}$.

8.) What is A'?

9.) Set a column vector *d* equal to the first row of A using ':' notation and set a vector e equal to the first row of A using the column notation.

10.) Create a vector, u, of 15 equally spaced points between $\frac{-}{2}$ and $\frac{}{2}$

ISRA University Islamabad Campus

Signals & Systems Lab

EXPERIMENT # 02*: Scripts, Functions, Symbolic Math's Toolbox and Flow Control using MATLAB*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……...

Lab Experiment 2: Scripts, Functions, Symbolic Math's Toolbox and Flow Control using MATLAB

Objective: The objective of this lab is to introduce you to writing M-file scripts, creating MATLAB Functions and reviewing MATLAB flow control like 'if-elseif-end', 'for loops' and 'while loops'.

Overview:

MATLAB is a powerful programming language as well as an interactive computational environment. Files that contain code in the MATLAB language are called M-files. You create M-files using a text editor, then use them as you would any other MATLAB function or command. There are two kinds of M-files:

- ***Scripts***, which do not accept input arguments or return output arguments. They operate on data in the workspace. MATLAB provides a full programming language that enables you to write a series of MATLAB statements into a file and then execute them with a single command. You write your program in an ordinary text file, giving the file a name of 'filename.m'. The term you use for 'filename' becomes the new command that MATLAB associates with the program. The file extension of .m makes this a MATLAB M-file.
- ***Functions***, which can accept input arguments and return output arguments. Internal variables are local to the function.

If you're a new MATLAB programmer, just create the M-files that you want to try out in the current directory. As you develop more of your own M-files, you will want to organize them into other directories and personal toolboxes that you can add to your MATLAB search path. If you duplicate function names, MATLAB executes the one that occurs first in the search path.

Scripts:

When you invoke a script, MATLAB simply executes the commands found in the file. Scripts can operate on existing data in the workspace, or they can create new data on which to operate. Although scripts do not return output arguments, any variables that they create remain in the workspace, to be used in subsequent computations. In addition, scripts can produce graphical output using functions like plot. For example, create a file called 'myprogram.m' that contains these MATLAB commands:

```
% Create random numbers and plot these numbers
clc

r = rand(1,50)
plot(r)
```

Typing the statement 'myprogram' at command prompt causes MATLAB to execute the commands, creating fifty random numbers and plots the result in a new window. After execution of the file is complete, the variable 'r' remains in the workspace.

Functions:

Functions are M-files that can accept input arguments and return output arguments. The names of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt. An example is provided below:

```
function f = fact(n)                    Function definition line
% Compute a factorial value.            H1 line
% FACT(N) returns the factorial of N,   Help Text
% usually denoted by N!

% Put simply, FACT(N) is PROD(1:N).     Comment
f = prod(1:n);                          Function body
```

M-File Element	Description
Function definition line (functions only)	Defines the function name, and the number and order of input and output arguments.
H1 line	A one line summary description of the program, displayed when you request help on an entire directory, or when you use 'lookfor'.
Help text	A more detailed description of the program, displayed together with the H1 line when you request help on a specific function
Function or script body	Program code that performs the actual computations and assigns values to any output arguments.
Comments	Text in the body of the program that explains the internal workings of the program.

The first line of a function M-file starts with the keyword 'function'. It gives the function name and order of arguments. In this case, there is one input arguments and one output argument. The next several lines, up to the first blank or executable line, are comment lines that provide the help text. These lines are printed when you type 'help fact'. The first line of the help text is the H1 line, which MATLAB displays when you use the 'lookfor' command or request help on a directory. The rest of the file is the executable MATLAB code defining the function.

The variable n & f introduced in the body of the function as well as the variables on the first line are all local to the function; they are separate from any variables in the MATLAB workspace. This example illustrates one aspect of MATLAB functions that is not ordinarily found in other programming languages—a variable number of arguments. Many M-files work this way. If no output argument is supplied, the result is stored in ans. If the second input argument is not supplied, the function computes a default value.

Flow Control:

Conditional Control – if, else, switch

This section covers those MATLAB functions that provide conditional program control. if, else, and elseif. The if statement evaluates a logical expression and executes a group of statements when the expression is true. The optional elseif and else keywords provide for the execution of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements.

The groups of statements are delineated by the four keywords—no braces or brackets are involved as given below

if <condition>
<statements>;
elseif <condition>
<statements>;
else
<statements>;
end

It is important to understand how relational operators and if statements work with matrices. When you want to check for equality between two variables, you might use

if A == B, ...

This is valid MATLAB code, and does what you expect when A and B are scalars. But when A and B are matrices, A == B does not test if they are equal, it tests where they are equal; the result is another matrix of 0's and 1's showing element-by-element equality. (In fact, if A and B are not the same size, then A == B is an error.)

```
>>A = magic(4);
>>B = A;
>>B(1,1) = 0;
>>A == B
        ans =
                0     1     1     1
                1     1     1     1
                1     1     1     1
                1     1     1     1
```

The proper way to check for equality between two variables is to use the isequal function:

if isequal(A, B), ...

isequal returns a scalar logical value of 1 (representing true) or 0 (false), instead of a matrix, as the expression to be evaluated by the if function.

Using the A and B matrices from above, you get

```
>>isequal(A, B)
        ans =
        0
```

Here is another example to emphasize this point. If A and B are scalars, the following program will never reach the "unexpected situation". But for most pairs of matrices, including our magic

```
if A > B

elseif
    'less'
elseif
    'equal'
    error('Unexpected situation')
```

squares with interchanged columns, none of the matrix conditions A > B, A < B, or A == B is true for all elements and so the else clause is executed:
Several functions are helpful for reducing the results of matrix comparisons to scalar conditions for use with if, including 'isequal', 'isempty', 'all', 'any'.

Switch and Case:
The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups. Only the first matching case is executed. The syntax is as follows
switch <condition or expression>
case <condition>
    <statements>;
…
case <condition>
…
otherwise
    <statements>;
end
There must always be an end to match the switch. An example is shown below.

```
switch rem(n,2) % to find remainder of any number 'n'
case 0
    disp('Even Number') % if remainder is zero
case 1
    disp('Odd Number') % if remainder is one
end
```

Unlike the C language switch statement, MATLAB switch does not fall through. If the first case statement is true, the other case statements do not execute. So, break statements are not required.
For, while, break and continue:
This section covers those MATLAB functions that provide control over program loops.
for:
The 'for' loop, is used to repeat a group of statements for a fixed, predetermined number of times. A matching 'end' delineates the statements. The syntax is as follows:
for <index> = <starting number>:<step or increment>:<ending number>
    <statements>;
end

```
    r(n) = n*n; % square of a
end
r
```

The semicolon terminating the inner statement suppresses repeated printing, and the r after the loop displays the final result.
It is a good idea to indent the loops for readability, especially when they are nested:

```
for i
    for j
        H(i,j) = 1/(i+j);
    
```

while:
The 'while' loop, repeats a group of statements indefinite number of times under control of a logical condition. So a while loop executes atleast once before it checks the condition to stop the execution of statements. A matching 'end' delineates the statements. The syntax of the 'while' loop is as follows:
while <condition>
<statements>;
end
Here is a complete program, illustrating while, if, else, and end, that uses interval bisection to find a zero of a polynomial:

```
while (j<=4 )
    a(j)=10*j;
    j=j+1;

x
```

The result is a root of the polynomial $x^3 - 2x - 5$, namely x = 2.0945. The cautions involving matrix comparisons that are discussed in the section on the 'if' statement also apply to the while statement.
break:
The break statement lets you exit early from a 'for' loop or 'while' loop. In nested loops, break exits from the innermost loop only.

```
a=0; b=0;
for i=1:10,
    if(i==6)
        break;
    end
    a(i)=i;
end
a
```

continue:
The continue statement passes control to the next iteration of the for loop or while loop in which it appears, skipping any remaining statements in the body of the loop. The same holds true for continue statements in nested loops. That is, execution continues at the beginning of the loop in which, the continue statement was encountered.

```
for j=1:10,
    if(j==6)
        continue;
    end
    b(j)=j;
end
b
```

Symbolic Math's Toolbox:

Overview:

Symbolic Math Toolbox™ and Extended Symbolic Math Toolbox™ software incorporates symbolic computation into the numeric environment of MATLAB® software. These toolboxes supplement MATLAB numeric and graphical capabilities with several other types of mathematical computation.

Symbolic Object:

Symbolic Math Toolbox™ software defines a new MATLAB® data type called a symbolic object. Internally, a symbolic object is a data structure that stores a string representation of the symbol. Symbolic Math Toolbox software uses symbolic objects to represent symbolic variables, expressions, and matrices.

The following example illustrates the difference between a standard MATLAB data type, such as double, and the corresponding symbolic object. The MATLAB command

sqrt(2)

returns a floating-point decimal number:

ans =

1.4142

On the other hand, if you convert 2 to a symbolic object using the sym command, and then take its square root by entering

a = sqrt(sym(2))

the result is

a =

2^(1/2)

MATLAB gives the result 2^(1/2), which means 21/2, using symbolic notation for the square root operation, without actually calculating a numerical value. MATLAB records this symbolic expression in the string that represents 2^(1/2). You can always obtain the numerical value of a symbolic object with the double command:

double(a)

ans =

1.4142

Notice that the result is indented, which tells you it has data type double. Symbolic results are not indented.

If you set a variable equal to a symbolic expression, and then apply the syms command to the variable, MATLAB® software removes the previously defined expression from the variable. For example,

```
>>syms a
>>f = a + b
```

returns

```
f =
a+b
```

Substitution:

You can substitute a numerical value for a symbolic variable using the subs command. For example, to substitute the value x = 2 in the symbolic expression,

f = 2*x^2 - 3*x + 1

enter the command

```
>>subs(f,2)
```

This returns f (2):

```
ans =
     3
```

When your expression contains more than one variable, you can specify the variable for which you want to make the substitution. For example, to substitute the value x = 3 in the symbolic expression,

```
>>syms
>>f = x^2*y + 5*x*sqrt(y)
```

enter the command

subs(f, x, 3)

This returns

```
ans =
9*y+15*y^(1/2)
```

On the other hand, to substitute y = 3, enter

```
>>subs(f, y, 3)
ans =
3*x^2+5*x*3^(1/2)
```

If you want to substitute x=2, and y=3, enter

```
>>subs(f,{x,y},{2,3})
ans =
   29.3205
```

Differentiation:

To illustrate how to take derivatives using Symbolic Math Toolbox™ software, first create a symbolic expression:

```
syms x
f = sin(5*x)
```

The command

```
diff(f)
```

differentiates f with respect to x:

```
ans =
5*cos(5*x)
```

ISRA University, Islamabad campus

Signals & Systems Lab

EXPERIMENT # 03*: Signals and their Classifications*

Name of Student: ……..

Roll No.: ……

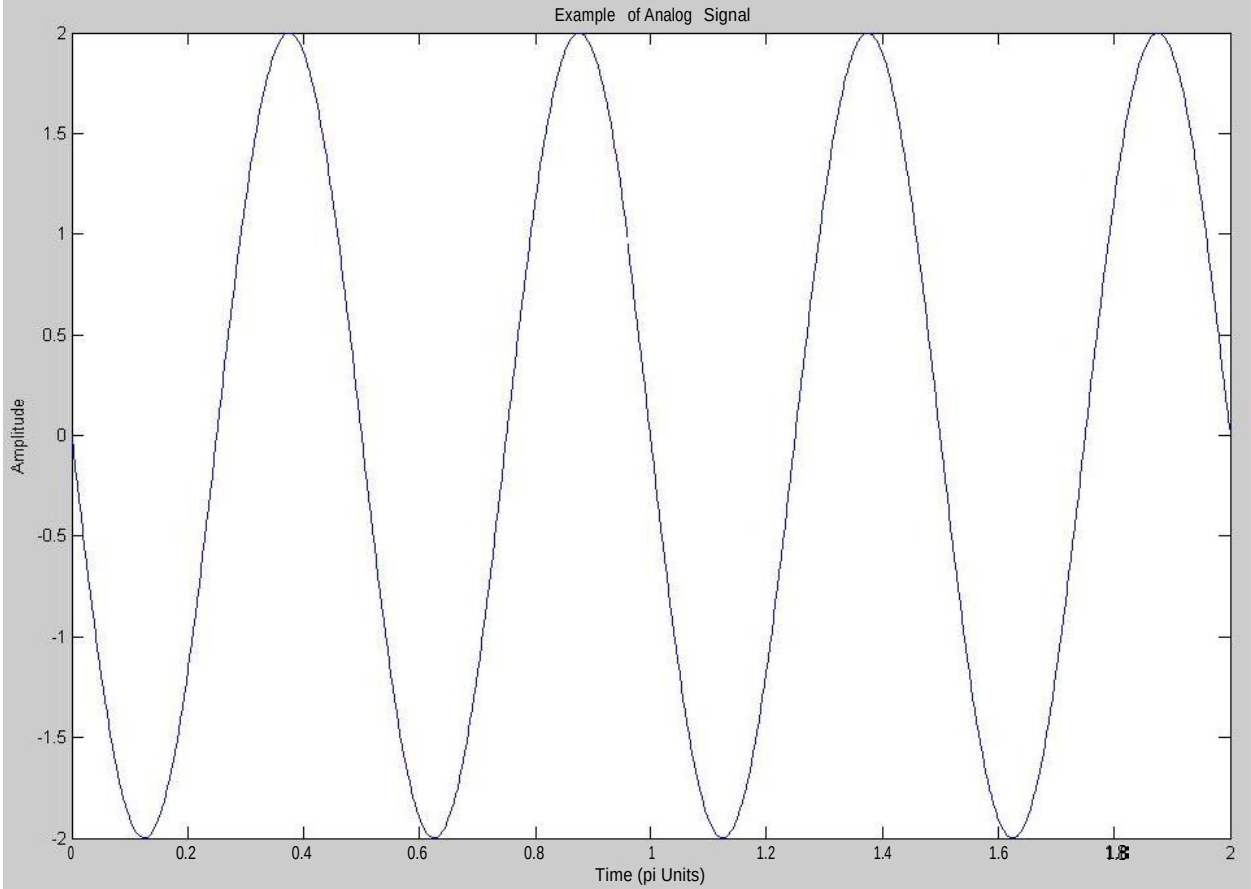
Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

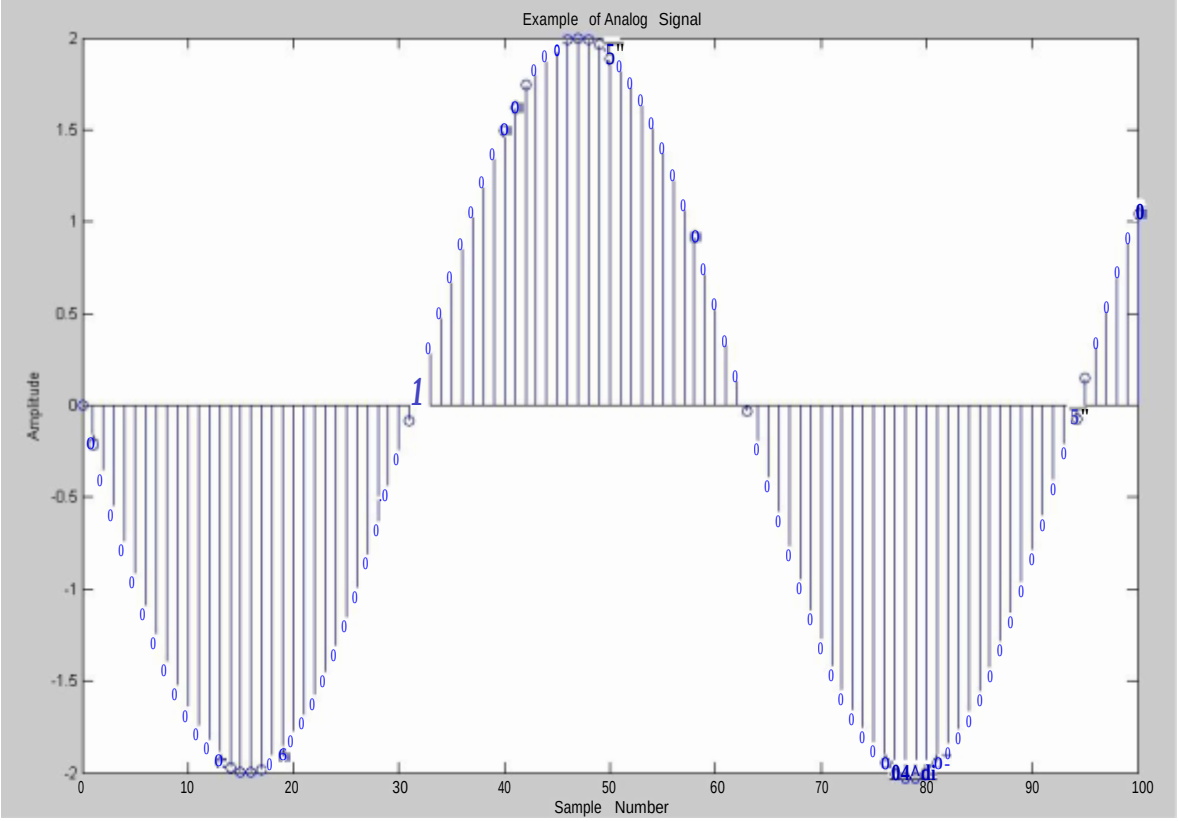
Remarks: ……

Instructor's Signature: ……...



We can also plot analog signal using stem command.

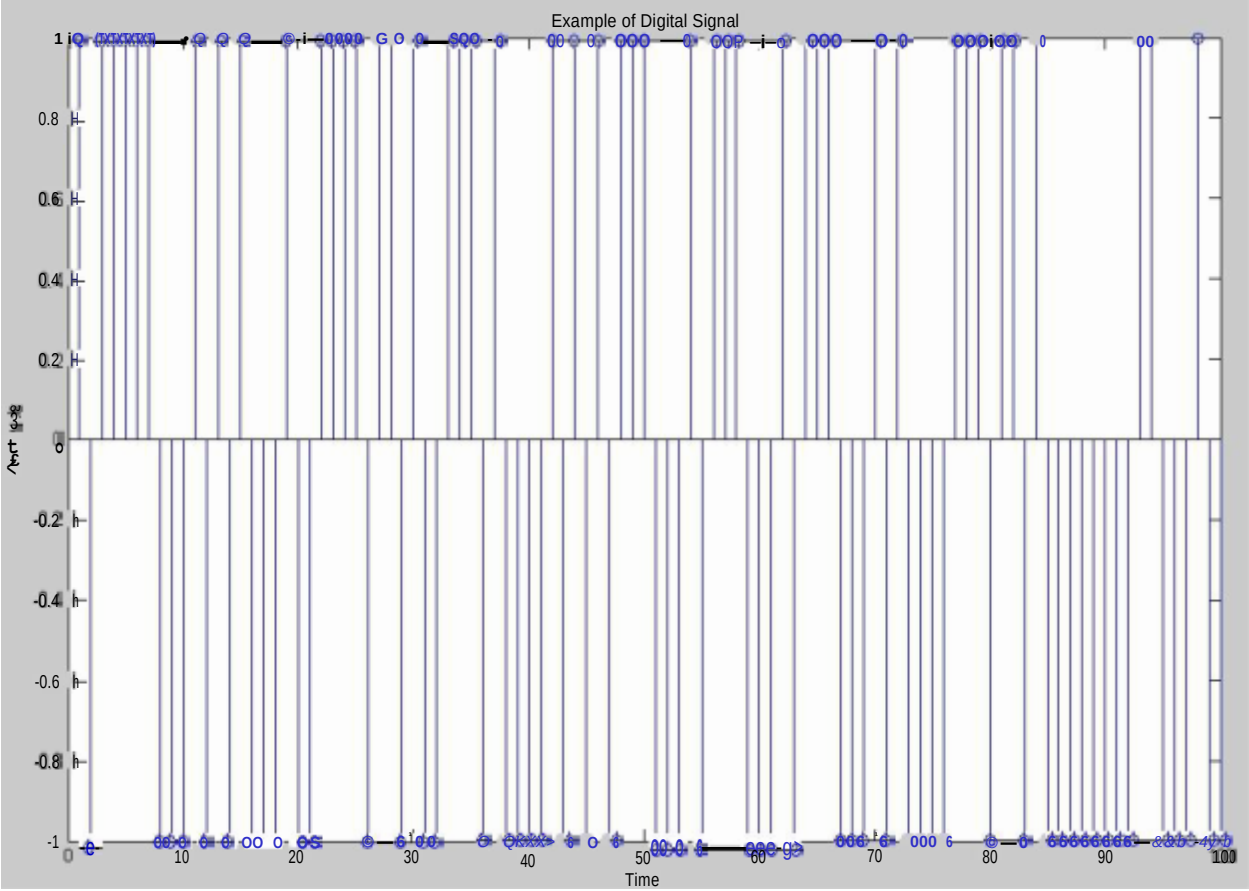
```
%Example of Discrete Time Analog Signal
A=2; %Amplitude
w=0.1; %Frequence
phase=pi; %Phase
n=(0:100); %Time axis
stem(n,A*sin(w*n+phase));
title('Example of Analog Signal');
xlabel ('Sample Number');
ylabel ('Amplitude');
```



Digital Signal:

Digital signal is the one which can take countable finite values in range.
Using MATLAB we can plot digital signal using stem of plot commands for example a random bipolar data.

```
%Digital Signal Example
d=round(rand(1,100)); %Random Values
bd=2*d-1; %binary data
phase=pi; %Phase
n=1:100; %Time axis
stem(n,bd);
title('Example of Digital Signal');
xlabel ('Time');
ylabel ('Amplitude');
```



Square wave is an another example of Digital signal.

```
%Digital Signal Example

A=1; %Amplitude
f=2; %Frequence
phase=pi; %Phase
t=(0:0.01:2*pi)/pi; %Horizontal time axis

plot(t,A*square(2*pi*f*t+phase));
axis([-0.1,2.1,-1.1,1.1]);
title('Example of Digital Signal');
xlabel ('Time (pi Units)');
ylabel ('Amplitude');
```

4) Even vs. Odd Signals

If x(n)=x(-n) then signal is called even signal,
Any signal can be decomposed into its even and odd part using following MATLAB function

```
function [xe,xo,m]=evenodd(x,n)
if any (imag(x)~=0)
    error('x is not real sequence');
end
m=-fliplr(n);
m1=min(m,n); m2=max(m,n); m=min(m1):max(m2);
nm=n(1)-m(1);
n1=1:length(n);
x1=zeros(1,length(m));
x1(n1+nm)=x;
x=x1;
xe=0.5*(x+fliplr(x));
xo=0.5*(x-fliplr(x));
```

5) Deterministic vs. Random Signals

A phenomena is called random, if the outcome of the experiment is uncertain. However, random phenomenon often follow certain recognizable patterns. This long-run regularity of random process can be described mathematically.

Example: Rolling a dice 1000 times.

```
one=0;two=0;three=0;four=0;five=0;six=0;%Possible Outcomes
exp=ceil(6*rand(1,1000)); %1000 rolls of dice
for i=1:1000 %Counting each outcome
    if(a(i)==1)
        one=one+1;
    end
    if(a(i)==2)
        two=two+1;
    end
    if(a(i)==3)
        three=three+1;
    end
    if(a(i)==4)
        four=four+1;
    end
    if(a(i)==5)
        five=five+1;
    end
    if(a(i)==6)
        six=six+1;
    end
end
%Probability of each outcome
probDice(1)=one/1000;
probDice(2)=two/1000;
probDice(3)=three/1000;
probDice(4)=four/1000;
probDice(5)=five/1000;
probDice(6)=six/1000;
stem(probDice);%Plot each outcome
axis([0 7 0 0.25]);
```

ISRA University Islamabad,Campus

Signals & Systems Lab

EXPERIMENT # 04*: Operations on the signals*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……..

Experiment 4: Operations on Signals

A single is classified with respect to its domain and range. Similarly operations on signals are classified into two categories.

1. Operations on domain
2. Operations on Range

Both categories will be dealt in the current laboratory.

1. Operations on Domain

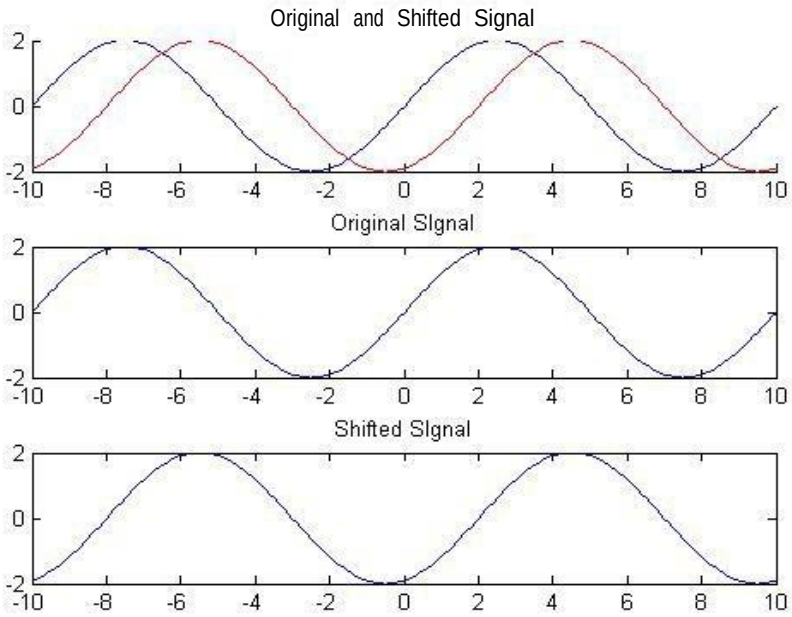
Domain operations are those which involve time axis or integer axis as main focus of operations. Domain operations are classified into following four categories.

1. Time Shifting
2. Time Scaling
3. Time Reversal
4. Sampling

1. Time Shifting

Time Shift or delay operation shifts the signals to the desired delay. Given a signal $x()$, a shifted signal will be of the form $y(t) = x(t - to)$ where to is the delay or shift in time domain. Let **Asin(2 ft)** is the signal which is desired to be shifted by an amount to . Here is the Matlab code for it.

```
% Time Delay/ Time Shifting
    inc=0.1;
    t=-10:inc:10;
    f=0.1;
    t0=2;    % Shift Units
    a=2;     % Amplitude
    x=a*sin(2*pi*f*t);
    y=a*sin(2*pi*f*(t-t0));
    subplot(311);
    hold on;
    plot(t,x);
    plot(t,y,'r');
    title('Original and Shifted Signal');
    subplot(312);
    plot(t,x);
    title('Original SIgnal');
    subplot(313);
    plot(t,y);
    title('Shifted SIgnal');
```



For discrete signal each sample of $x(n)$ is shifted by an amount k to obtain the shifted sequence $y(n)$.

$$y(n) = \{x(n - k)\}$$

If we let $m = n - k$ then $n = m + k$ and we get

$$y(m + k) \quad \{x(n)\}$$

Matlab code for the function of time shift operation is as follows:

```
function [y,n]=sigshift(x,m,n0)
      n=m+n0;
      y=x;
```

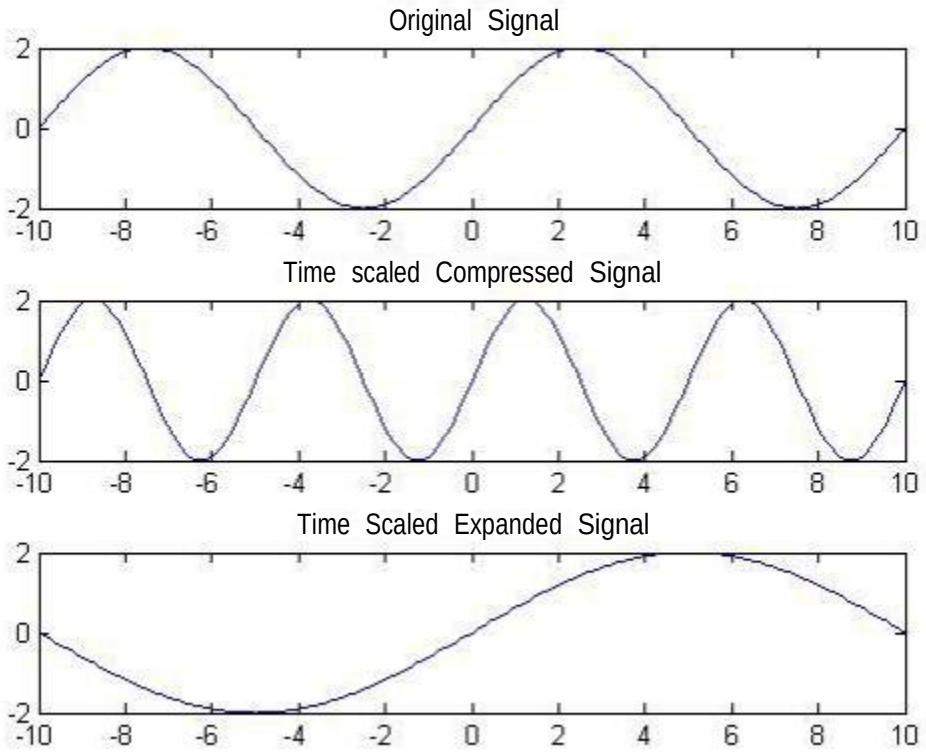
2. Time Scaling

Operation of time scaling scales the time axis to a certain scale resulting in increasing or decreasing the frequency of the signal which compresses or expands the signal on time domain.General expression for the time scaled output is given below.

$$y(t) = x(t)$$

Let **Asin(2 ft)** is the original signal on time scale. We scale the *t* domain by amount alpha and beta. Alpha scaled signal will be compresses due to increment in frequency while beta scaled signal will be expanded. Exampled is coded below.

```
%time Scaling
inc=0.1;
t=-10:inc:10;
f=0.1;

alpha=2;    % Compression Units
a=2;        % Amplitude
beta=0.5;   % Expension units
x=a*sin(2*pi*f*t);
y=a*sin(2*pi*f*(alpha*t));
z=a*sin(2*pi*f*(beta*t));

subplot(311);
plot(t,x);
title('Original Signal');
subplot(312);

plot(t,y);
title('Time scaled Compressed SIgnal');
subplot(313);
plot(t,z);
title('Time Scaled Expanded SIgnal');
```



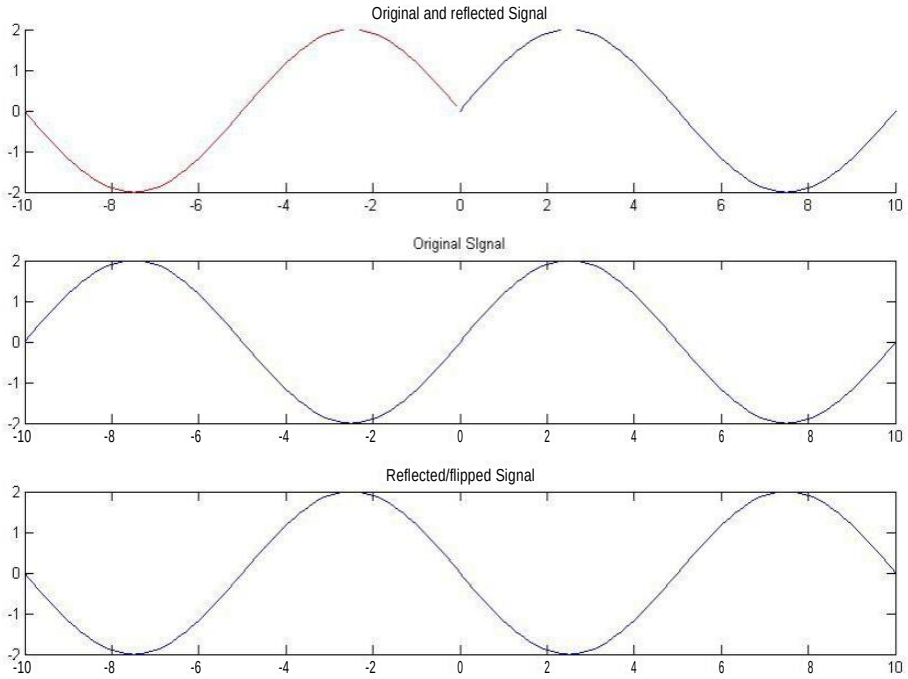
3. Time Reversal

Time Reversal operation flips each sample of the signal about t=0 or n=0 to obtain a folded sequence.

$$y(n) = \{x(-n)\}$$

In Matlab fliplr(x) function is used to flip the sample values and –fliplr(x) is used to flip the indices.

```
inc=0.1;
t=-10:inc:10;

f=0.1;
a=2;        % Amplitude
x=a*sin(2*pi*f*t);

rx=fliplr(x);
rt=-1*fliplr(t);
l=length(x);

subplot(311);
hold on;
plot(t(l/2:l),x(l/2:l));
plot(rt(1:l/2),rx(1:l/2),'r');
title('Original and reflected Signal');

subplot(312);
plot(t,x);
title('Original SIgnal');
subplot(313);
plot(rt,rx);
title('Reflected/flipped SIgnal');
```



A Matlab function to implement the signal flipping is given below.

```
function [y,n]=sigfold(x,m)
      y=fliplr(x)  %flips the amplitude levels
      n=-1*fliplr(n) %flips the indices on negative sides
```

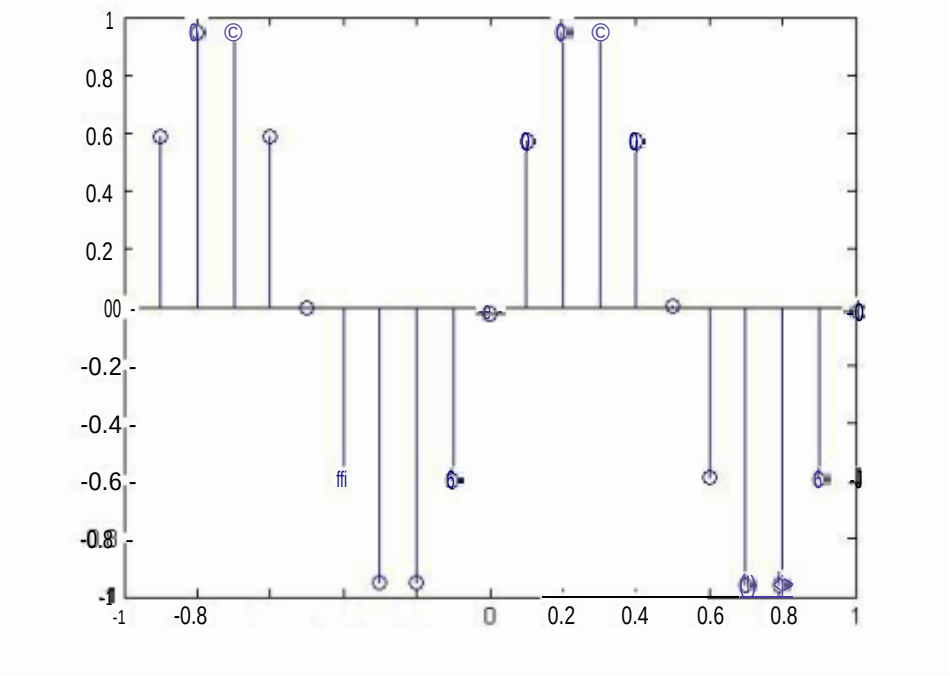
4. Sampling

Sampling is the reduction of a continuous signal to a discrete signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal).

A sample refers to a value or set of values at a point in time and/or space.

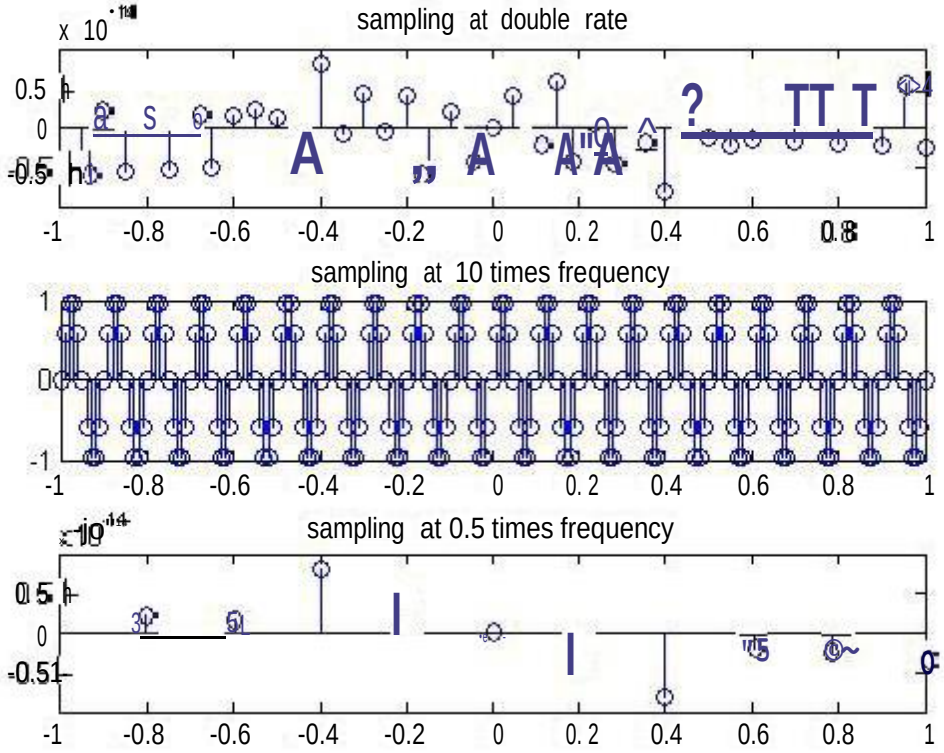
Let $x = \sin(2 \quad ft)$ be the signal with highest frequency component f. We sample the signal at different rates as follows.

```
% Sampling of single frequency component
f=1;
fs=10*f;     %Sampling frequency
ts=1/fs;     % Sampling Interval
t=-1:ts:1;
x=sin(2*pi*f*t);
stem(t,x);
```



Following example shows the sampling of the given signal at different rates. The output graph shows how sampling rate predicts the signal behavior and loss of information.

```
% Sampling of single frequency component at different rates
f=10;
fs1=2*f;     %Sampling frequency
fs2=10*f;

fs3=0.5*f;
ts1=1/fs1;     % Sampling Interval
ts2=1/fs2;
ts3=1/fs3;

t1=-1:ts1:1;
t2=-1:ts2:1;
t3=-1:ts3:1;
x1=sin(2*pi*f*t1);
x2=sin(2*pi*f*t2);
x3=sin(2*pi*f*t3);
subplot(311); stem(t1,x1);
title('sampling at double rate');
subplot(312);stem(t2,x2);
title('sampling at 10 times frequency');
subplot(313);stem(t3,x3);
title('sampling at 0.5 times frequency');
```



Operations on Range

Range operations involve amplitudes of the signal as major focus of operations. They are classified as

1. Amplitude Scaling
2. Addition of Signals
3. Subtraction of signals
4. Multiplication of Signals
5. Derivative of signals

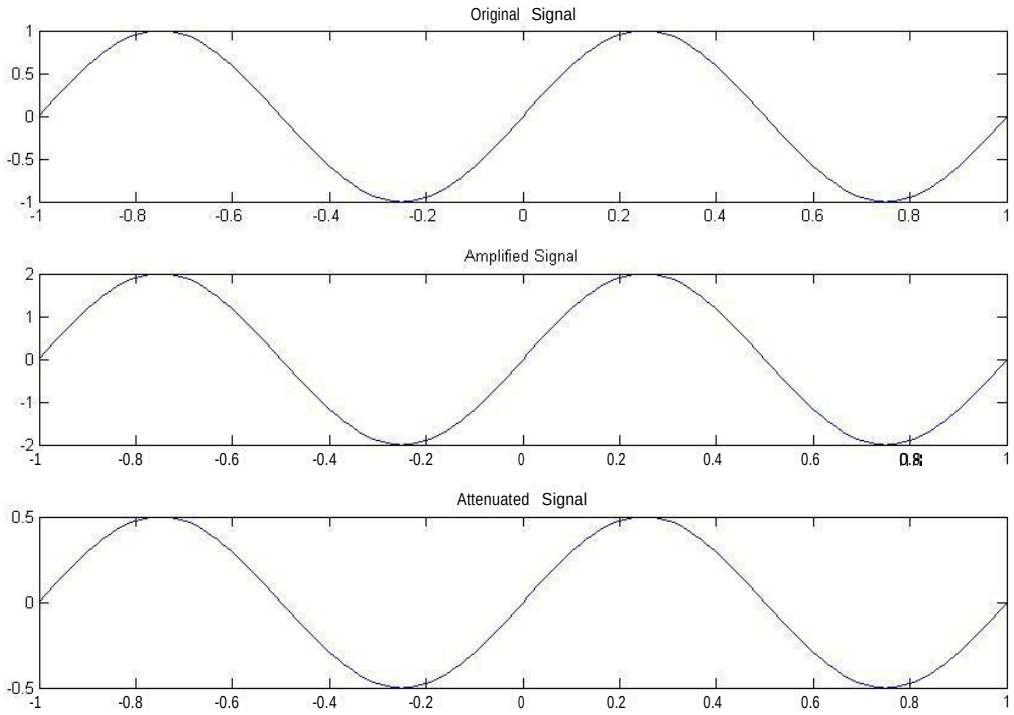
1. Amplitude Scaling

Amplitude scaling rescales the amplitude of the signal. As a result signal may be amplified or attenuated. Given a signal $x = A\sin(2 ft)$ where A is the amplitude of the signal. We can rescale the amplitude by some constant multiplier alpha or beta. Amplitude scaled output will be

$$y = \{A\sin(2 ft)\}$$

Code given below scales the amplitude by constant alpha and beta which amplifies and attenuate the signal respectively.

```
% Amplitude Scaling
t=-1:0.01:1;
f=1;
a=1;  % Amplitude
alpha=2; % Amplitude scale
beta=0.5; % Attenuated scale
x=a*sin(2*pi*f*t);
y=alpha*sin(2*pi*f*t);
z=beta*sin(2*pi*f*t);
subplot(311);
plot(t,x);
title('Original Signal');
subplot(312);
plot(t,y);
title('Amplified Signal');
subplot(313);
plot(t,z);
title('Attenuated SIgnal');
```



A Matlab function to implement the Amplitude scaling is given below.

```
function y=sigscale(x,alpha)
y=alpha*x;
```

2. Signal Addition

This is a sample by sample addition given by

$$\{x1(n)\} + \{x2(n)\} \quad \{x1(n) + x2(n)\}$$

It is implemented in Matlab using + operator however this requires the lengths of the vectors to be same. But if the signals are of different lengths or if the sample positions are different for same length sequences, then we cannot directly use the '+ 'operator. We first have to augment the x1(n) and x2(n) so that we have the same position vector and hence the same length. This involves MATLAB indexing operations. Logical operators of '&','<', :>' and *find* functions are used to make x1(n) and x2(n) of same lengtha. The following function called *sigadd* performs the said operation.

```
% Addition of sgnals
function [y,n]=sigadd(x1,n1,x2,n2)
n=min(min(n1),min(n2)):max(max(n1),max(n2));
```

```
y1=zeros(1,length(n));   %initialization
y2=y1;
y1(find((n>=min(n1)) & (n<=max(n1))==1))=x1;
y2(find((n>=min(n2)) & (n<=max(n2))==1))=x2;
y=y1+y2;
```

```
Example:
```

Let $x1 = [\ 1 \quad 2 \quad 3]$ having indices as $n1 = [\ \text{-}1 : 1]$

And $x2 = [\ 5 \quad 6 \quad 7 \quad 8\ ]$ having indices $n2 = [1 : 4]$

To add the sequences, we use the function as below.

```
>> [y,n]=sigadd(x1,n1,x2,n2)
```

3. Signal Subtraction

Sequence/Signals subtraction is similar to addition of signals except the function name and operator sign. Example is given below.

```
% Subtraction of sgnals
function [y,n]=sigsub(x1,n1,x2,n2)
n=min(min(n1),min(n2)):max(max(n1),max(n2));
y1=zeros(1,length(n));
y2=y1;
y1(find((n>=min(n1)) & (n<=max(n1))==1))=x1;
y2(find((n>=min(n2)) & (n<=max(n2))==1))=x2;
y=y1-y2;
```

```
Example:
```

Let $x1 = [\ 1 \quad 2 \quad 3]$ having indices as $n1 = [\ \text{-}1 : 1]$

And $x2 = [\ 5 \quad 6 \quad 7 \quad 8\ ]$ having indices $n2 = [1 : 4]$

To subtract the sequences, we use the function as below.

```
>> [y,n]=sigsub(x1,n1,x2,n2)
```

4. Signal Multiplication

Multiplication of signals or sequences involves sample by sample multiplication. After making the lengths of the vectors same, we multiply the signals using '.' (dot) operator. Here is the example.

```
% Multiplication of sgnals
function [y,n]=sigmul(x1,n1,x2,n2)
n=min(min(n1),min(n2)):max(max(n1),max(n2));
y1=zeros(1,length(n));   %initialization
y2=y1;
y1(find((n>=min(n1)) & (n<=max(n1))==1))=x1;
y2(find((n>=min(n2)) & (n<=max(n2))==1))=x2;
y=y1.*y2;   % Element wise multiplication
```

Example:

Let $x1 = [\ 1 \quad 2 \quad 3]$ having indices as $n1 = [\ -1 : 1]$

And $x2 = [\ 56 \quad 78\ ]$ having indices $n2 = [1 : 4]$

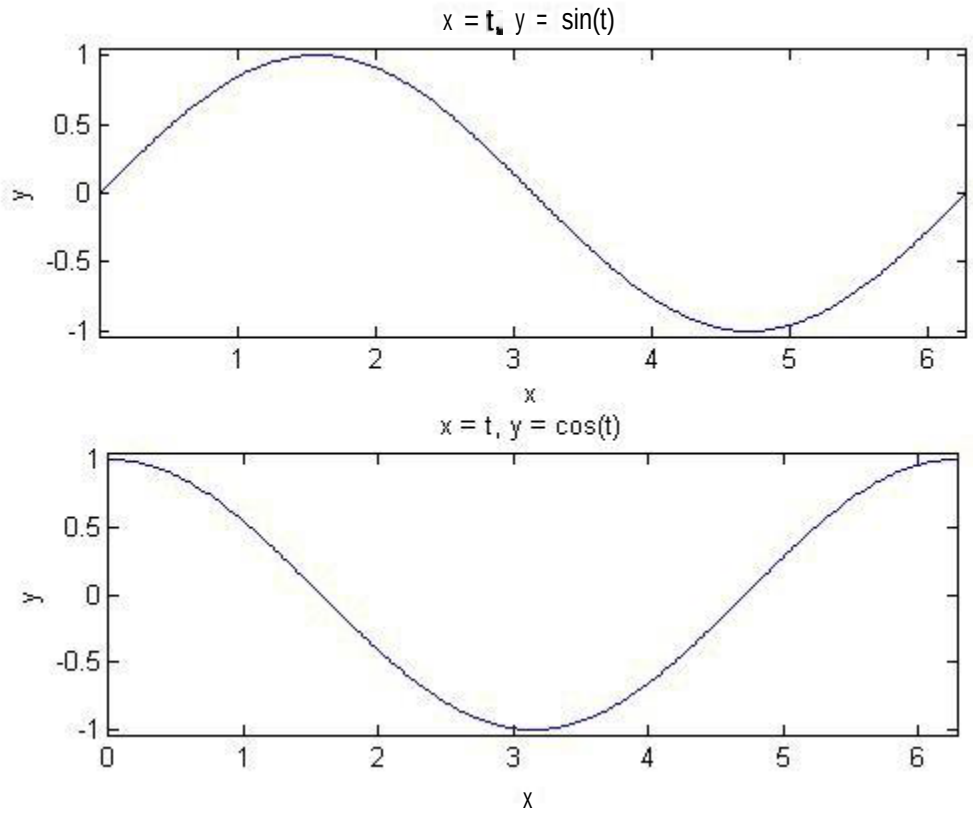
To multiply the sequences, we use the function as below.

```
>> [y,n]=sigmul(x1,n1,x2,n2)
```

5. Derivative of Signal

Derivative of a given signal is calculated using symbolic Mathematics. Following is the example to find the derivative of the signal.

```
%Derivative of signal
syms x y t f
x=sin(t);
y=diff(x);
subplot(211);
ezplot(t,x);    %command to plot in syms mode
subplot(212);
ezplot(t,y);
```



Exercise

1. Let $x(n) = \{1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1\}$. Determine and plot the following sequences.

   a) $x1(n) = 2x(n-5) - 3x(n-4)$

   b) $x2(n) = x(3-n) + x(n)(xn-2)$

2. Write a Matlab function named ***multioperations*** which returns the following output on graph simultaneously.
   a. Amplitude scaled signal
   b. Time Scaled signal
   c. Time Shifted Signal

   Alpha, beta, gamma are the variables used for defining the scales for shifting, time scaling and amplitude scaling respectively. Take the input signal $A\sin(2\pi fn+\theta)$. Alpha, beta and gamma must be taken from user on run time.

3. Write a Matlab function named ***multioperators*** which returns the following output on graph simultaneously.
   a. Sum of the signals
   b. Difference of the signals
   c. Product of the signals

   Take input from user about the operation. Then use switch statement to perform the respective operations. Use *Input()* function to take input from the user.

ISRA University Islamabad, Campus

Signals & Systems Lab

EXPERIMENT # 05*: Basic Signals*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……..

Signals and System

Lab 5|| Basic Signals

Objective:

This lab is about the basic sequences and signals and their use in generating other sequences. Followings are the signals that we will study in this lab.

1. Unit Impulse
2. Unit Step
3. Exponential Signals
4. Unit Ramp
5. Rectangular Pulse (Rect Signal)

1. Unit Impulse

Unit Impulse is defined as follow

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$$

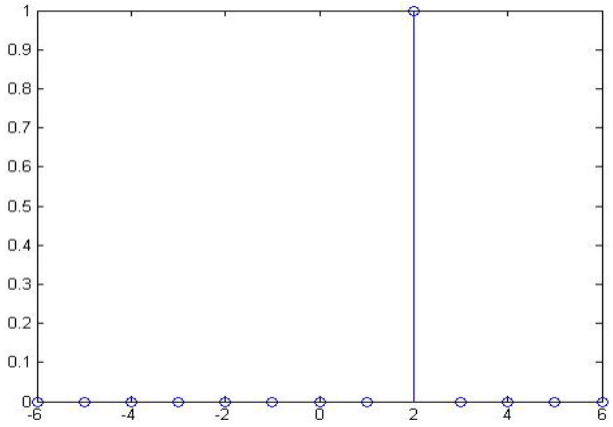
The impulse exists only on the origin point (zeroth index) and is zero elsewhere. In Matlab we implement the sequence as follows.

```
function [x,n]=impulsesig(l,no)
      n=-l:l; % defines n axis in symmetric length
      x=zeros(1,length(n)); %initialization
      x(l+no+1)=1;
```

Example:

In Command window we call this function as

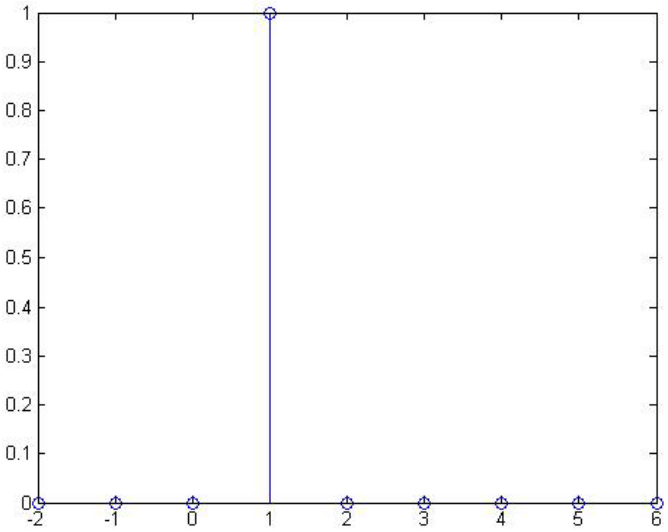
```
      [x,n]=impulsesig(6,2)
      Stem(n,x)
And following result is produced.
```



The above code works by creating a symmetric axis about zero that may not be the case always. In case where different length of the axis is required and impulse is to be placed in same pattern, we do it as follows.

```
function [x,n]=impseq(n0,n1,n2)
%Generates x(n)=delta(n-n0); n1<=n<=n2
n=n1:n2;
x=[(n-n0)==0];
```

```
Example:
 In command Window, let we want a pulse at n0=2 whereas our
axis starts from -2 and ends at 6.
We call this function as
     [x,n]=impseq(-2,6,1);
     Stem(n,x)
Following result is produced.
```



Continuous unit impulse is defined as below

$$\delta(t) = \begin{cases} 10 & t = \\ 00 & t \neq \end{cases}$$

Shifted impulse is given below.

$$\delta(t) = \begin{cases} 10 & t - to \\ 0 & t = otherwise \end{cases}$$

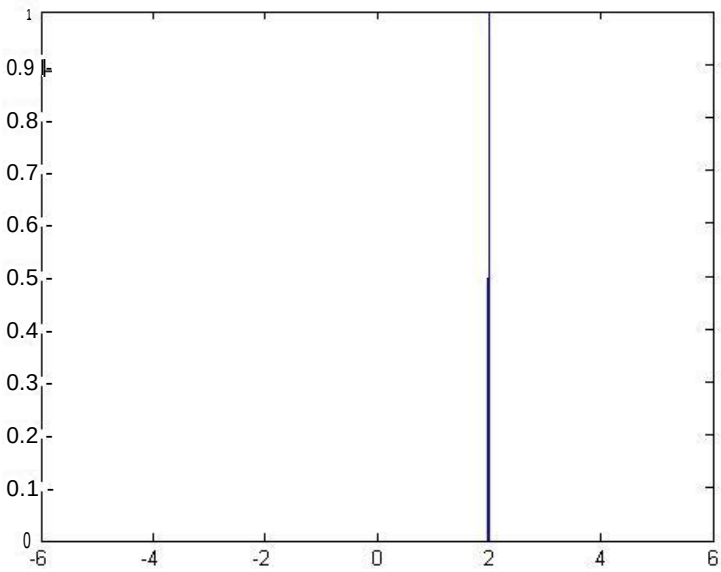
Matlab code for the CT unit impulse is given below.

```
function [x,t]=contimpulse(l,n0,inc)
t=-l:inc:l;
x=zeros(1,length(t));
x((l+n0)/inc+1)=1;
```

The *t* axis is produced in same symmetric manner except for the difference that it has been divided now in small parts, defined by the variable *inc* that makes the domain comprising of several elements of interval making the signal continuous. Changing the *inc* value will change the shape of the impulse signal. Small value will give a smoother shape of the signal.

In command window we call this function as:

```
[x,t]=contimpulse(6,2,0.01);
Plot(t,x)
Output produced is as follows.
```



The same can be done in the following way.

```
function [x,t]=cuifunct(l,no,inc)
t=-l:inc:l;
x=[zeros(1,(l+no)/inc) 1 zeros(1,(l-no)/inc)]
```

2. Unit Step Sequence

Unit Step signal is defined as follows

$$u(n) = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$

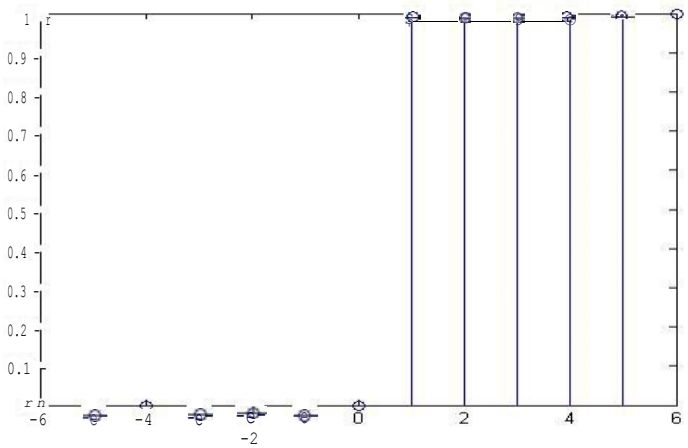
The signal has one amplitude on positive axis starting from zeroth index. Matlab code for the said signal is given below.

```
function [x,n]=dus(l,no)
n=-l:l;
len=length(n);
x=zeros(1,length(n));
x(l+no+1:len)=1;
end
```

The code produces the symmetric axis in the same way as for the unit impulse. Only difference is that now a vector of 1s is placed as amplitude in place of single impulse.

In command window run the following example.

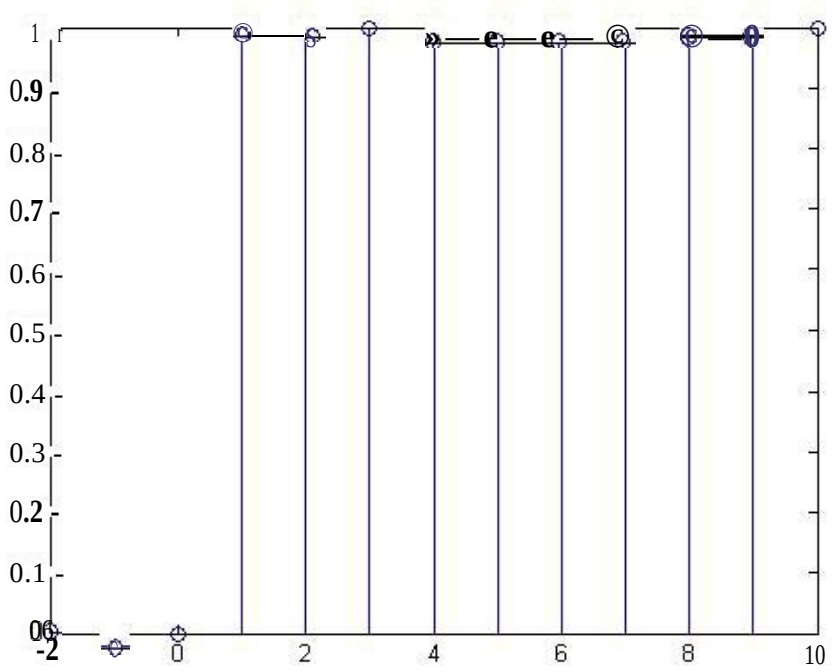
```
[x,n]=dus(6,1);
Stem(n,x)
Following output will be produced.
```



For non-symmetric n axis we can write the code as follows.

```
function [x,n]=step(n0,n1,n2)
n=n1:n2;
x=[(n-n0)>=0];
end
```

```
Example: let no=1; n1=-2; and n2= 10;
[x,n]=step(no,n1,n2);
Stem(n,x)
We will see the following output.
```



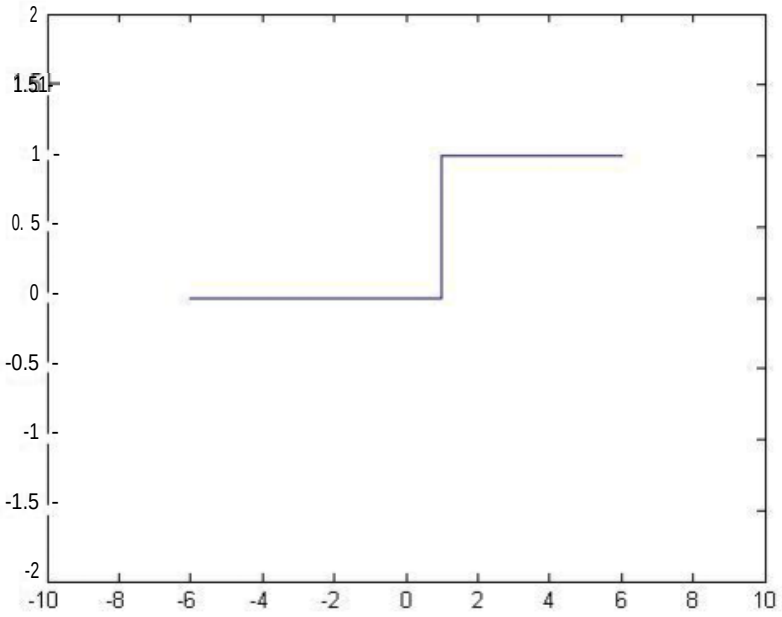
Continuous Unit step signal is defined as follows.

$$\mu(t) = \begin{cases} 10 & t \geq = \\ 00 & t < \end{cases}$$

Matlab code for implementing unit step in continuous domain is given below.

```
function [x,t]=contstep(l,n0,inc)
t=-l:inc:l;
len=length(t);
x=zeros(1,len);
x((l+n0)/inc+1:len)=1;

Example:
[x,t]=contstep(6,1,0.01);
Plot(t,x);
Axis([-10 10 -2 2]);
```

Output is given below



The same can be done using following approach.

```
function [x,t]=cusfunct(l,no,inc)
t=-l:inc:l;
x=[zeros(1,(l+no)/inc) 1 ones(1,(l-no)/inc)];
```

3. Exponential Signals

Exponential signal vary with respect to some exponent that may be real or imaginary. Two types of exponential signals are there.

1. Real exponential signal

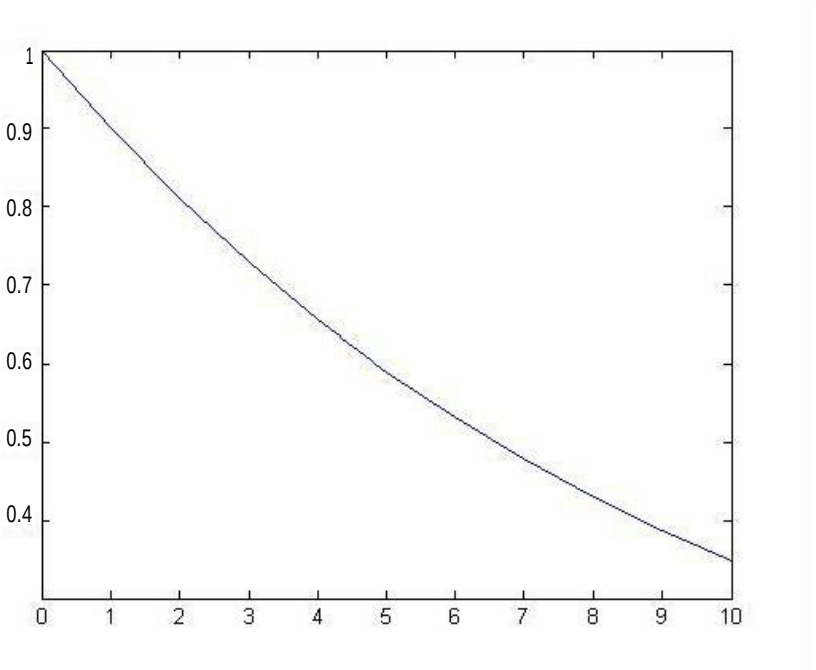
A real exponential is defined as follows.

$$X(n)=a^n \qquad \forall n, a \in \square$$

Example: $X(n)=0.9^n$ $\qquad 0 \leq n \leq 10$

We do it in Matlab as

```
n=0:10;
x=0.9.^(n);
plot(n,x)
```



The shape of the real exponential varies as decaying or rising exponentials. Depending upon the real constant, signal decays or rises.

2. Complex Exponential signals

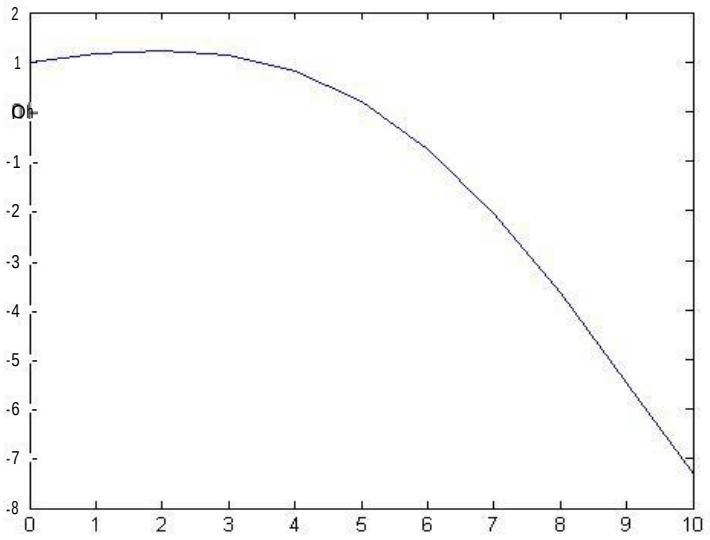
Complex valued exponential signal is defined as

$$X(n)=e^{(\alpha+jw)n} \qquad \forall\, n$$

Matlab function for plotting the complex valued exponential is given below.

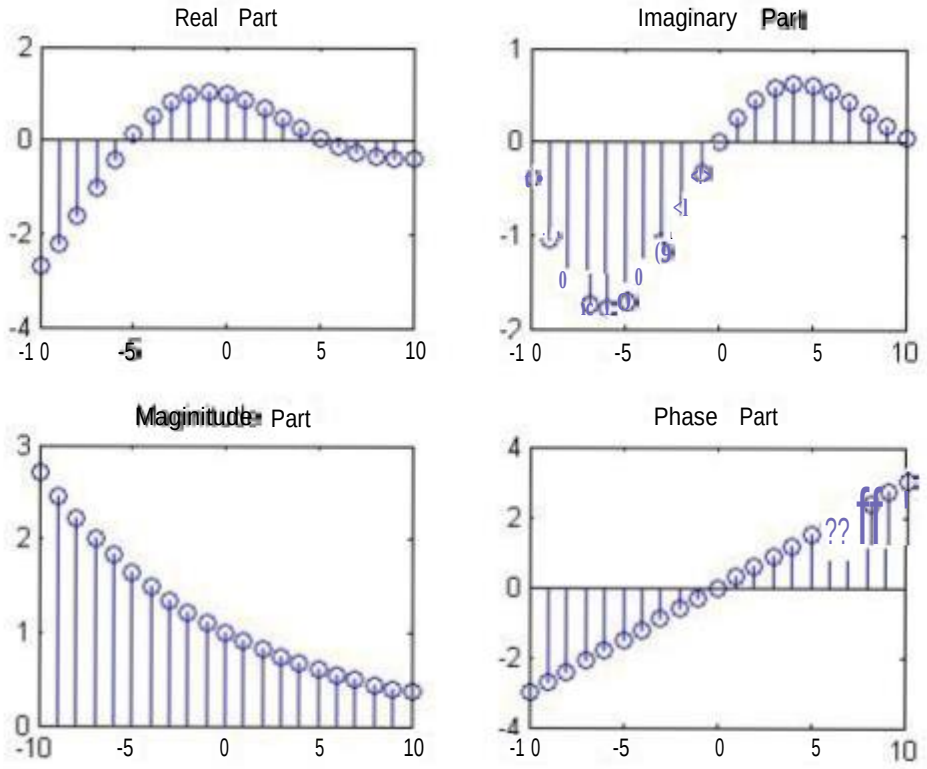
Example: $x(n)=\mathbf{e}^{\mathbf{an}}$ a=0.2+0.3j $0 \leq n \leq 10$

```
n=0:10;
a=0.2+0.3j;
x=exp(a*n);
plot(n,x)
```



Example: Given a complex exponential signal $x(n)=e^{(-0.1+0.3j)n}$. Find the real, imaginary part of the signal. Find the magnitude and phase of the signal and plot them.

```
n=-10:10;
alpha=-0.1+0.3j;
x=exp(alpha*n);
realx=real(x); %Real Part
imagx=imag(x); % Imaginary Part
magx=abs(x);    % Magnitude
phasex=angle(x); %Phase
subplot(221);stem(n,realx);title('Real Part');
subplot(222);stem(n,imagx);title('Imaginary Part');
subplot(223);stem(n,magx);title('Maginitude Part');
subplot(224);stem(n,phasex);title('Phase Part');
```



4. Unit Ramp

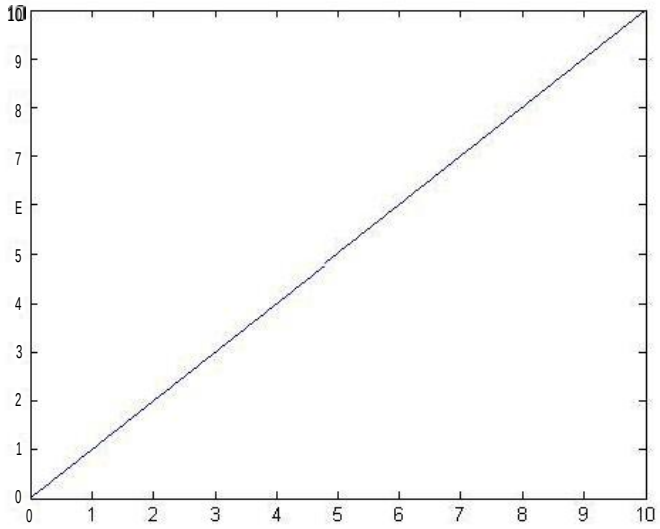
Ramp signal is defined as

$$r(t) = \begin{cases} t & t >= 0 \\ 0 & t = otherwise \end{cases}$$

The amplitude values vary as domain increases. Ramp signals exts on positive side only. Matlab Code for Unit ramp is given below.

```
t=0:0.01:10;
x=t;
plot(t,x)
```

Output signal is given below.



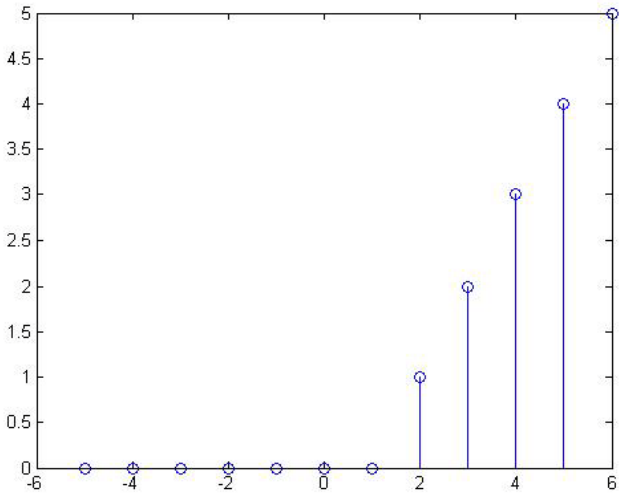
Another approach is to generate a ramp signal on shifted point or interval. Function for the shifted ramp is given below.

```
function [x,n]=rampseq(no,n1,n2)
%produces ramp sequence on axis n and starting from no
n=n1:n2;
x1=[(n-no)>=0];
x=x1.*(n-no);
end
```

Example:

```
[x,n]=rampseq(1,-5,6)
Stem(n,x)
```

Output is shown below.



5. **Rectangular Pulse**

Rect signal is defined as

$$rect = \begin{cases} 1 & -1/2 \le t \le 1/2 \\ 0 & t = otherwise \end{cases}$$

Rect signal produces a rectangular pulse of the width equal to the time interval with half of the width lying on negative side and half on positive side.

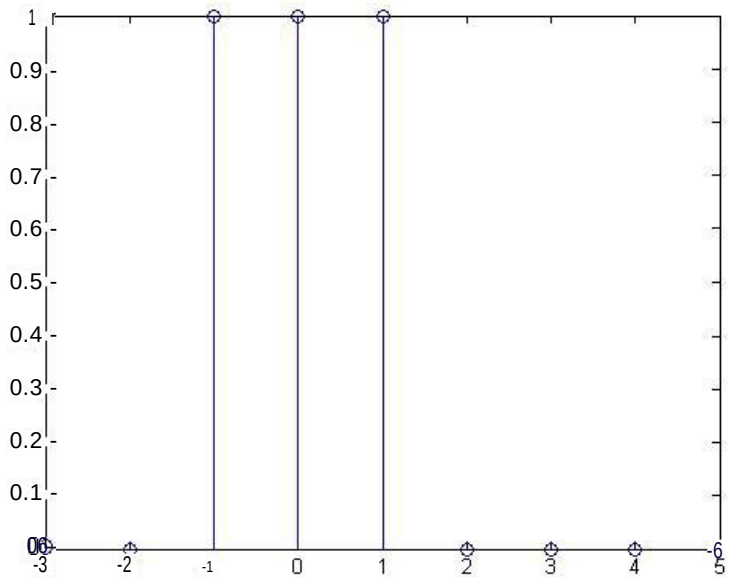
Matlab code is given below.

```
function [x,n]=rectseq2(t,n1,n2)
n=n1:n2;
x=[(n+(t/2))>=0 & (n-(t/2))<=0];
end
```

Example:

In command window use the following example

```
[x,n]=rectseq(2,-3,5);
stem(n,x);
```


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-3
-2
-1
0
1
2
3
4
5

Exercise

1. Use the **impulse** and **step** functions to implement the function y[n] given by the relation. Implement amplitude scaling through **sigscale** function.

   a. $y[n] = 2\,[n+3] - 5\,[n+6] + [n-5] - 7\,[n-2]$

   b. $y(t) = 3\,(t+4) + (t-1) - 5\,(t-2)$

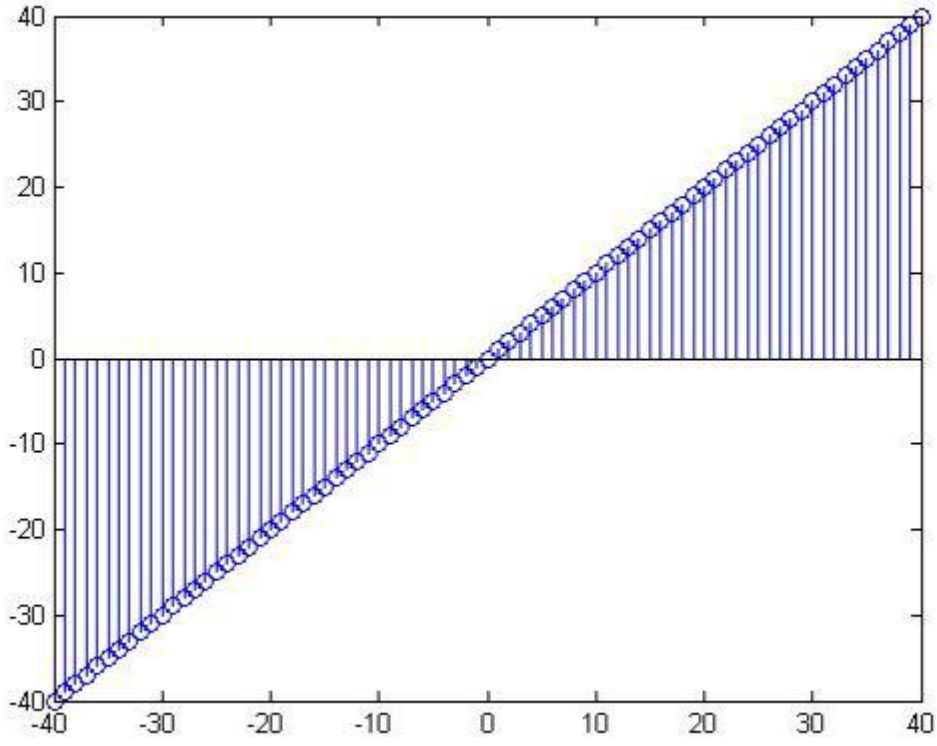
Use L=6 for y[n] and L=5 for y(t). Inc=0.01. Plot the sequences.

2. Generate and plot the following sequences over the indicate intervals.

   a. $x(n) = 2\,(n-2) - (n-4) \qquad -35 \le n \le$

   b. $x(n) = n[(n) - (n-10)] + 10e^{-0.3(n-10)}[(n-10) - (n-20)] \qquad 0 \le n \le 20$

3. Use the functions for signal flipping, amplitude scaling, signal addition and ramp to generate the following sequence.



4. Use **rect**(t) function from your manual and generate a periodic rectangular signal. Take width of the pulse =2. $-35 \le n \le$

ISRA University Islamabad Campus

Signals & Systems Lab

EXPERIMENT # 06*: Sound Manipulation and Playback*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……

Signals and Systems ||Lab-6

Playback and Sound Manipulation

Objective:

Objective of the lab is to manipulate an input audio signal by applying different operations.

Playback

An analog signal input to the sound card is sampled and digitized by sound software, such as the "sound recorder" in windows. The recorded sound signal is saved in a wav file. This file can be retrieved in Matlab and playback.

Mono/Stereo Sound

Matlab understands mono and stereo sounds as single column and 2 column vectors respectively.

Mono or monophonic describes a system where all the audio signals are mixed together and routed through a single audio channel. Mono systems can have multiple loudspeakers, and even multiple widely separated loudspeakers. The key is that the signal contains no level and arrival time/phase information that would replicate or simulate directional cues.

Stereophonic sound systems have two independent audio signal channels, and the signals that are reproduced have a specific level and phase relationship to each other so that when played back through a suitable reproduction system, there will be an apparent image of the original sound source. Stereo would be a requirement if there is a need to replicate the aural perspective and localization of instruments on a stage or platform, a very common requirement in performing arts centers.

Recording of sound

Sound can be recorded into two modes

1. Mono Mode 2. Stereo Mode

For recording of a sound we first generate an audio object by using the following command

```
ai=analoginput('winsound')
```

analoginput constructs an analog input object ai associated with the adopter 'winsound'.

Mono Mode: if you add 1 channel, the sound card is said to be in mono mode and must have hardware id of 1. Command for adding channel is as below:

```
addchannel(ai,1)
```

Stereo Mode: if you add 2 channels to a1, sound card is said to be in stereo mode. Channels are added as below.

```
Addchannel(ai,1)
Addchannel(ai,2)
Or
Addchannel(ai,1:2)
```

To delete a channel from the stereo, channel 2 must be deleted.

```
delete(ai,channel(2))
```

Resulting signal becomes mono signal.

Manipulation

Since Matlab treats all the inputs and output as matrices/vectors, therefore the manipulation of an audio signal is no different than altering the elements of matrices. Avoid using loops in manipulating a signal

Example: Make a folder in C: directory with name "soundlab". Copy the file tessound.wav (placed at network address) to the "soundlab" folder.

```
[soundsig,fs,nbits]=wavread('c:\soundlab\testsound.wav');
% Soundsig is 2-column matrix
%fs is the sampling frequency
%nbits is the no of bits per sample
left=soundsig(:,1);  % Splitting soundsig into left column matrix
right=soundsig(:,2); % Splitting soundsig into right column matrix
l=length(left);
half=left(1:2:l);    % dividing the left matrix into half
reverse=flipud(left); % flippud=flip updown
wavplay(soundsig,fs);
```

wavread returns the sound signal as 2-column matrix with sampling frequency and no. of bits per sample. Sound signal is applied different operations and then played by wavplay function.

Changing length of sound signal

To change the length of the sound signal we have to choose the required no of sample.

For example to change the sound signal from 10 sec to 4 seconds length, we use the following command

```
Cutsig=soundsig((1:4*fs),:);
```

Echo

Echo signal is the delaye and attenuated version of the sound signal which is added in the original sound wave to produce noise.

Consider the same sound signal as your input signal.We generate the echo signal for the sound using user defined delay and attenuation. Matlab function code is as below:

```
function [sigx,echosig,reflectedsig]=echogen(soundsig,fs,td,attn)
%generates an echo signal of the input sound signal
zer=zeros(td*fs,2);
sigx=[soundsig;zer];
reflectedsig=[zer;soundsig*attn];
echosig=sigx+reflectedsig;
```

Example:
In command window, we define the delay , attenuation and proceed to call the echogen function as below:

```
attn=0.5;
td=2;
[soundsig,fs,nbits]=wavread('c:\soundlab\testsound.wav');
[sigx,echo,ref]=echogen(soundsig,fs,td,attn);
wavplay(echo,fs);
wavwrite(echo,fs,nbits,'c:\soundlab\echosig');
```

The code above produces an echo signal giving a reflected sound an attenuation of 0.5 and time delay of 2 second. Wavplay function plays the echo signal with the sampling frequency fs.

Example:

Using the same sound signal file and its resulting parameters, try the following.

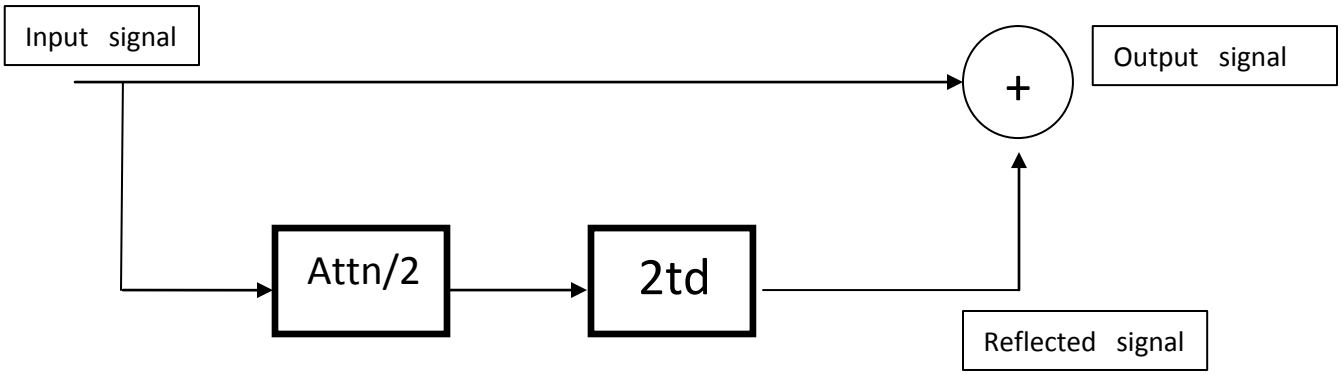
1. wavplay(soundsig,fs)
2. wavplay(soundsig,0.5*fs)

3. wavplay(soundsig,2*fs)
4. wavplay(half,fs)
5. wavplay(half,0.5*fs)
6. wavplay(half,2*fs)
7. wavplay(reverse,fs)
8. wavplay(reverse,0.5*fs)
9. wavplay(reverse,2*fs)
10. wavplay(2*soundsig,fs)

Turn on your speakers/headphones and hear the entire above signal and note the differences between each. Last command does increases the volume of the sound signal.

Exercise

1. Implement the **echogen** function using following parameters and write down the differences which you find in audio output.
    a. Att=0.5, td=0.5
    b. Att=0.3, td=1
    c. Att=0.1. td=2
2. Write down the code for the output of the following system. Take same audio input file that is placed on your network address.



3. Take the audio file as input in Matlab. Cut the resulting signal sound signal into one third of the original signal and save the new signal at fs=22050 and nbits=8. Playback the new file and state the difference between the two sounds. Mention code as well.

ISRA University Islamabad Campus

Signals & Systems Lab

EXPERIMENT # 07*: Convolution and Laplace Transform*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……

Signals and Systems || Lab 7

Convolution and Laplace Transform

Objective

The theme of this lab to perform the convolution for finding the output of an LTI system using system's impulse response and input signal. Second part of the lab comprises transfer function representation and Laplace transformation.

Convolution

Consider a discrete time system with input x[n] and output t[n]. Output y[n] is computed output of the system through impulse response which is the output of the system when the input is unit impulse.

When Impulse response isgivenwe can find out the system output by following relation

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

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

For continuous signal, output is computed through following relation.

$$y(t) = \int_{-\infty}^{\infty} x(T)h(t-T)dT$$

Convolution using Matlab

To find out the output through convolution, firstly vector x[n] and h[n] are to be defined. Then the output is computed through the following command

$$yn = conv(x, h)$$

Command **conv(x,h**) assumes that the 1st element in x correspond to the n=0, 1st element in vector h corresponds to n=0. SO the 1st element in yn will also correspond to n=0.

The command **Conv ()** can also be used to multiply polynomials.

Suppose the coefficients of the polynomial a are given in vector A and that of b are given in B. then coefficients of the output polynomial can be found out as:

For Example
a(s)=S+1
b(s)=S+2

then
A=[1 1];
B=[1 2];
ab=conv(A,B)

output comes out to be
ab=[1 3 2]

Example:

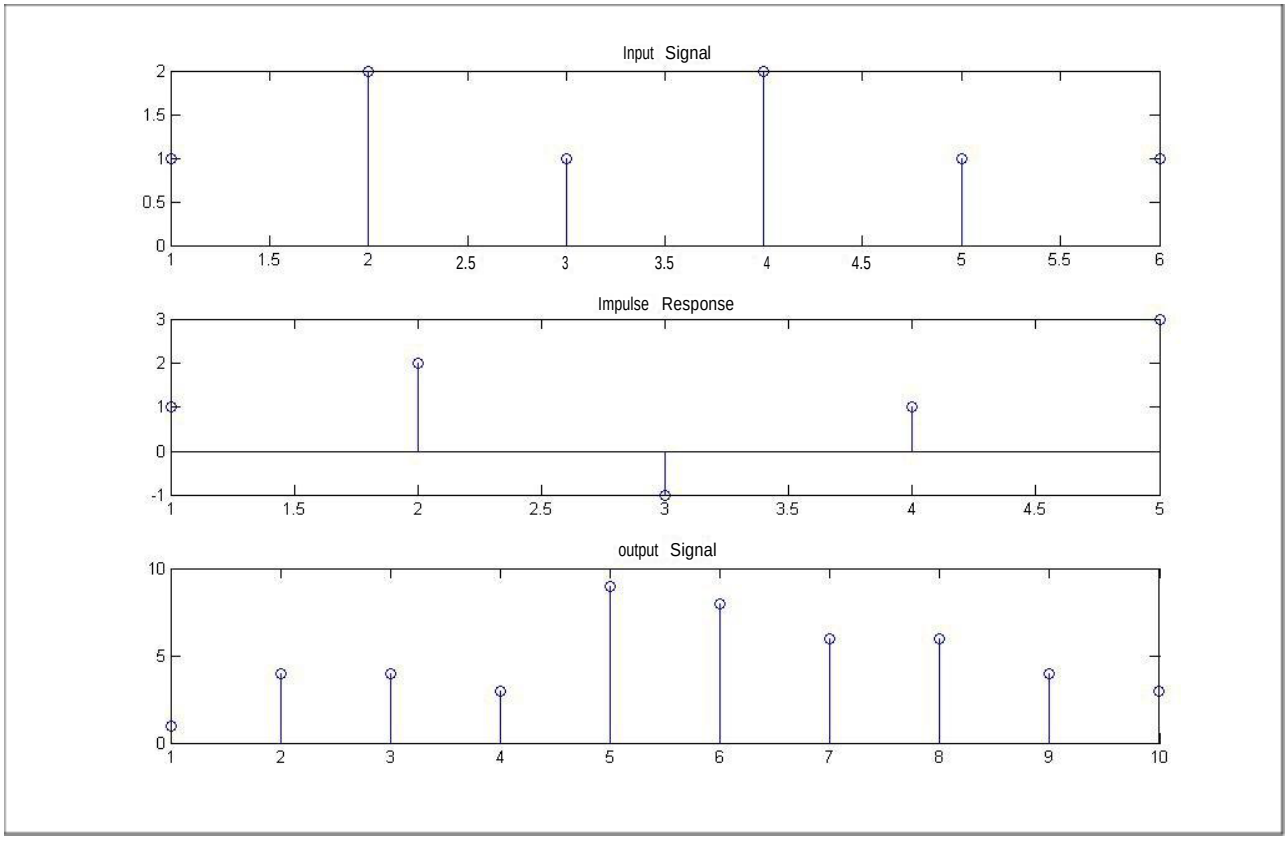
Given the following input signal for discrete LTI system and impulse response.

$x[n] = [1, 2, 1, 2, 1, 1]$

$h[n] = [1, 2, -1, 1, 3]$

Code to find the convolved signal is given below.

```
x=[1,2,1,2,1,1];
h=[1,2,-1,1,3];
y=conv(x,h);
subplot(311);stem(x);title('Input Signal');
subplot(312);stem(h);title('Impulse Response');
subplot(313);stem(y);title('output Signal');
```



If x[n] and h[n] are of different lengths or different starting points then the output will be computed correctly but the indices would have to be adjusted. For example: if x[n] starts from n=-1 and h[n] starts from n=-3 then the output signal will start from n=-4.

Example: Find the output of the LTI system when x(n)= {1,1,2,1,1,3} and h(n)={1,2,-1,1}

Matlab code for this scenario is given below.

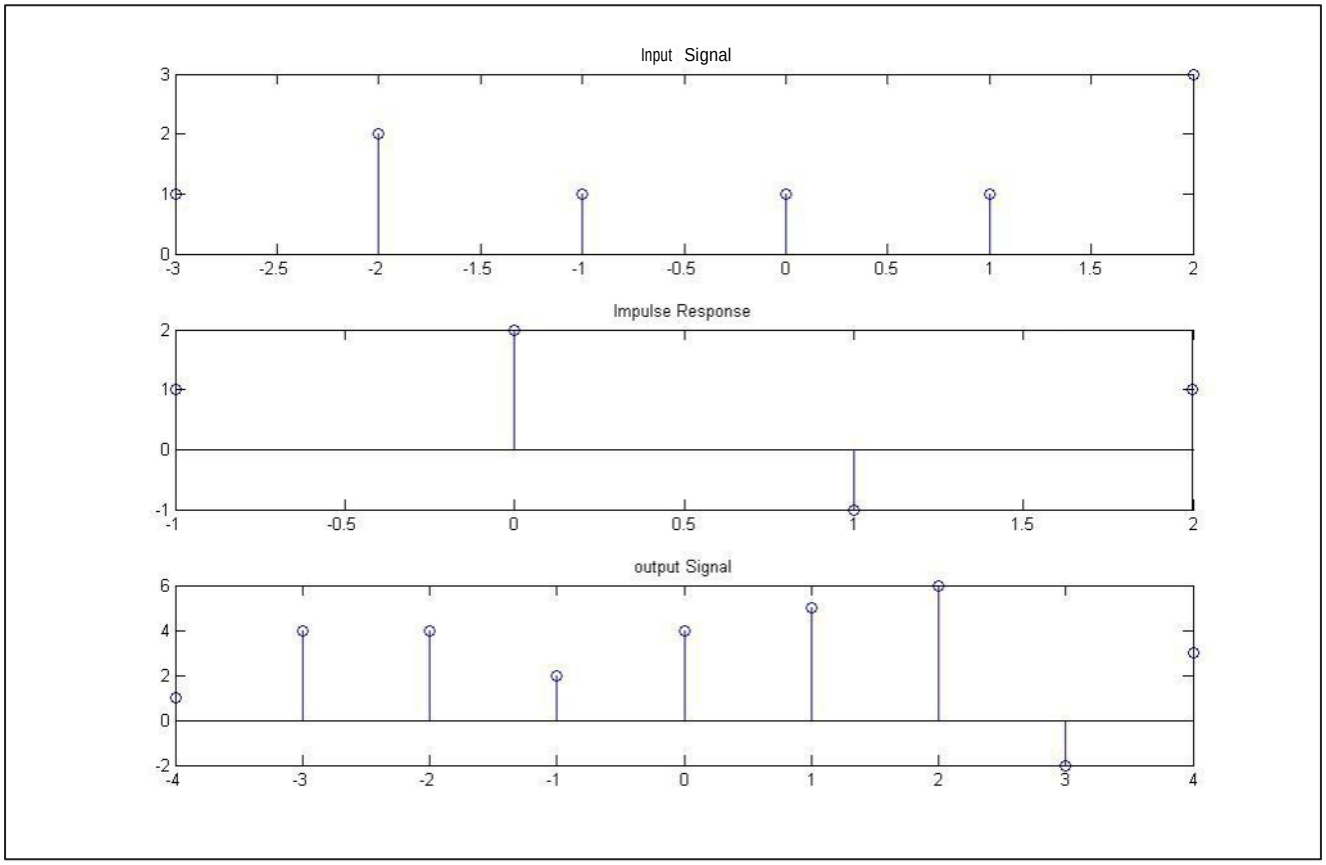
```
function [y,k]=conv_sig(x,n,h,m)
%finds the output for different lengths of input and impulse response

ki=n(1)+m(1);      %Starting index of the output
ke=n(length(n))+m(length(m));   %Ending index of the output
k=ki:ke;          % Output indices
y=conv(x,h);
```

In command Window:

```
x=[1 2 1 1 1 3];
n=[-3:2];
h=[1 2 -1 1];
```

```
m=[-1:2];
[y,k]=conv_sig(x,n,h,m);
subplot(311);stem(n,x);title('Input Signal');
subplot(312);stem(m,h);title('Impulse Response');
subplot(313);stem(k,y);title('output Signal');
```

Output is shown below



In case of continuous signals, we find the convolution in the same manner.

Example: Find the output when

$$x(t) = \alpha^t u(t)$$

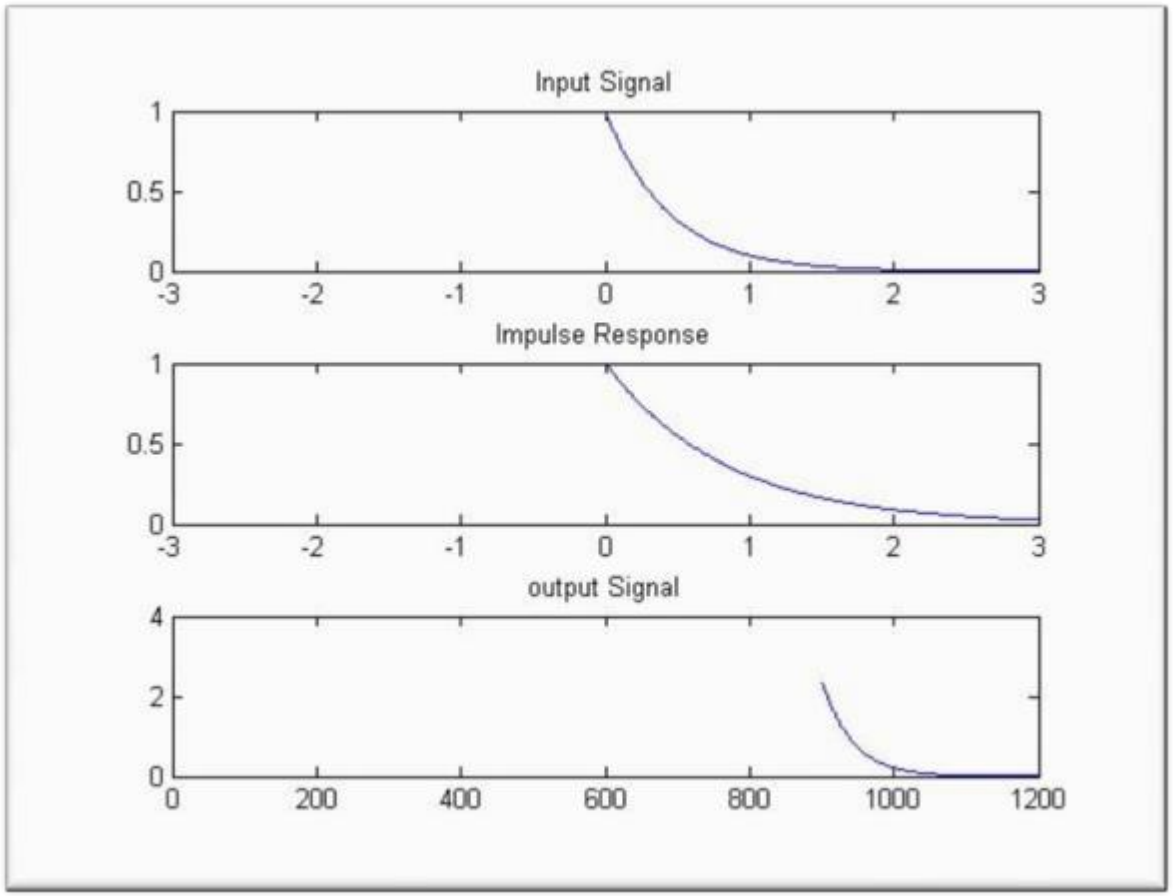
$$h(t) = \beta^t u(t)$$

where

$$\alpha\beta = 0.1, \quad = 0.3 And - \leq\leq 3$$

Matlab code is

```
alpha=0.1;
t=-3:0.01:3;
beta=0.3;
x=(alpha).^t.*heaviside(t);
h=(beta).^t.*heaviside(t);
y=conv(x,h);
subplot(311);plot(t,x);title('Input Signal');
subplot(312);plot(t,h);title('Impulse Response');
subplot(313);plot(T,y);title('output Signal');
```



Transfer Function Representation

Transfer function of a system is defined as the ratio between Output and Input.

Transfer functions are defined in MATLAB by storing the coefficients of the numerator and denominator in vectors.

Generally a transfer function is represented as

$$H(s) = B(s) / A(s)$$

Where B and A are defined as

$$B = b_m s^m + b_{m-1} s^{m-1} + b_m s^m + ...$$
$$A = a_m s^m + a_{m-1} s^{m-1} + a_m s^m + ...$$

Coefficients of A and B are stored in vectors and then used forward.

Example: Given the following transfer function,

$$H(s) = \frac{2s^2 + 3}{s^3 + 4s^2 + 5}$$

We implement the transfer function in Matlab as

```
num=[2  0 3];   %Coefficients of the numerator
den=[1        4        0        5];        %Coefficients        of        the        denominator
```

To convert the vectors in transfer function we use

```
h=tf(num,den);  %Transfer function

h

Transfer function:
  2 s^2 + 3
---------------
s^3 + 4 s^2 + 5
```

Zero,poles and gain can be found using the following

```
[z,p,k]=tf2zp(num,den};
z =

      0 + 1.2247i
      0 - 1.2247i
```

```
p =

  -4.2737
   0.1369 + 1.0729i
   0.1369 - 1.0729i


k =

     2
```

Transfer function can be obtained back from zero poles and gain.

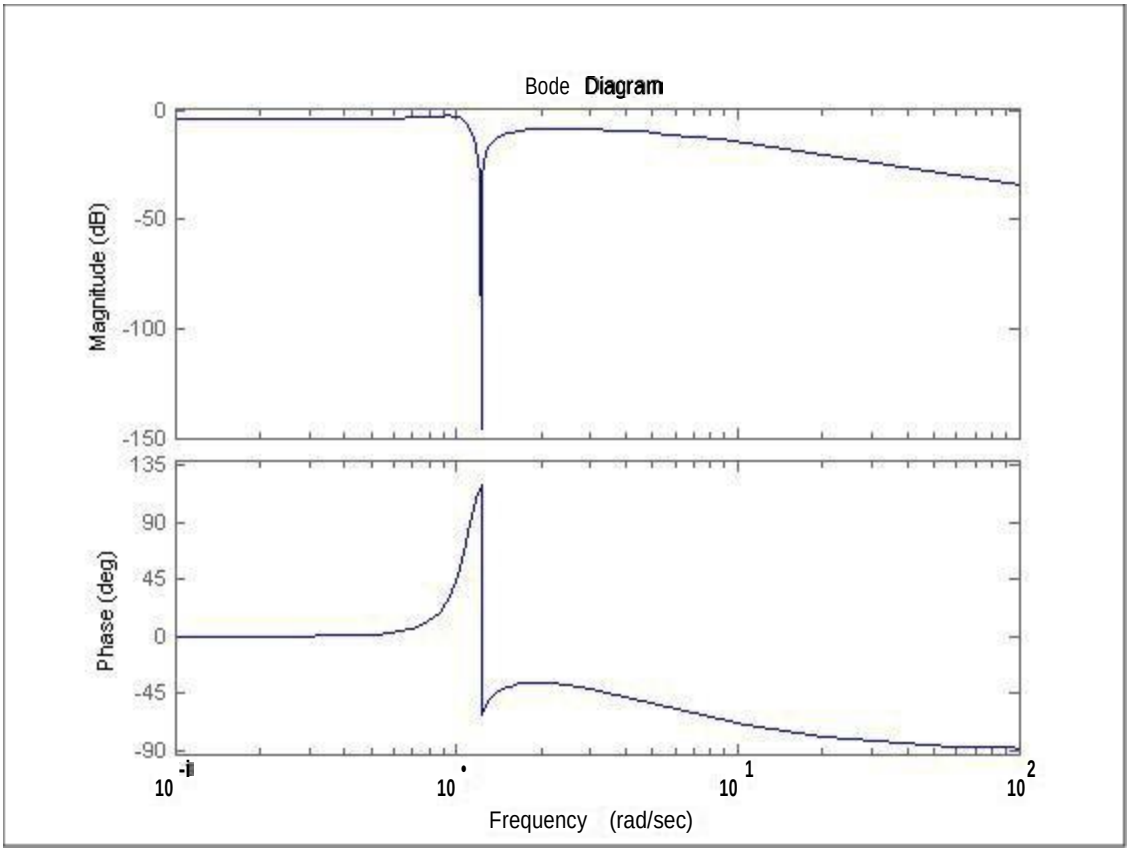
```
[num,den]=zp2tf(z,p,k);
num =

       0    2.0000         0    3.0000


den =

    1.0000    4.0000         0    5.0000

tf(num,den)

Transfer function:
   2 s^2 + 3
---------------
s^3 + 4 s^2 + 5
```

Now suppose we want to check the response of the system against a range of frequencies, we can this response using following

```
w=pi:pi/1000:2*pi*f;
[h,k]=freqz(num,den,w);
```

To draw the poles,zeros,gain and phase of the transfer function, we draw bode plot using following function

```
bode(num,den)
```



Laplace Transform

Laplace transform is used to transform a function/signal from time domain to s domain (complex domain). System can be analyzed and performed different operations in S domain and then can be taken back into t domain. Laplace transform converts the function expression into simple algebraic polynomial. Following expression is used to calculate the Laplace transform of a time domain function f(t).

$$F(S) = \int_{-\infty}^{\infty} f(t)\, e^{-st} dt$$

In Matlab we solve the Laplace transform using symbolic math toolbox. Function used for finding the Laplace is Laplace().

Example:
Let f(t)=t and we have to find the Laplace transform of the function. Matlab code for this is given below.

```
syms t
f=t;
```

```
F=laplace(t)
```

Output returned is
$F=1/S^2$

Example:
Laplace transform of e^{-at} is calculated as:

```
Syms a t s
f=exp(-a*t);
F=laplace(f);
```
Output comes out to be

```
F=1/s+a
```

Inverse Laplace Transform

Laplace transform of any time domain function can be returned back to time domain by using inverse Laplace transform. Following is the relation for the inverse laplace transform

$$f(t) = \frac{1}{2\pi i}\lim_{T\to\infty}\int_{-iT}^{+iT} F(S)e^{st}ds$$

In Matlab we find out the inverse laplace transform using symbolic math toolbox. ilaplace is function for finding the inverse laplace.

Example:
Given below the transfer function

$$H(S) = \frac{(S+3)}{(S+1)(S+2)}$$

Find the inverse laplace.
Matlab Code is given below

```
Syms s
H=(s+3)/ ((s+1)*(s+2));
F=ilaplace (H);
```

Output comes out as:

```
F=2*exp (-t)-exp (-2*t)
```

In finding the inverse Laplaceoften we encounter improper fractions having numerator power greater than denominator. For such fractions we first divide the polynomials (numerator and denominator) and then make partial fractions before taking inverse Laplace.

Example:
Find the inverse Laplace of the following transfer function

$$H(s) = \frac{S^3 + 3S^2 + 3}{S^2 + 8S + 15}$$

In Matlab we solve it as:

```
num=[1 3 0 3]; % Co-efficients of numerator
den=[1 8 15];  % co-efficients of denominator
[Q,R]=deconv(num,den);
% Divisions of two polynomials to make improper fraction a proper fraction
[r,p,k]=residue(num,den);
%Partial fraction of the transfer function

%By residue we get the following parameters
% r =

   23.5000
    1.5000


p =

   -5
   -3


k =

        1        -5 %
```

From values above we write Matlab expression for h(s) as

```
syms s
h=(s-5)+(23.5/(s+5))+(1.5/(s+3));
f=ilaplace(h);
```

Output comes out to be

```
f = dirac(1,t)-5*dirac(t)+47/2*exp(-5*t)+3/2*exp(-3*t)
```

Note:

1. Deconv returns the output as quotient (Q) and Remainder ®
2. Residue converts the fraction to partial fraction and returns three vectors r (constants of numerators), p (poles) and K (direct terms).
3. Dirac is the built-in function for unit impulse.

Exercise

1-Consider the following two discrete functions:

$$x[k] = \begin{cases} fe+i, & 0 \le f\, d \le 4\ ,4, \\ 0, \text{ otherwise.} \end{cases} \quad \text{an d} \quad h[k] = \begin{cases} 1 - f\, c k, & 0 \le f\, d \le 3\ 3, \\ 0. \text{ otherwise.} \end{cases}$$

a) Plot the two functions in the same figure using the Matlab stem.
b) Using the MATLAB conv function, find the convolution of the two functions. Plot the convolution output using the stem.
c) Verify the result obtained in the previous question using hand calculations. Use the graphical method discussed in class.

2- Find the convolution of

(i) $2tu(t)$ an d $t^3u(t)$

(ii) $e^t u(t)$ an d $tu(t)$

(iii) $e^{-2t}u(t)$ an d $e^{t}u(t)$

Plot both the functions and their convolved output in the same figure using subplot.

3- Given x [n] = u [n] – u [n – 10] and h [n] = {1,-2, 4, 6, -5, 8, 10}.

↑

Find y[n] = x[n]*h[n]. Plot x[n], h[n] and y[n] using stem command in 3 subplots

4- Find the inverse Laplace of the following systems.

(i) $\frac{1}{s^2(s+1)}$ (ii) $\frac{1}{(s-1)(s-2)}$ (iii) $\frac{1}{(s^2+1)^2}$

5- Given

$$H(s) = \frac{2 + (2s+1)(s^2+4)}{(s^2+4)(s^2+1)}$$

a) Find the partial fraction expansion of H(s) using Matlab and also write the expression with the help of Matlab output.

b) Find the quotient and remainder of H(s) using Matlab.

c) Find h (t), the Inverse Laplace Transform of H(s) using Matlab from partial fraction expansion.

d) Compute the Inverse Laplace Transform directly using Matlab symbolic toolbox and compare you result with part (c).

ISRA University Islamabad Campus

Signals & Systems Lab

EXPERIMENT # 08*: Periodicity, Harmonics and Fourier*

Series

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor’s Signature: ……..

Signals and Systems

Lab8|| Periodicity, Harmonics and Fourier series

Objective:

The theme of the lab is to check the periodicity of the given signal, making a signal periodic , checking its various harmonics and use them to generate a Fourier series of a particular periodic signal.

Sections are described as follows:

Periodicity

Periodic signals are very common class of the signal that we encounter in various systems. A periodic continuous time signal has the property that there is a positive value t for which

$$X(t) = X(t + aT)$$

Where a=0,1,2,3,4….. for all values of t.

In other words, a periodic signal holds the property of being unchanged by a time shift of T, where T is the time period of the signal.

A periodic signal in discrete time is defined similarly to continuous time signal. Specifically a discrete time signal is periodic with period N where N is positive integer. We may define a discrete time periodic signal x(n) as

$$X[n] = X[n + bN]$$

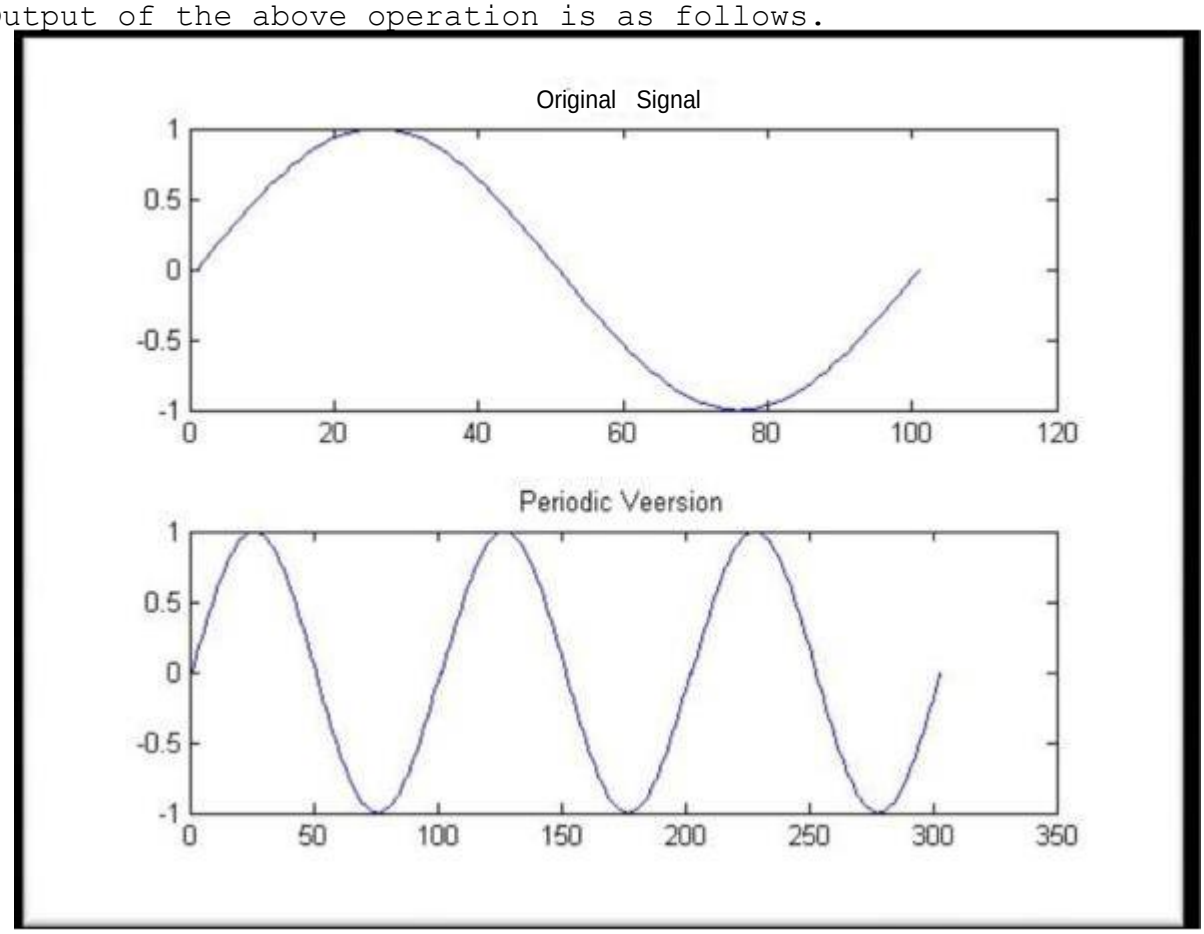
Where b=0,1,2,3,4

Let us assume we have a sinusoidal signal $x(t) = \sin(2 ft)$ with frequency of 1Hz. We can generate its periodic version as follows.

```
% Code to represent a periodic signal
f=1;              %Single cycle
x=sin(2*pi*f*t); %Original signal
y=x'*ones(1,3);  %  Copies of the original signal
y1=y(:);          %Converting the copied signal into column form
```

```
per_sig=y1';      % Copied vectors in a row
subplot(211);plot(x);title('Original Signal');
subplot(212);plot(y2);title('Periodic Veersion');
```

Output of the above operation is as follows.



Same process holds for discrete time signal as well.

Fundamental Period

The fundamental period is the smallest positive value of T (continuous) and N (discrete) for which the equation of periodicity holds. Consider an example of sine function. In this case signal repeats itself after $2,\ 4,\ 4,\ 8$. But the fundamental period is 2 .

Harmonics

A harmonic of a wave is a component frequency of the signal that is an integer multiple of the fundamental frequency, i.e. if the fundamental frequency is *f*, the harmonics have frequencies 2*f*, 3*f*, 4*f*, . . . etc.

The harmonics have the property that they are all periodic at the fundamental frequency, therefore the sum of harmonics is also periodic at that frequency. Harmonic frequencies are equally spaced by the width of the fundamental frequency and can be found by repeatedly adding that frequency. For example, if the fundamental frequency is 25 Hz, the frequencies of the harmonics are: 50 Hz, 75 Hz, 100 Hz etc.

Even and odd Harmonics

If the harmonic of a signal function has even coefficients,it is called an even harmonic. Otherwise it the coefficients is odd, it is called odd harmonics.

For example

$x(\ ) = \cos(\ \ t)$ is any given signal

Then

$\cos(\omega t)$, $\cos(3\ \ t$ ⎮ $t)$, $\cos(7\ \ t)$..... are odd harmonics

$\cos(2\omega t)$, $\cos(4\ \ t$ ⎮ $t)$, $\cos(8\ \ t)$..... are even harmonics.

Different waveforms can e obtained by adding one type of harmonics. Foe example: adding infinite cosines/sine generates a square wave. Both will have 90 degree phase shift from each other. Particularly it is not possible to add infinite number of harmonics, but as we increase the number of harmonics, a decent waveform can be obtained.

Example:

Let us assume that we want to find the first 4 harmonics of the following signal.

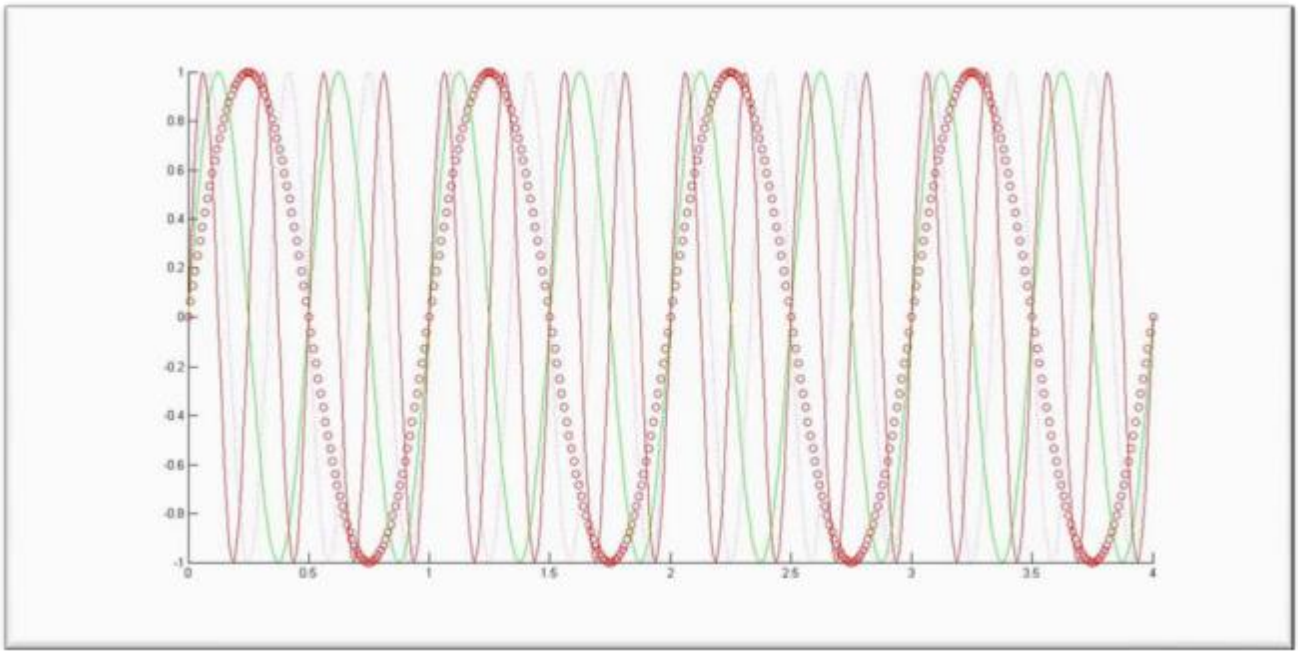
$$x(\ ) \quad = \sin(\ \ t)$$

We proceed as follows

```
% Code to represent the Harmonics of a signal
inc=0.01;
t=0:inc:4;
w=2*pi*f;
fundamental=sin(w*t);    %fundamental harmonic
har2=sin(2*w*t);         %2nd harmonic
har3=sin(3*w*t);         %3rd harmonic
har4=sin(4*w*t);         %4th harmonic
hold on;
plot(t,fundamental,'ro');
plot(t,har2,'g');
```

```
plot(t,har3,'m:');
plot(t,har4,'r');
hold off;
```

Output is given below.



Same harmonics can be found as follows

```
% Code to represent the Harmonics of a signal
inc=0.01;
t=0:inc:4;
w=2*pi*f;
n=1;    %1st harmonic
fundamental=sin(w*n*t);
n=2;    %2nd Harmonic
har2=sin(w*n*t);
n=3;    %3rd harmonic
har3=sin(w*n*t);
n=4;    %4th harmonic
har4=sin(w*n*t);
hold on;
plot(t,fundamental,'ro');
plot(t,har2,'g');
plot(t,har3,'m:');
plot(t,har4,'r');
hold off;
```

A for loop can also be used for generation of harmonics.

Fourier Series

Fourier series can be stated as *any periodic signal can be expressed as a series of harmonically related sinusoids whose frequencies are integer multiple of the fundamental frequency.*
It was discovered by a French mathematician Fourier.
General representation of the Fourier series is given below.

$$f(x) = a_0 + \sum_{n=1}^{\infty} (a_n \cos nx + b \sin nx)$$

Where a_o,a_n and b_n are the coefficients of constant term, cosines and sines respectively and are defined as follows.

$$a_0 = \frac{1}{2T}\int_{-T}^{T} f(x)\,dx$$

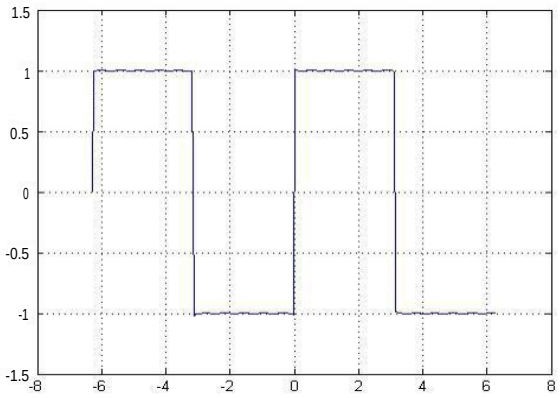
$$a_n = \frac{1}{T}\int_{-T}^{T} f(x)\cos nx\,dx$$

$$b_n = \frac{1}{T}\int_{-T}^{T} f(x)\sin nx\,dx$$

Above expressions of the coefficients hold for the signal having Time period of T.

In Matlab we implement the Fourier series by finding out the coefficients of the fourier series and implementing the function.

Example:
Find the Fourier series of the following signal



Calculation for the Fourier coefficients yield the following

$$a_0 = \quad |$$

$$a_n = \quad |$$

$$b_n = \frac{2}{n}(1 - \cos \quad)$$

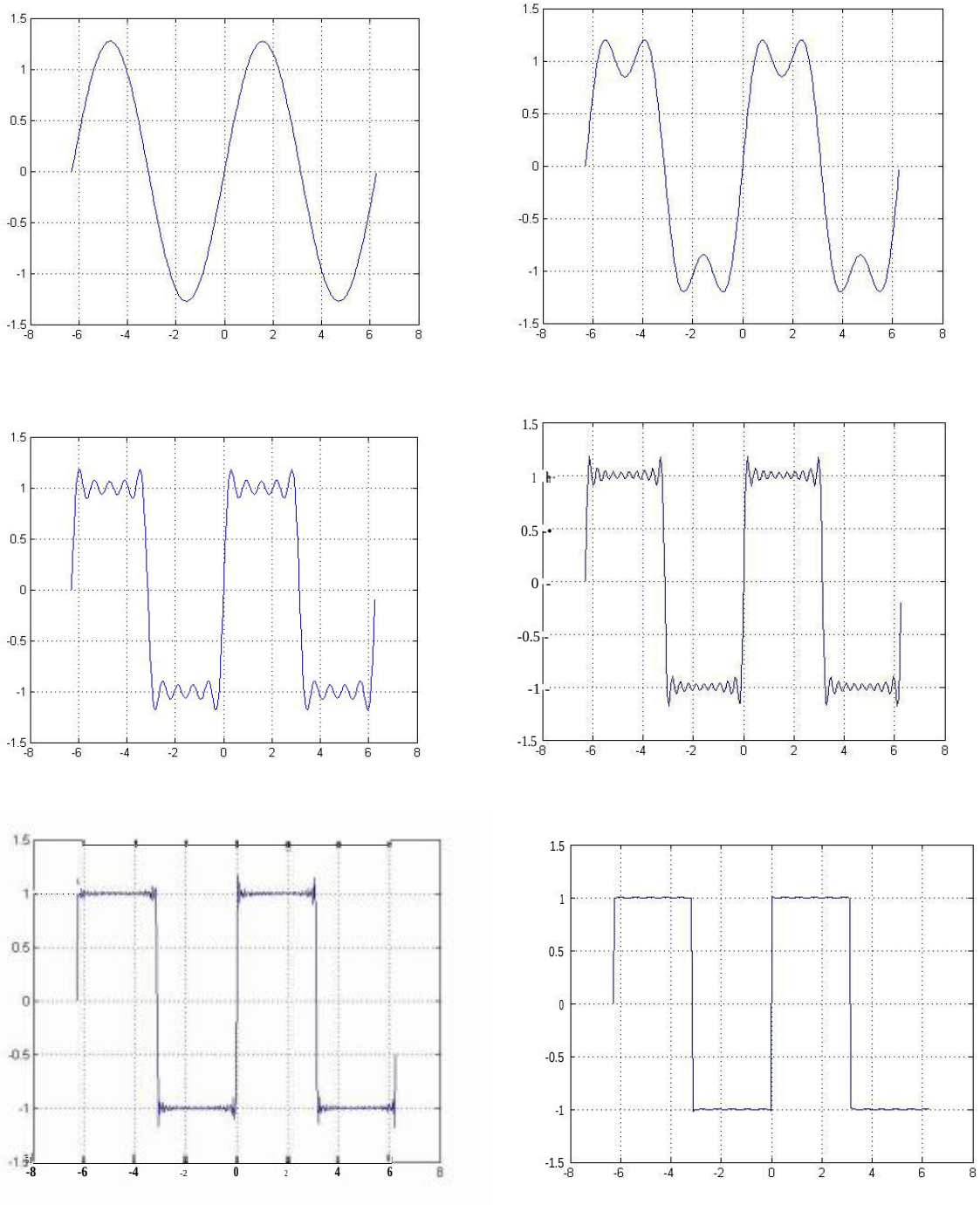
Matlab code is as follows

```
function [t,frsr]=sqrfs(har,tmax)
fo=1/(2*pi);
wo=2*pi*fo;
t=-tmax:0.05:tmax;
n=1:har;
ao=0;
an=zeros(1,length(n));
bn=(2./(pi*n)).*(1-cos(n*pi));
y=zeros(har,length(t));
for n=1:har,
    y(n,:)=an(n)*cos(n*wo*t)+bn(n)*sin(n*wo*t);
end
frsr=ao+sum(y,1);
```

Command Window Part is as follow

```
tmax=2*pi;   % Time period of the signal is 2pi
for i=1:6,
    har=[1 3 10 20 50 2000];
    [t,f]=sqrfs(har(i),tmax);
    plot(t,f);
    grid;
    pause;
end
```

Output is as shown below.

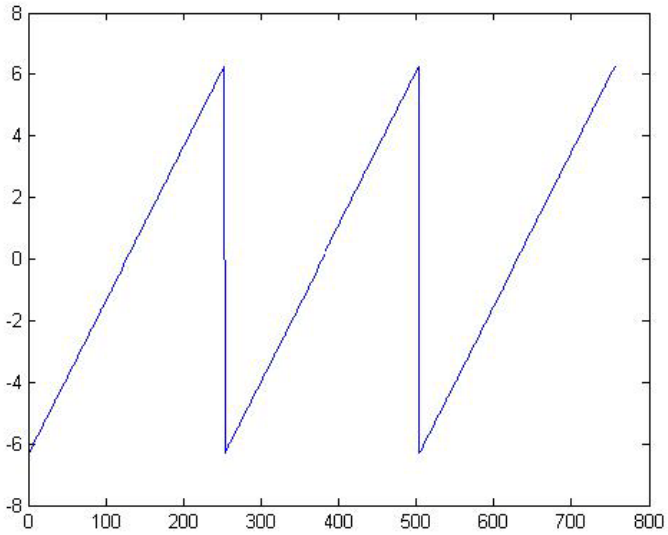


Explanation:

Output shows different harmonics added up to produce the final output, that is the given signal. As number of harmonic increases the discontinuity (ripples) move towards the corner.

Example:

Find the fourier series of the following sawtooth wave.



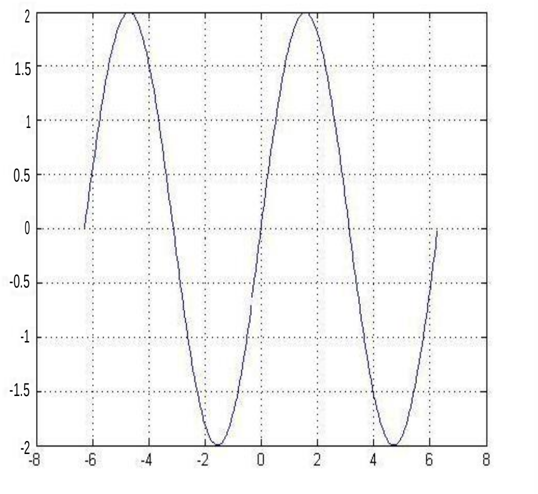
Matlab code is follows.

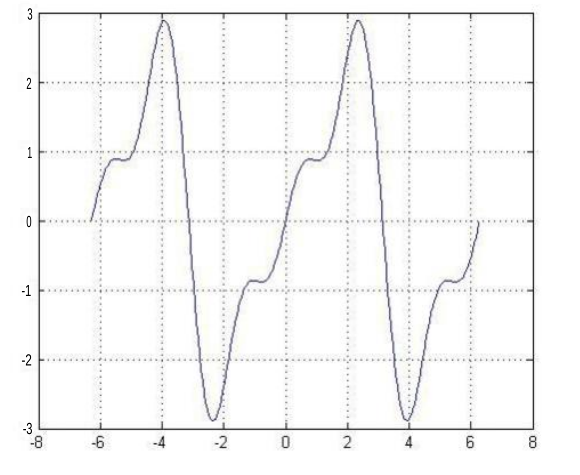
```
function [t,frsr]=sawtooth(har,tmax)
fo=1/(2*pi);
wo=2*pi*fo;
t=-tmax:0.05:tmax;
n=1:har;
ao=0;
an=zeros(1,length(n));
bn=(2./n).*(-1).^(n+1);
y=zeros(har,length(t));
for n=1:har,
    y(n,:)=an(n)*cos(n*wo*t)+bn(n)*sin(n*wo*t);
end
frsr=ao+sum(y,1);
```

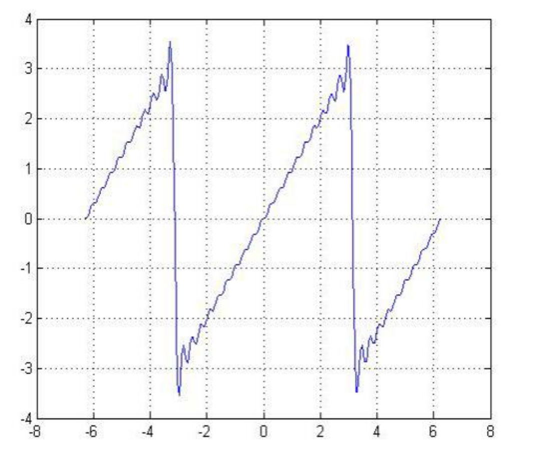
Command window part is as follow.

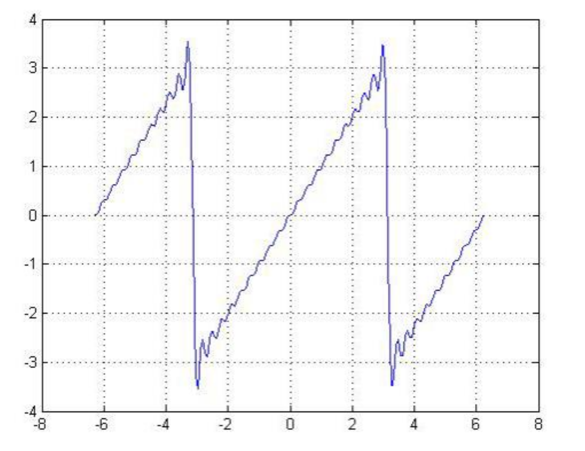
```
tmax=2*pi;   % Time period of the signal is 2pi
for i=1:6,
    har=[1 3 10 20 50 2000];
    [t,f]=sawtooth(har(i),tmax);
    plot(t,f);
    grid;
    pause;
end
```

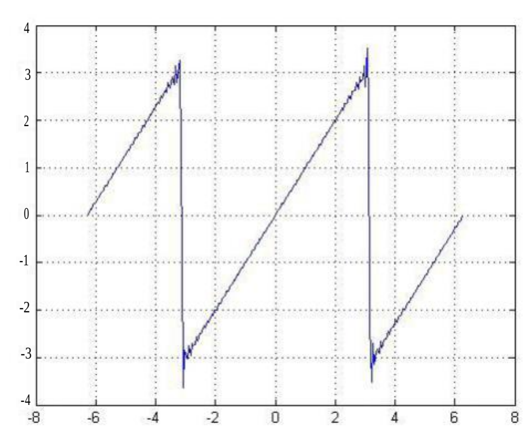
Output is as shown below.

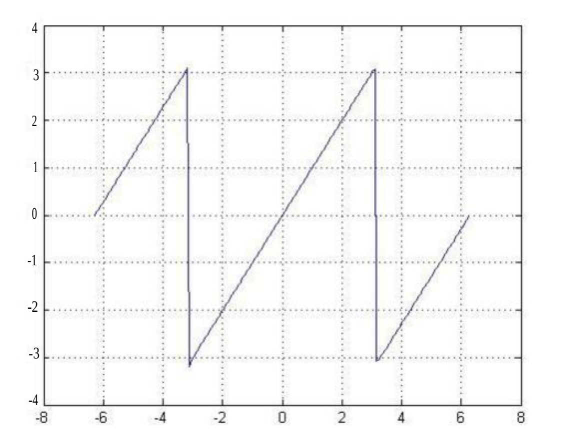












Shorten Calculations

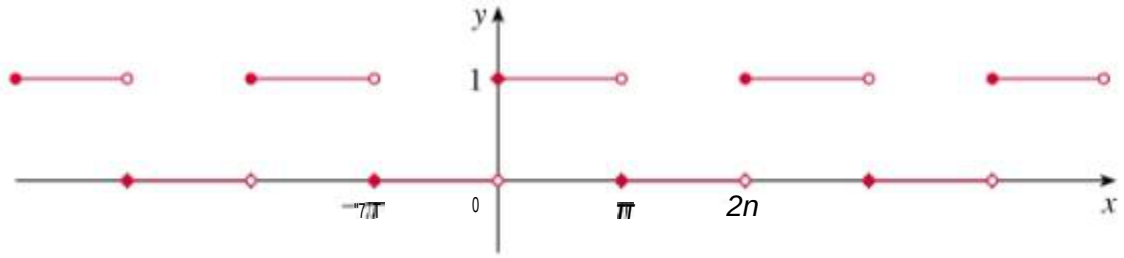
Fourier Series coefficients can be calculated by using following tips.

1. If f(t) is even, bn will be zero. For example cos(t) is even signal
2. If f(t) is odd , an will be zero. For example sin(t) is odd signal.
3. If signal is equally spaced from the base lene, ao will be zero.
4. Coefficients can be calculted using symbolic maths toolbox

Exercise

Q1: Find the Fourier coefficients and Fourier series of the following function.

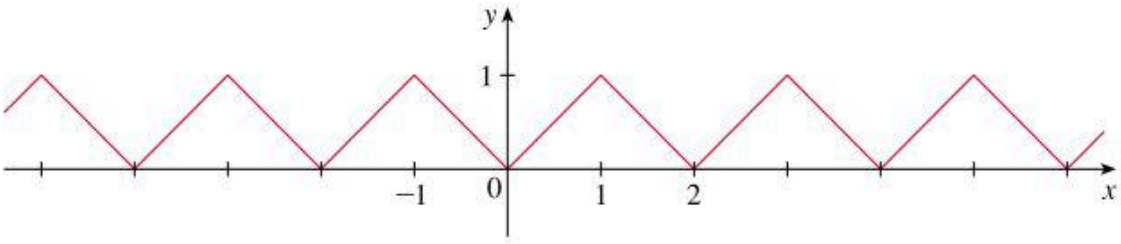
$$f(x) = \begin{cases} 0 & \text{if } -\pi \le x < 0 \\ 1 & \text{if } 0 \le x < \pi \end{cases} \quad \text{and} \quad f(x + 2\pi) = f(x)$$



Write MATLAB code for Fourier series implementation using following harmonics.

H1=[3 20 33 100]. Use subplot to plot all the harmonics result.

Q2: Find the Fourier series function of the triangular wave defined by f(x)=|x| for -1<x<1 and f(x+2)=f(x) for all x. The graph of x is shown below.



Write MATLAB code for Fourier series implementation using following harmonics.

H1=[5 25 33 50]. Use subplot to plot all the harmonics result.

Q3: A voltage Esinwt where t represents time, is passed through a so called half wave rectifier that clips negative part of the wave. Find the Fourier series of the resulting function using MATLAB.

$$f(t) = \begin{cases} 0 & \text{if } -\frac{\pi}{\omega} \le t < 0 \\ E \sin \omega t & \text{if } 0 \le t < \frac{\pi}{\omega} \end{cases} \qquad f(t + 2\pi/\omega) = f(t)$$

Where E=5, w=pi.

Use harmonic H=[20 40 60 100]. Use subplot to plot all the harmonics result.

ISRA University Islamabad Campus

Signals & Systems Lab

EXPERIMENT # 09*: Fourier Transforms*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……

Lab09||Signals and Systems

Fourier Transform

Objective

Main objective of the lab is to seek the concept of transformation of a time domain signal to frequency domain using Fourier transformation and its analysis. In the last, we will find the frequency response of a certain system over a range of frequencies.

Fourier Transform

Fourier Transformation transforms a composite time domain signal to frequency domain (complex) which shows the frequency components present in the system. The system can easily be analyzed through this transform. Fourier Transform falls in two types

1. Continuous time Fourier transform
2. Discrete Time Fourier transform

We will discuss them one by one

Continuous time Fourier transform

CTFT is defined by the following relation.

$$F(e^{jw}) = \int_{-\infty}^{\infty} f(t)\, e^{-jwt} dt$$

Inverse Fourier transform transforms the signal back into time domain and give mathematically as

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

In Matlab we implement CTFT using symbolic math toolbox. **Fourier** and **Fourier** are the two functions used for Fourier and inverse furrier transform.

Example:

Find the Fourier transform of the following signal.

$$x(t) = \delta(t)$$

CTFT by mtalba is computed as follows.

```
syms t
ft=dirac(t);
fw=fourier(ft);
pretty(fw)
```

Example

To find the CTFT of a unit step function we proceed as follows

$$x(t) = u(t)$$

```
u=sym('heaviside(t)');
fw=fourier(u);
pretty(fw)
```

Example

Find the CTFT of $x(t) = e^{-t^2/2}$.

```
syms f t
f=exp(-t^2/2);
ft=fourier(f);
pretty(ft);
f=ifourier(ft)
pretty(f)
```

Example

Find the Fourier Transform of the following signal.

$$x(t) = te^{-t^2}$$

```
syms w t
f=t*exp(-t^2);
fw=fourier(f);
pretty(fw);
```

Discrete Time Fourier Transform

Discrete Time Fourier transform is defined as below

$$X(e^{jwn}) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-jwn}$$

As compared to CTFT, DTFT deals with finite discrete time signals. Discrete time signals can be of infinite or finite number of samples. We deal with both of them one by one.

DTFT for infinite duration signals

Let us take the following example for discrete signal having infinite number of samples.

Example

Fourier transform of the discrete signal $x(n) = (0.5)^n \ u(n)$ can be calculated as

$$X(e^{jwn}) = \sum_{n=-\infty}^{\infty} (0.5)^n \ u(n) e^{-jwn}$$

$$X(e^{jwn}) = \sum_{n=0}^{\infty} (0.5)^n e^{-jwn}$$

$$X(e^{jwn}) = \sum_{n=0}^{\infty} (0.5e^{-jw})^n$$

$$X(e^{jwn}) = \frac{1}{1-0.5e^{-jw}}$$

$$X(e^{jwn}) = \frac{e^{jw}}{e^{jw}-0.5}$$

If x(n) is of infinite duration, then Matlab can not be used directly to compute $X(e^{jw})$. However we can evaluate the expression X(ejw) over $[0,\pi]$ frequencies and plot the magnitude and phase (or real and imaginary parts).

Following is the code and output for the Fourier transform evaluation of the above signal.

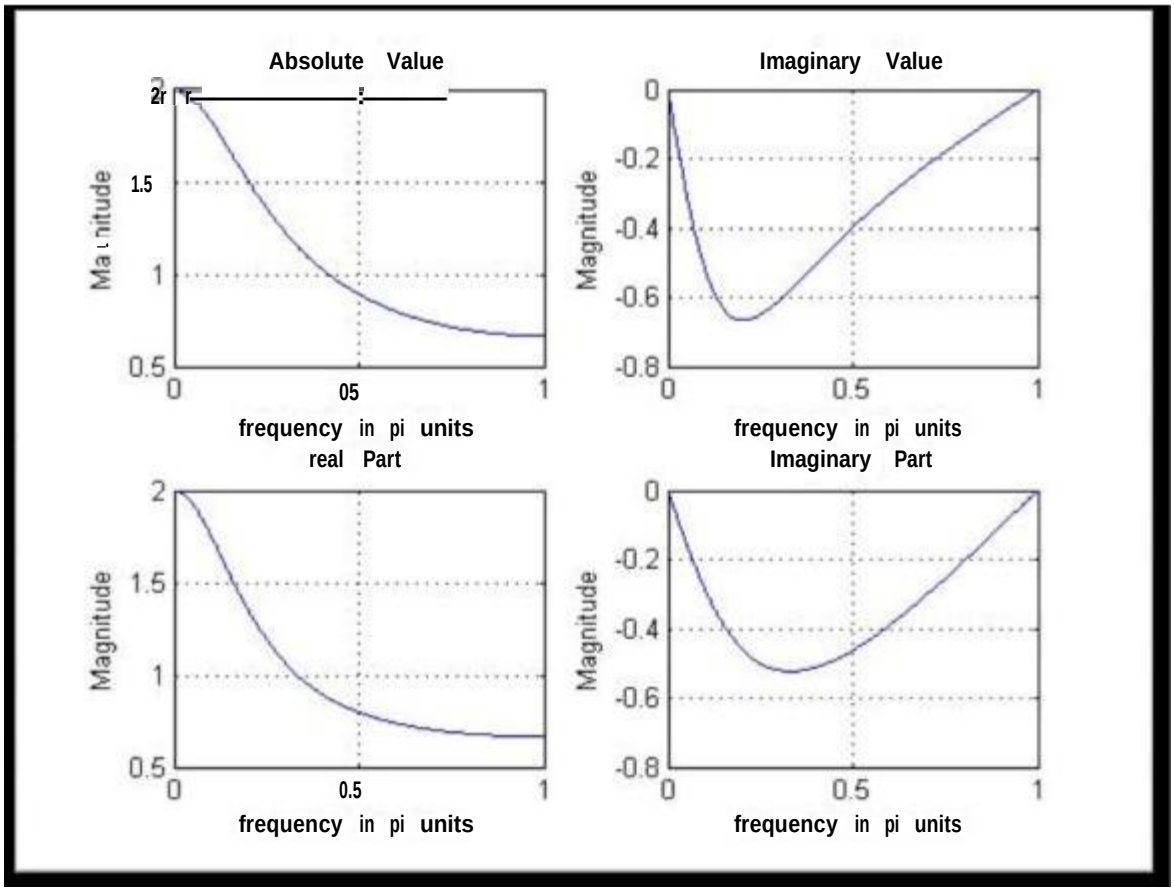
```
%Fourier Transform Evaluation

w=[0:1:500]*pi/500;  % Range of frequencies over 501 points

x=exp(j*w)./(exp(j*w)-0.5); %Fourier Transform

magx=abs(x);       %Magnitude

angx=angle(x);     %Phase

realx=real(x);     %Real Part

imagx=imag(x);     %Imaginary Part

subplot(221);

plot(w/pi,magx);grid;  %plotting in Pi units

xlabel('frequency in pi units')

ylabel('Magnitude')

title('Absolute Value');

subplot(222);plot(w/pi,imagx);grid;

xlabel('frequency in pi units')

ylabel('Magnitude')

title('Imaginary Value');

subplot(223);plot(w/pi,realx);grid;

xlabel('frequency in pi units')

ylabel('Magnitude')

title('real Part');

subplot(224);plot(w/pi,angx);grid;
```

xlabel('frequency in pi units')

ylabel('Magnitude')

title('Imaginary Part');

Output is shown below.



DTFT for finite duration signals

If x(n) is of finite duration then X(e^{jwn}) can be computed at any frequency w. If we evaluate

X (e^{jwn}) at equidistant frequencies between [0,π] then expression for X(e^{jwn}) can be implemented as Matrix multiplication operation.

Let us assume that the signal x(n) has N samples between $n_1 \leq n \leq n_N$ and that we want to evaluate the X(e^{jw}) at

$$\omega = \frac{\pi}{M} * K \quad \text{where } K = 0, 1, 2, \ldots\ldots,$$

Which are (M+1) equispaced frequencies between [0,π]. Expression for $X(e^{jw})$ can be written as

$$X(e^{jwk}) = \sum_{l=1}^{N} e^{-j(\frac{}{M})kn} x(n)$$

Where $X(e^{jwk})$ and x(n) are evaluated as column vectors X and x respectively, we have

$$\mathbf{X=Wx}$$

Where W is an (M+1)xN matrix given by

$$We = \quad ^{-\frac{j}{M}kn} \quad \text{Where } n_1 \leq n \quad _N \quad \text{and } K = 0,1,2,.....,$$

Furthermore n and k are taken as row vectors. Therefore taking transpose of above equation

$$w = [\exp(-j\frac{}{M}k^T n)]$$

In Matlab we take sequence and indices as row vectors so we can write the final expression for $X(e^{jwn})$ as

$$X = x * \exp(- \quad j * pi / M).\wedge (n' * k)$$

Where $n = n_1 n \quad _N$ and $k = 0: M;$

Example

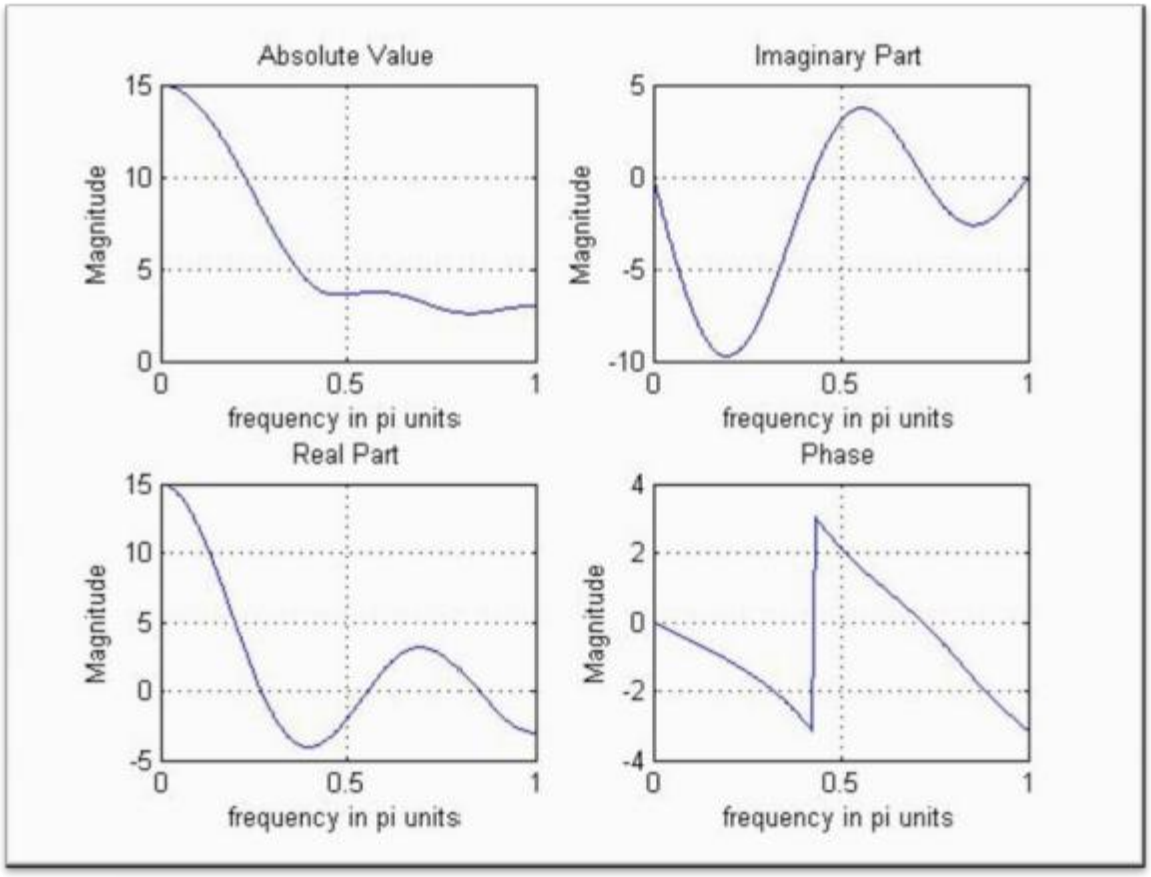
Find the DTFT for the following discrete time signal

$$x(n) = \quad \{1, 2, 3, 4, 5\}$$

Matlab code for the above signal is given below.

```
n=-1:3;

y=1:5;

k=0:500;
```

```
w=(pi/500)*k;
x=y*exp(-j*pi/500).^(n'*k);
magx=abs(x);
angx=angle(x);
realx=real(x);
imagx=imag(x);
subplot(221);plot(w/pi,magx);grid;
xlabel('frequency in pi units');
ylabel('Magnitude');
title('Absolute Value');
subplot(222);plot(w/pi,imagx);grid;
xlabel('frequency in pi units');
ylabel('Magnitude')
title('Imaginary Part');
subplot(223);plot(w/pi,realx);grid;
xlabel('frequency in pi units');
ylabel('Magnitude');
title('Real Part');
subplot(224);plot(w/pi,angx);grid;
xlabel('frequency in pi units');
ylabel('Magnitude');
title('Phase');
```



Properties of Fourier Transform

Fourier Transform has the following two properties.

1. **Periodicity**

   DTFT is periodic in w with period of 2π..

$$X(e^{jw}) = X(e^{j[w+2]})$$

We need only one period of $X(e^{jw})$ vi.e $[0,2\pi]$ or $[-\pi,\pi]$ for analysis and the whole domain.

2. **Symmetry**

   For real valued x(n), $X(e^{jw})$ is conjugate symmetric.

$$X(e^{-jw}) = X^*(e^{jw})$$

Or

$$\text{Re}[X(e^{-jw})] = \text{Re}[X(e^{jw})]$$
$$\text{Im}[X(e^{-jw})] = \text{Im}[X(e^{jw})]$$
$$|X(e^{-jw})| = |X(e^{jw})|$$
$$\angle X(e^{-jw}) = -\angle X(e^{jw})$$

To plot $X(e^{jw})$, we need to consider only a half period of $X(e^{jw})$. Generally period is chosen to be $[0,\pi]$

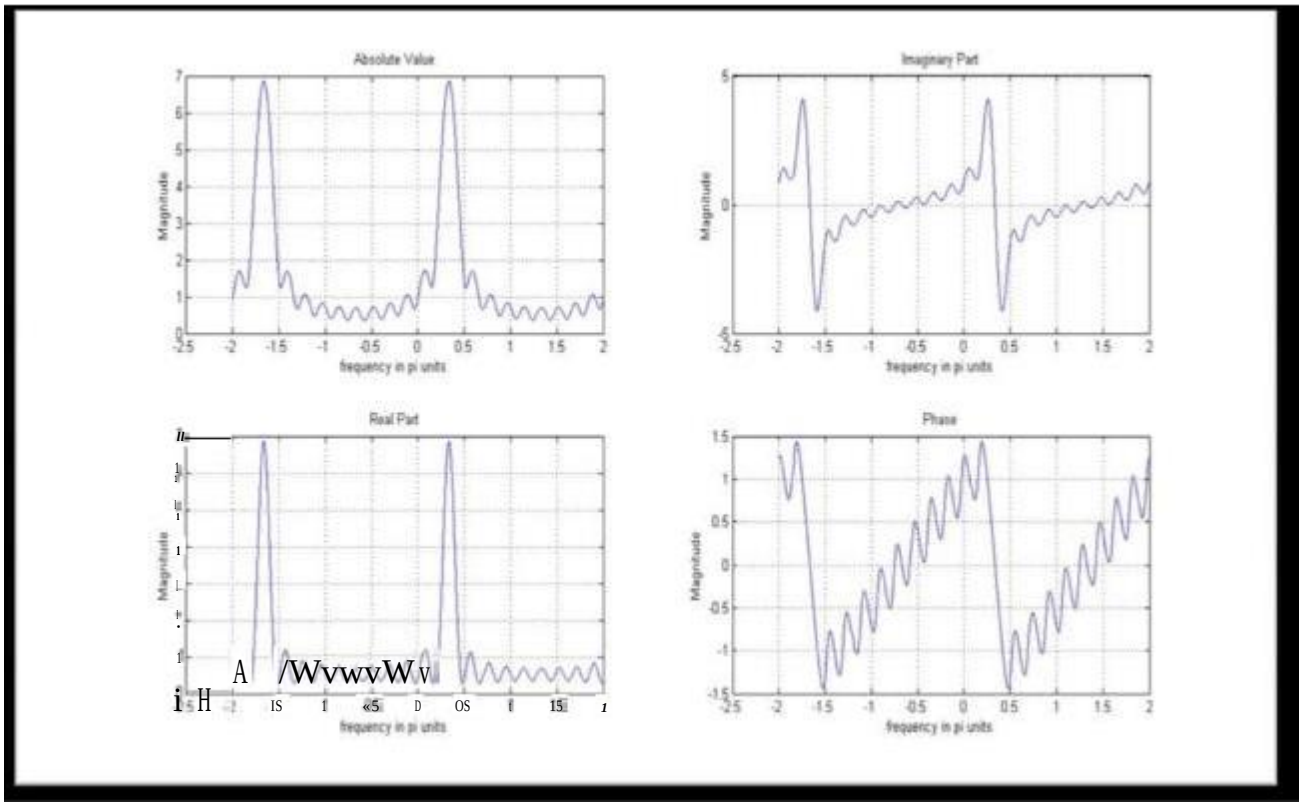
Example

Find the DTFT of the following signal and check for the periodicity of the transform. Perform computation over $[-2\pi:2\pi]$ frequency and 400 points.

$$x(n) = (0.9e^{jn/3}) \quad 0 \le n \le 10$$

```
n=0:10;

x=(0.9*exp(j*pi/3)).^n;

k=-200:200;

w=(pi/100)*k;

X=x*(exp(-j*pi/100).^(n'*k));

magx=abs(X);angx=angle(X);

realx=real(X);imagx=imag(X);

subplot(221);plot(w/pi,magx);grid;

xlabel('frequency in pi units');

ylabel('Magnitude');

title('Absolute Value');

subplot(222);plot(w/pi,imagx);grid;

xlabel('frequency in pi units');

ylabel('Magnitude')

title('Imaginary Part');

subplot(223);plot(w/pi,realx);grid;
```

```
xlabel('frequency in pi units');

ylabel('Magnitude');

title('Real Part');

subplot(224);plot(w/pi,angx);grid;

xlabel('frequency in pi units');

ylabel('Magnitude');

title('Phase');
```



Output shows that X(e^{jwn}) is periodic.

Frequency Response

In this section, we will find the frequency response of a system over a certain range of frequencies.

Freqz is the Matlab function for this purpose.

Example

Suppose we have the following system

$$y(n) = \frac{x(n) + x(n-1)}{2}$$

Where x(n) is the input of the system. Impulse response of the system is given by

$$h(n) = \frac{\delta(n) + \delta(n-1)}{2}$$

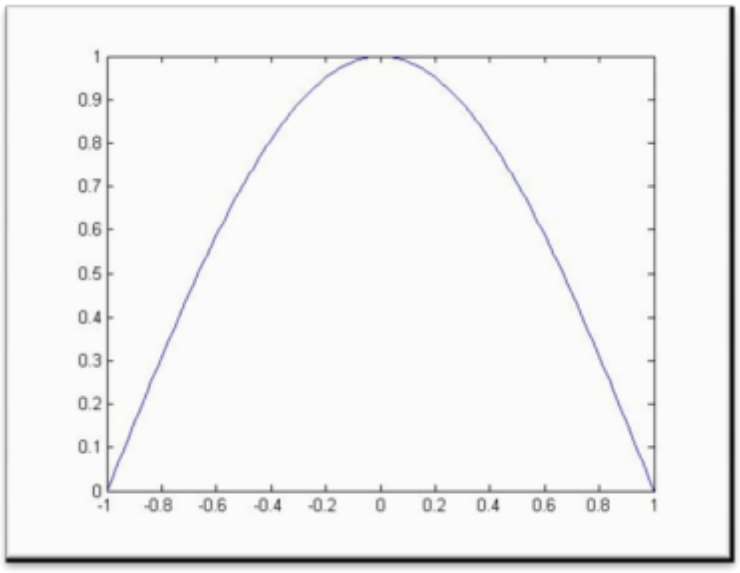
Taking Fourier transform gives the frequency response of the system which is given by

$$H(jw) = \frac{1 + e^{-jw}}{2}$$

Matlab implementation of the above system is as below

```
%Frequency response
num=[1 1];
den=[2];
w=-pi:pi/100:pi; %Range of frequencies
h=freqz(num,den,w);
plot(w/pi,abs(h));
```

Output is as shown below



Output shows that the system is low pass filter.

Exercise

E.1: For each of the following sequences determine the DTFT $X(e^{jw})$. Plot the magnitude and angle of $X(e^{jw})$.

1. X[n]= {4,3,2,1,0,-1,-2,-3,-4} Comment on the angle plot.

2. $X(n)=(n+2)(-0.7)^{n-1}\, u(n-2)$

E.2: A symmetric rectangular pulse is given by

$$R_N(n)= \begin{cases} 1 \text{ for } -N \leq n \leq N \\ 0 \text{ otherwise} \end{cases}$$

Determine the DTFT for N=5,15,25,100. Scale the DTFT so that $X(e^{j0})=1$. Plot the normalized DTFT over [- ,]. Study these plots and comment on their behaviour as a function of N.

E.3: Find and plot the frequency response of the given functions using freqz.

1. $f(n) = \frac{x(n) - x(n-1)}{2}$

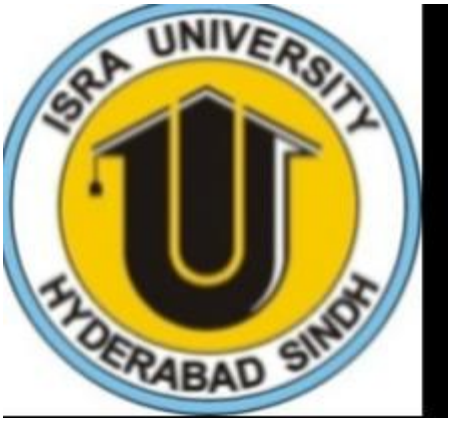
2. $f(n) = \frac{x(n) + x(n-1) + x(n-2)}{3}$

E.4 : Find the CTFT of the following signals using symbolic Maths toolbox.

1. $2tu(t) - t$
2. $e^{tt}u(t) - e^{-2}u(t)$

ISRA University Islamabad Campus

Signals & Systems Lab



EXPERIMENT # 10: *Introduction to Simulink and Its implementation*

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……

Lab10 || Introduction to Simulink and Implementation

Objective

Objective of the lab is to introduce the Simulink tool for the Matlab and develop understanding of implementing the system model and analysis using this tool.

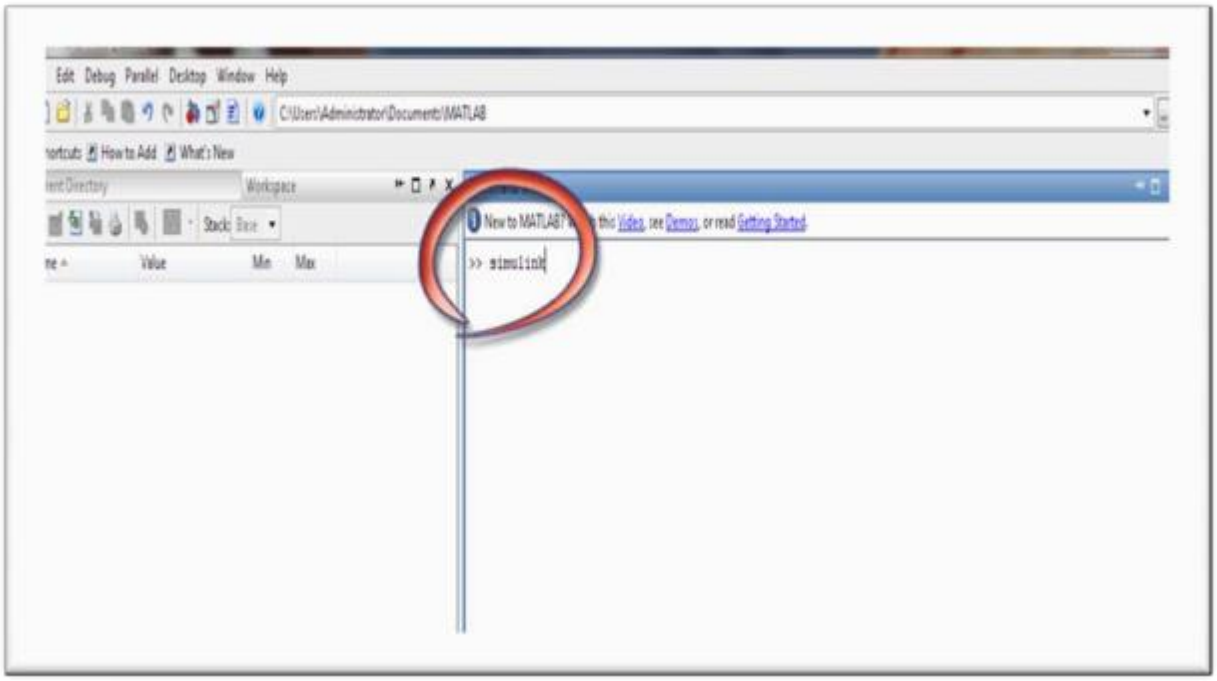
Simulink

Simulink is a graphical environment used for the modeling, simulation and analysis of a dynamic system. Main interface of the tool is a block diagram based.

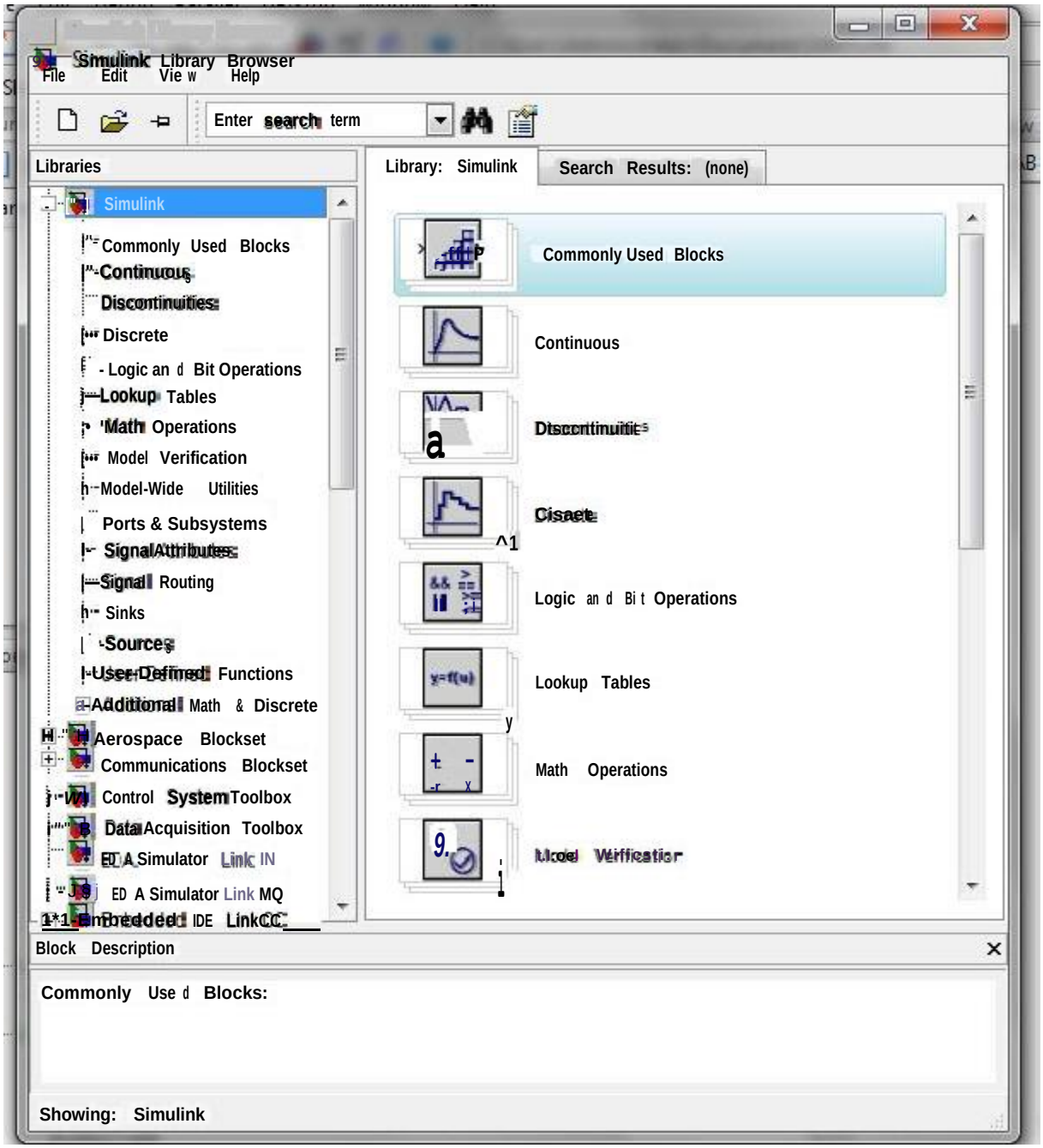
It provides an interactive graphical environment and a customizable set of block libraries that let you accurately design, simulate, implement, and test control, signal processing, communications, and other time-varying systems.

In Matlab we can access the Simulink tool through three different ways.

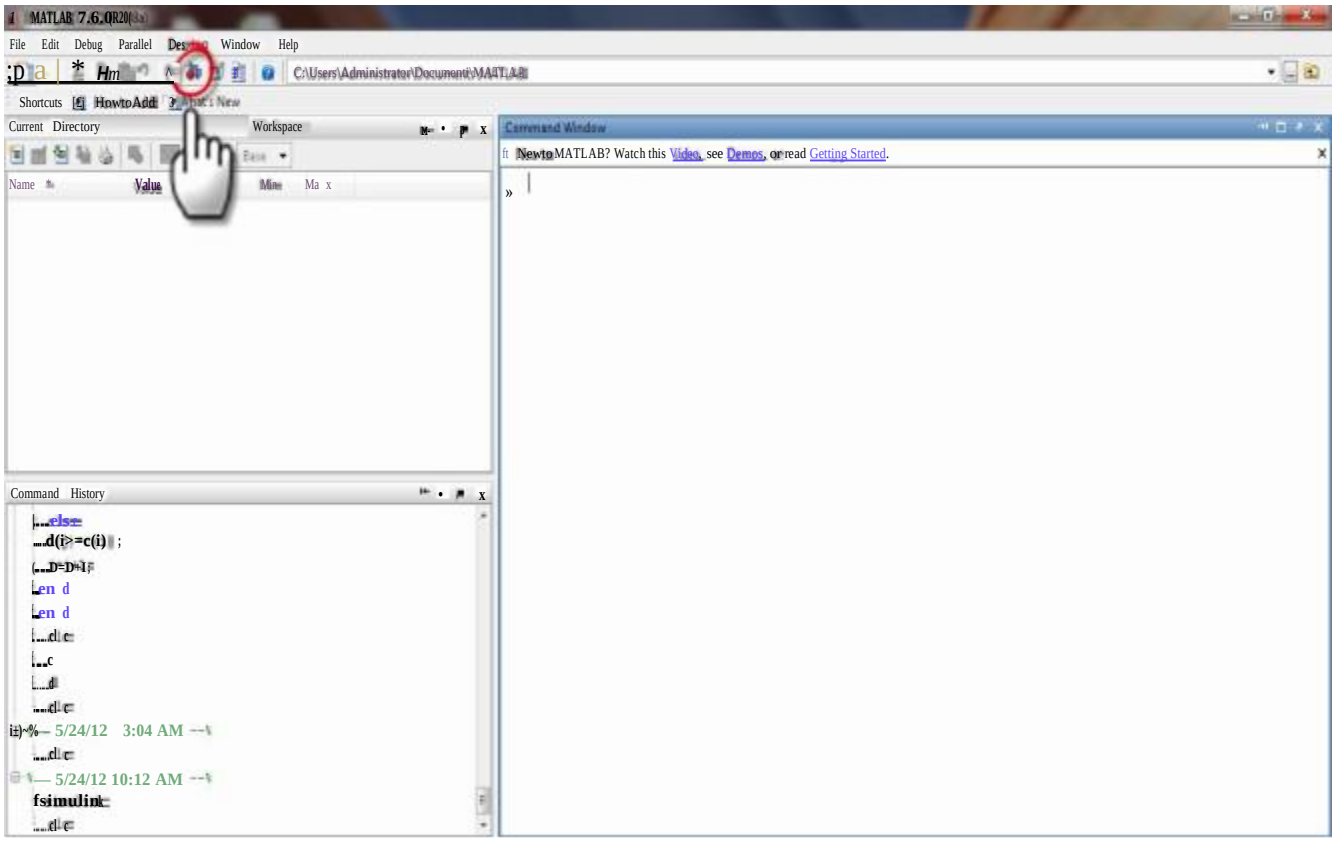
1. By typing **Simulink** in command window.



We get the following Simulink Library browser window open by this command



2. In command prompt we can access the Simulink by following the menu **File>New>Model**
3. Clicking the Simulink Red icon on the main drop down menu.

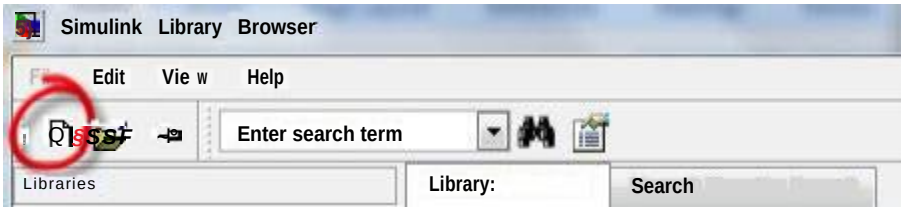


Simulink library Browser

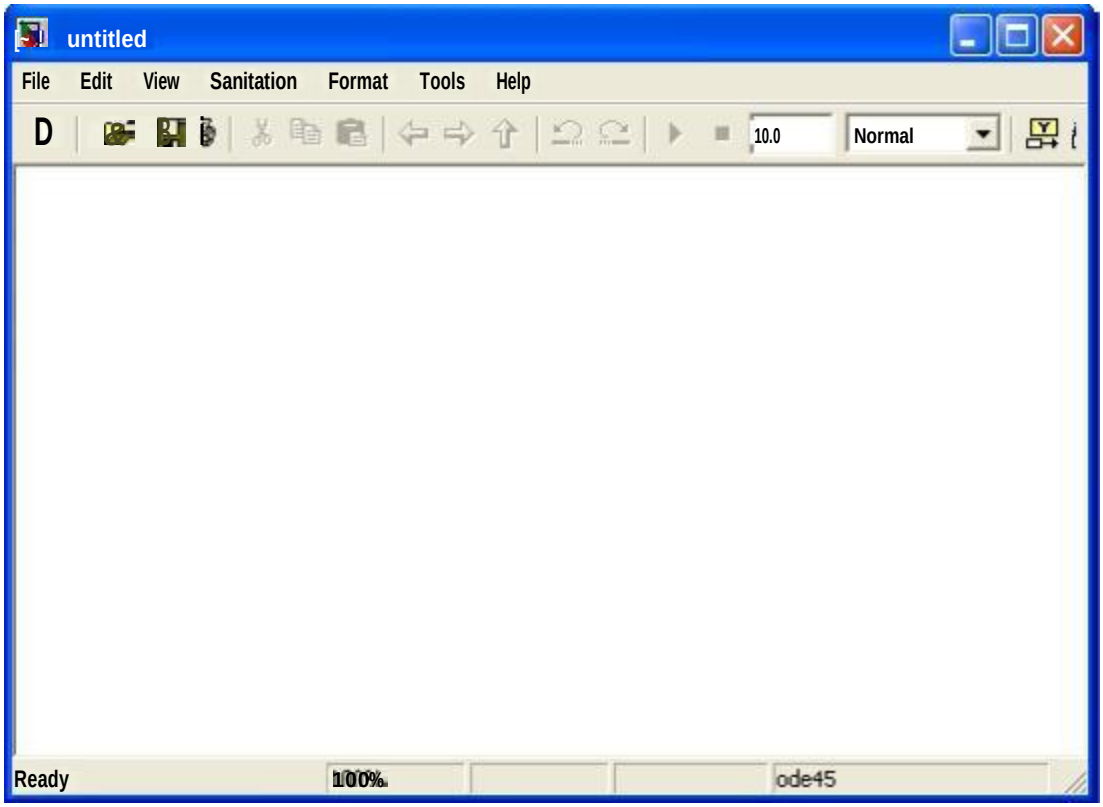
The Simulink Library Browser is the library where you find all the blocks you may use in Simulink. Simulink software includes an extensive library of functions commonly used in modeling a system. These include:

1. Continuous and discrete dynamics blocks, such as Integration, Transfer functions, Transport Delay, etc.
2. Math blocks, such as Sum, Product, Add, etc
3. Sources, such as Ramp, Random Generator, Step, etc

Create a new Model. Click the New icon on the Toolbar in order to create a new Simulink model.



We get the following window open



This is called Model window where we can design our model.

Example:

Suppose we want to implement the system which generates sine wave using Simulink.

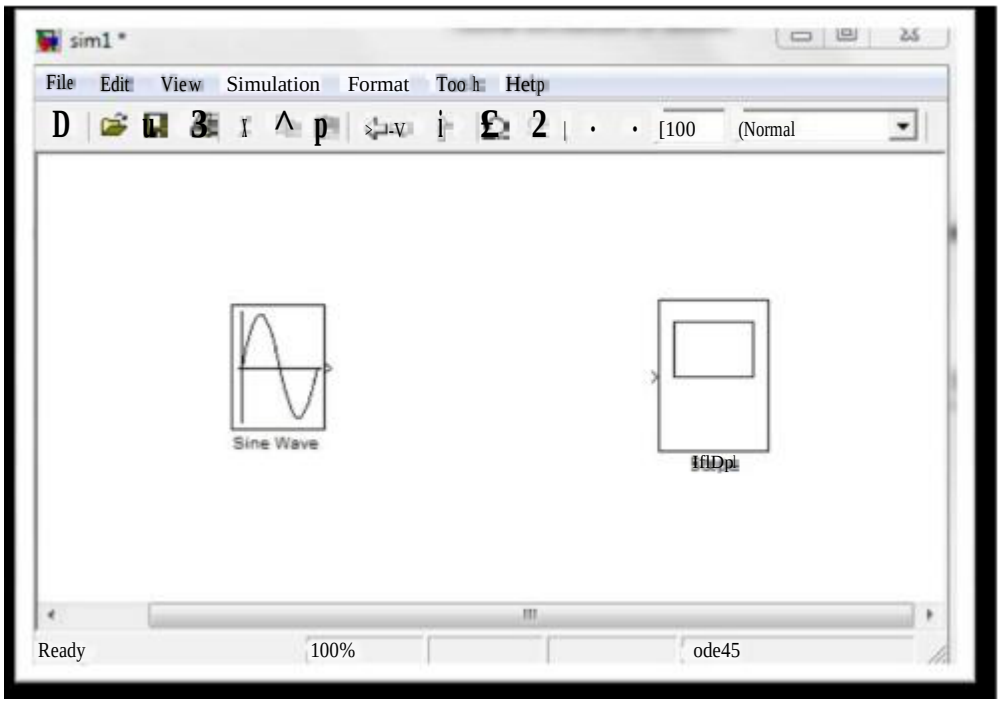
To implement the system we need two blocks.

1. Source which generates sine wave
2. Device to see output

   In Simulink sine wave falls under category of **Sources**. Output is seen at oscilloscope in **Sink** category.

   Here is how we implement the system

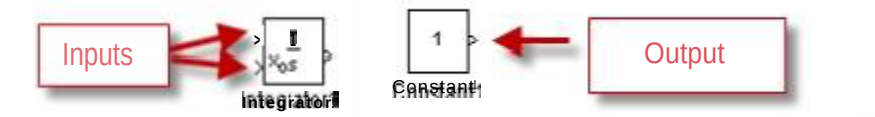
   1. Open library browser. From **Sources** category select **sine wave** block and drag it to the model window
   2. From **Sink** category, select **scope** (oscilloscope) and drag it to the model window. You get the following window after getting the blocks



Next step is the wiring of the blocks.

Wiring Techniques

Use the mouse to wire the **inputs** and **outputs** of the different blocks. Inputs are located on the left side of the blocks, while outputs are located on the right side of the blocks.



When holding the mouse over an input or an output the mouse changes to the following symbol.

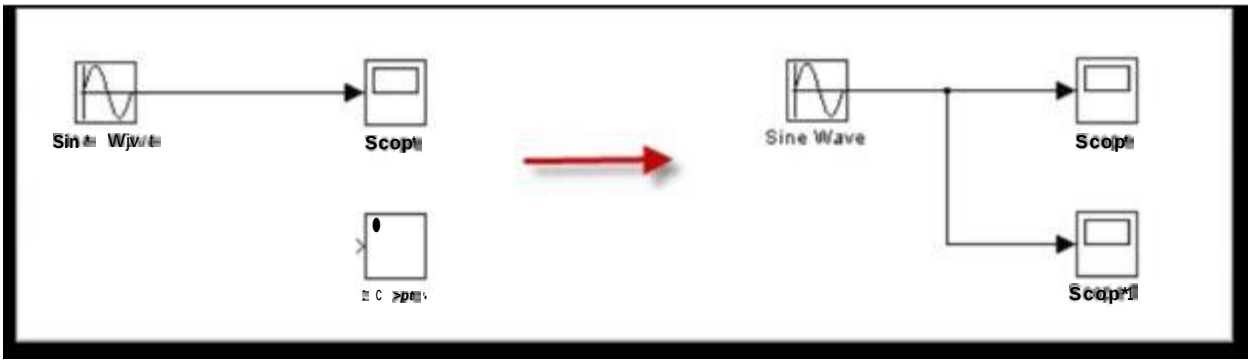
Use the mouse, while holding the left button down, to drag wires from the input to the output.

Automatic Block Connection:

Another wiring technique is to select the source block, then hold down the Ctrl key while left-clicking on the destination block. Try the different techniques on the example above.

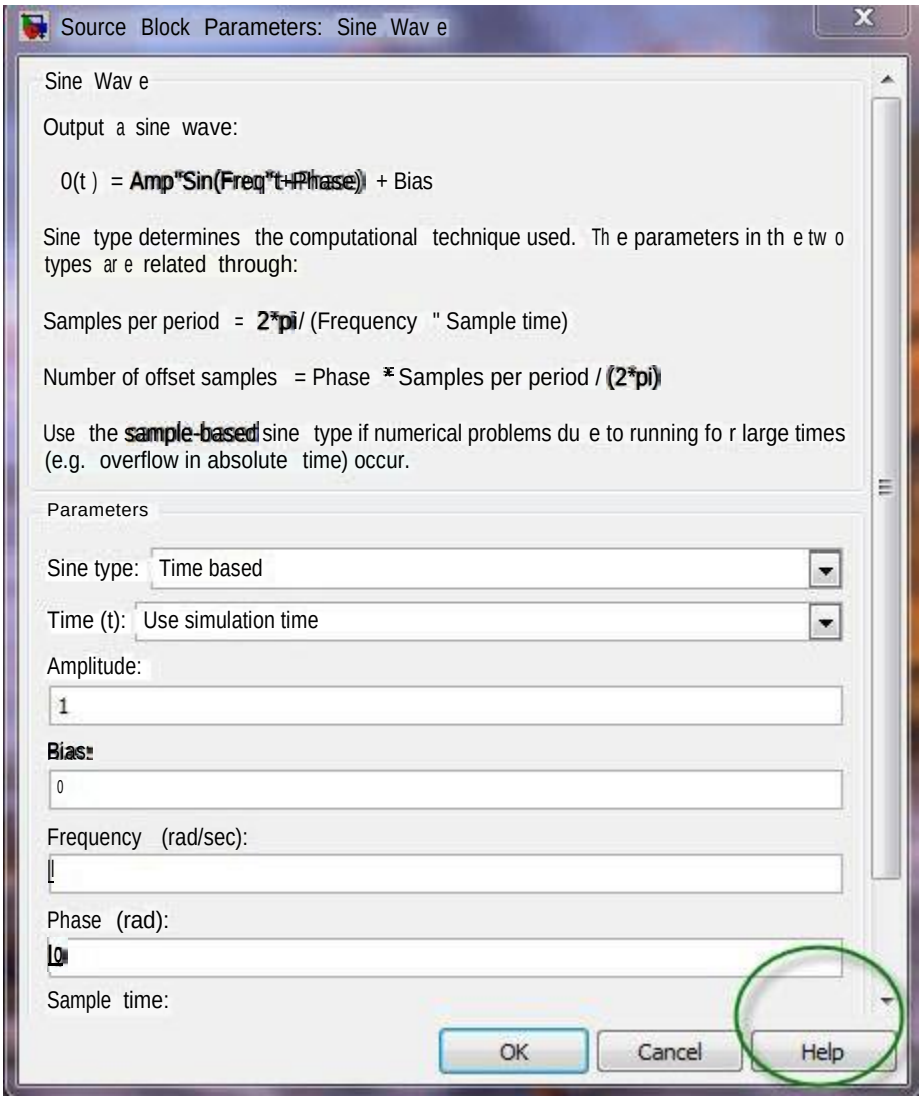
Connection from a wire to another block

If wire a connection from a wire to another block, like the example below, you need to hold down the Ctrl key while left-clicking on the wire and then to the input of the desired block.



Help window

In order to see detailed information about the different blocks, use the built-in Help system

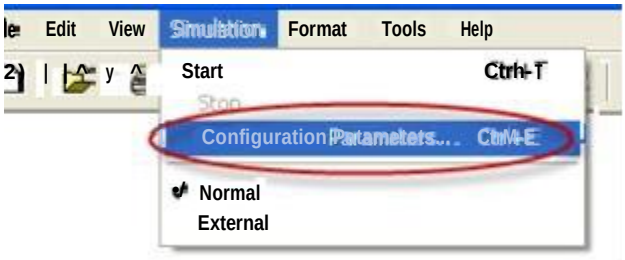


All standard blocks in Simulink have detailed Help. Click the Help button in the Block Parameter window for the specific block in order to get detailed help for that block.

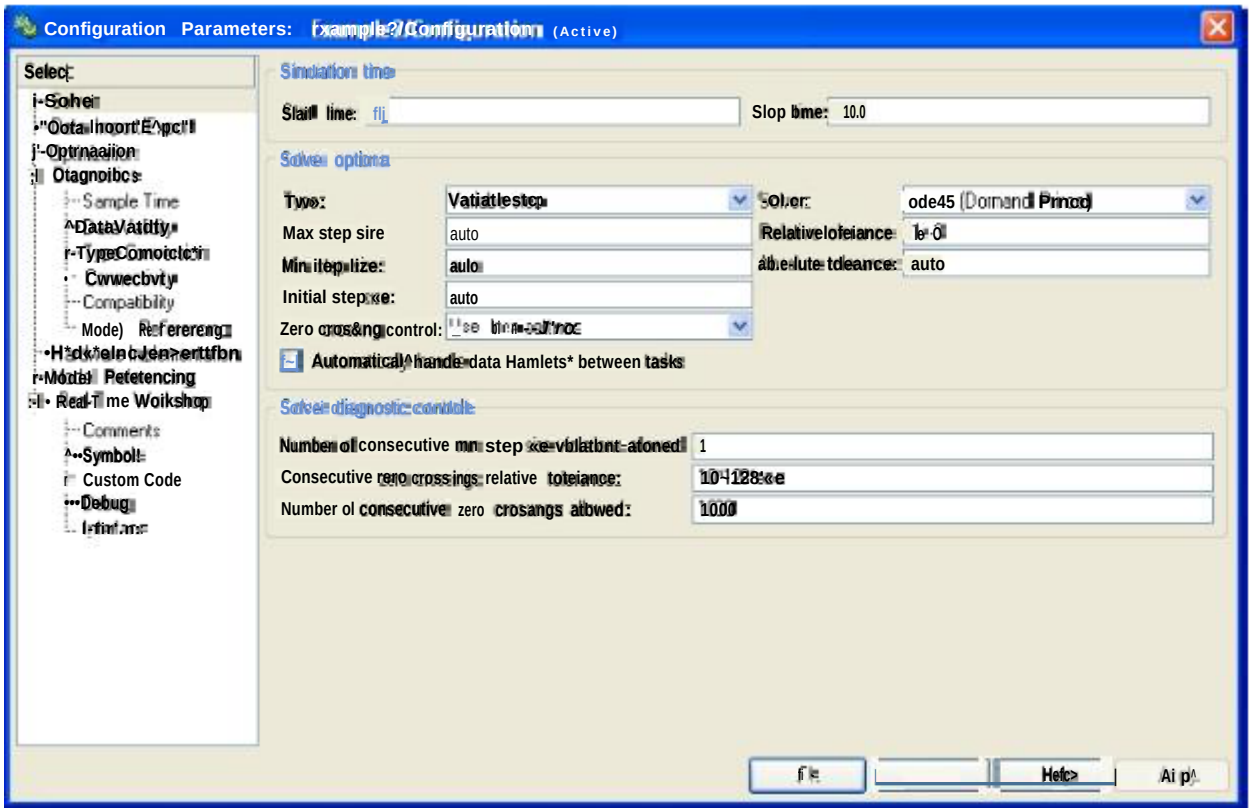
The Help Window then appears with detailed information about the selected block:

Configuration Parameters

There are lots of parameters you may want to configure regarding your simulation. Select "Configuration Parameters…" in the Simulation menu



The following window appears:



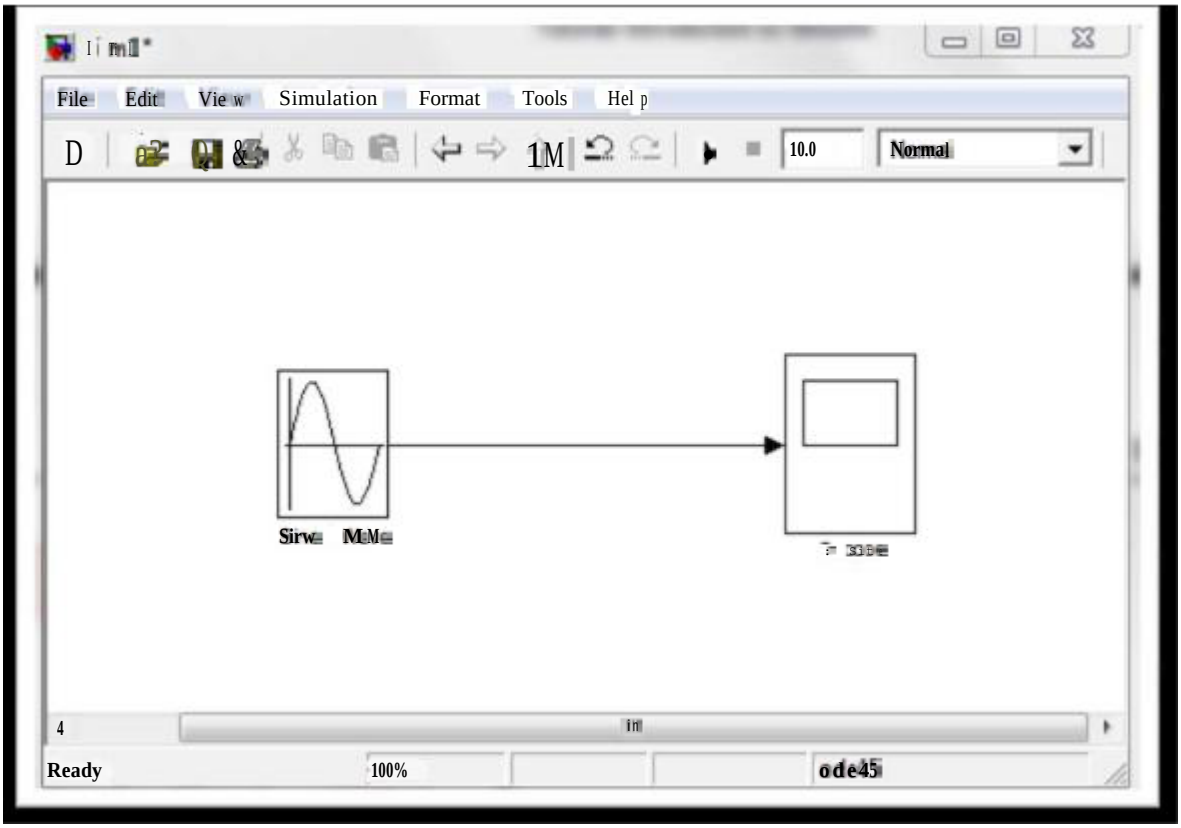
Here you set important parameters such as:

- Start and Stop time for the simulation
- What kind of Solver to be used (ode45, ode23 etc.)
- Fixed-step/Variable-step

Note: Each of the controls on the Configuration Parameters dialog box corresponds to a configuration parameter that you can set via the "**sim**" and "**simset**" commands.

Solvers are numerical integration algorithms that compute the system dynamics over time using information contained in the model. Simulink provides solvers to support the simulation of a broad range of systems, including continuous-time (analog), discrete-time (digital), hybrid (mixed-signal), and multirate systems of any size.

After setting all the steps, system is connected as follows.

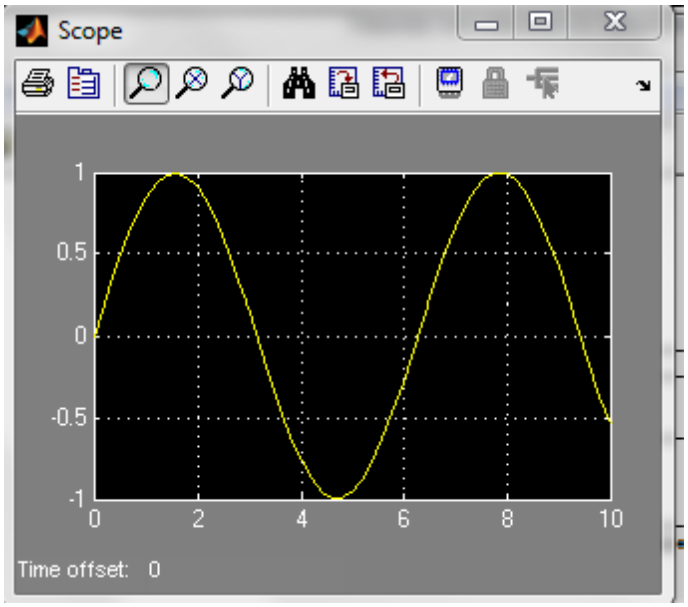


Save the model file.

For running the file press the **Run** button on the menu or **Simulation>Start** or **ctrl+I.**

After this, double ccik on the scope to see the output.

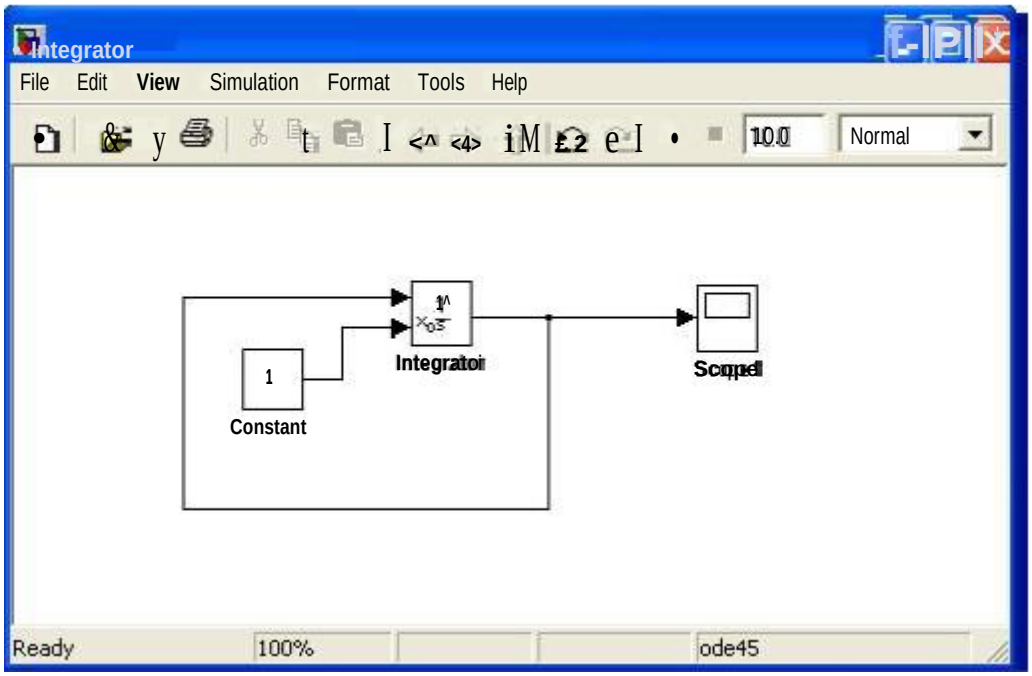
It appears as follows.



You can right click to select the option **autoscale** to view the full output.

Example: **Integrator with initial value**

Create the following model and run the simulation

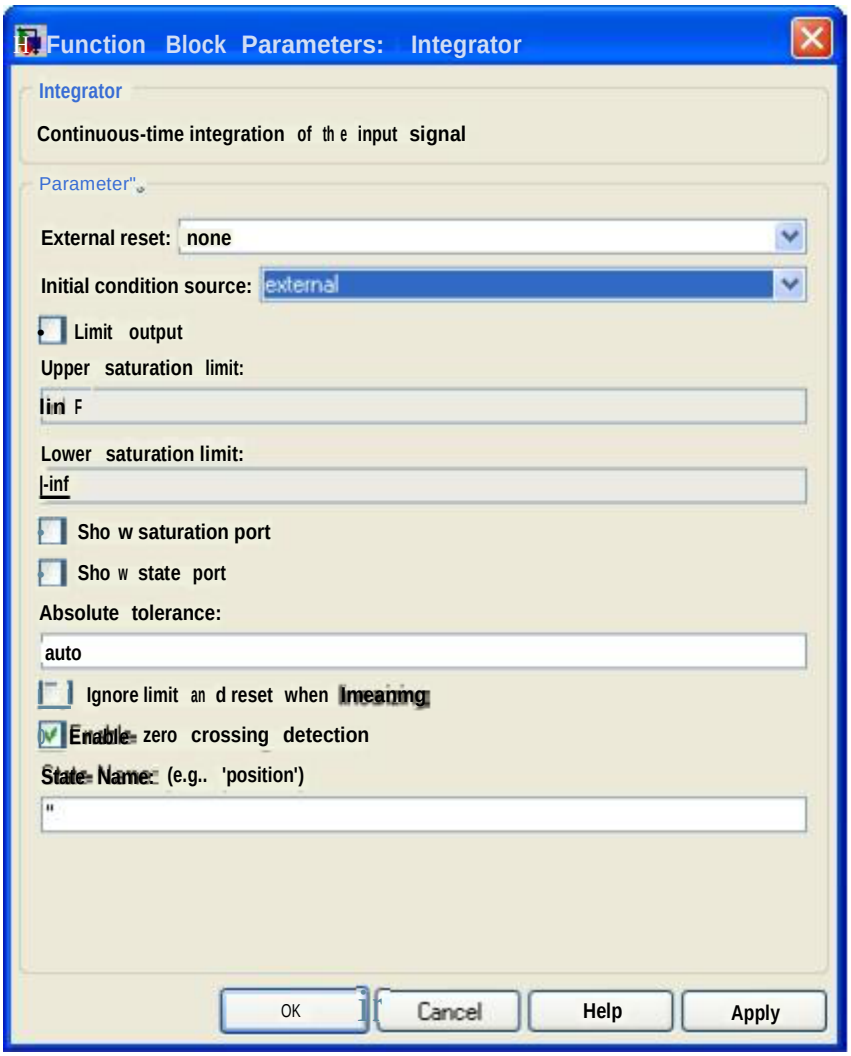


Step1: Place the blocks on the model surface

This example use the following blocks:

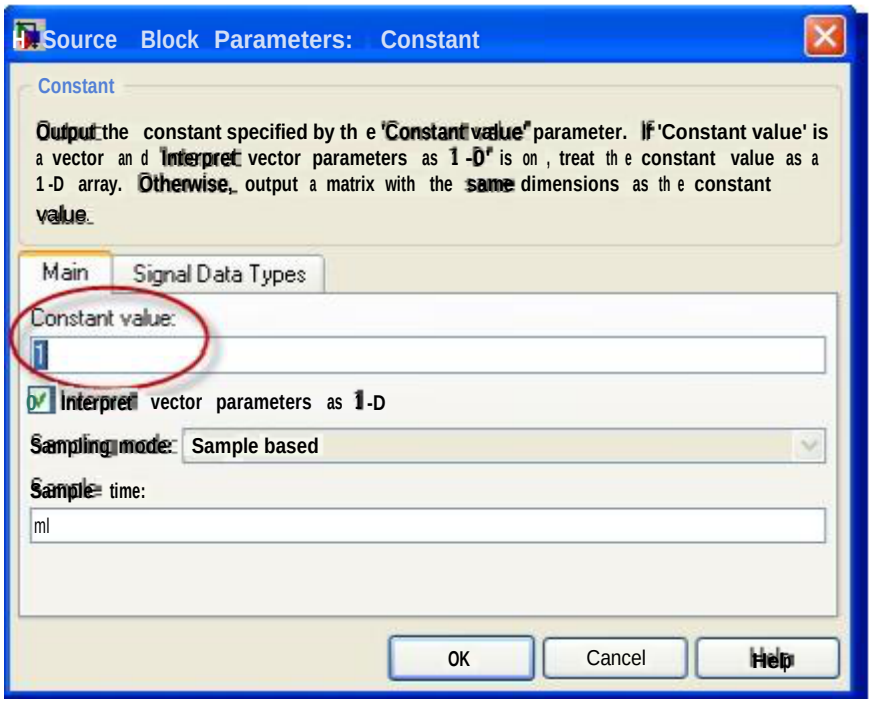
Step 2: Configuration

Double-click on the Integrator block. The Parameter window for the Integrator block appears:



Select "Initial condition source=external". The Integrator block now looks like this:

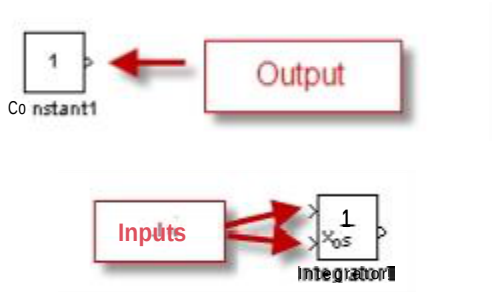
Double-click on the Constant block. The Parameter window for the Constant block appears:



In the Constant value field we type in the initial value for the integrator, e.g., type the value 1.

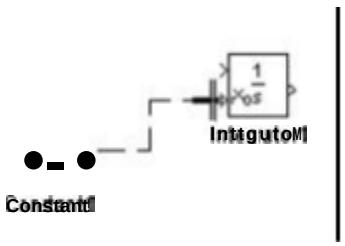
Step 3: **Wiring**

Use the mouse to wire the inputs and outputs of the different blocks.



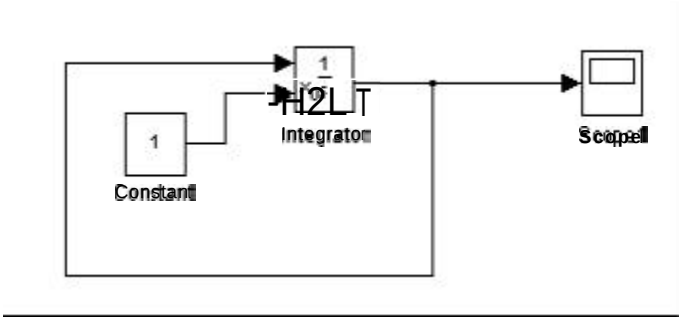
When holding the mouse over an input or an output the mouse change to the following symbol.

Draw a wire between the output on the Constant block to the lower input in the Integrator block, like this:



You could also do like this:

Wire the rest of the blocks together and you will get the following diagram:

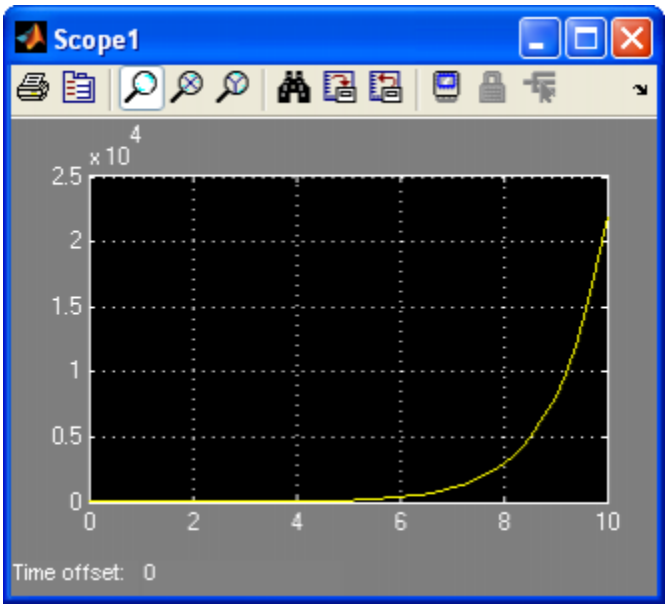


Step 4: Simulation

Start the simulation by clicking the "Start Simulation" icon in the Toolbar:

Step 5: The Results

Double-click in the Scope block in order to see the simulated result:



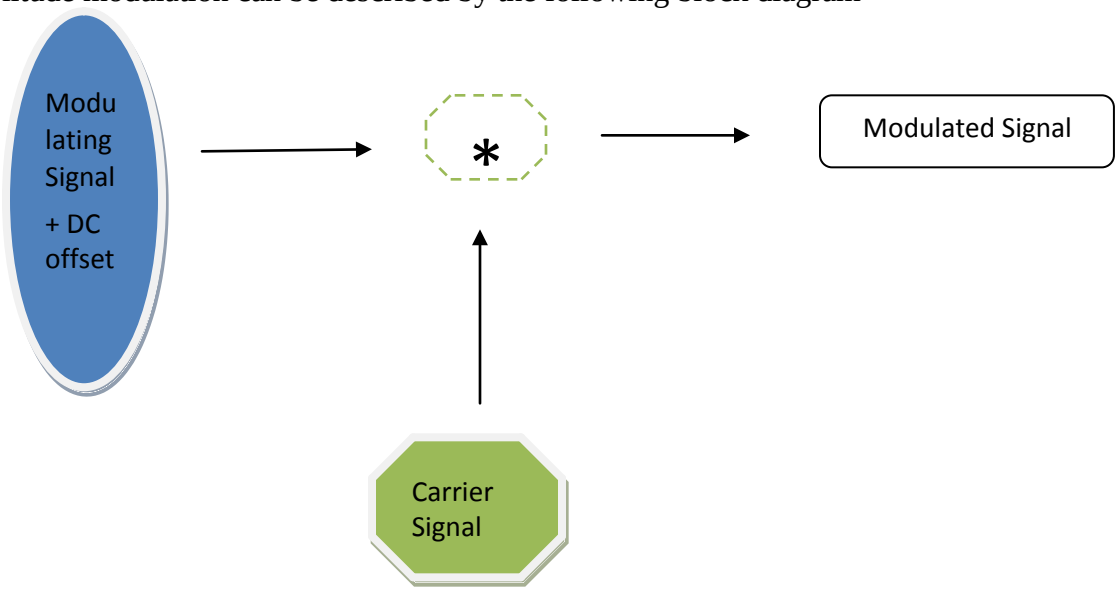
Example:

Implement the Amplitude Modulation using Simulink

Amplitude Modulation:

Amplitude modulation is the modulation of a wave by varying its amplitude, used chiefly as a means of radio broadcasting.

Amplitude modulation can be described by the following block diagram

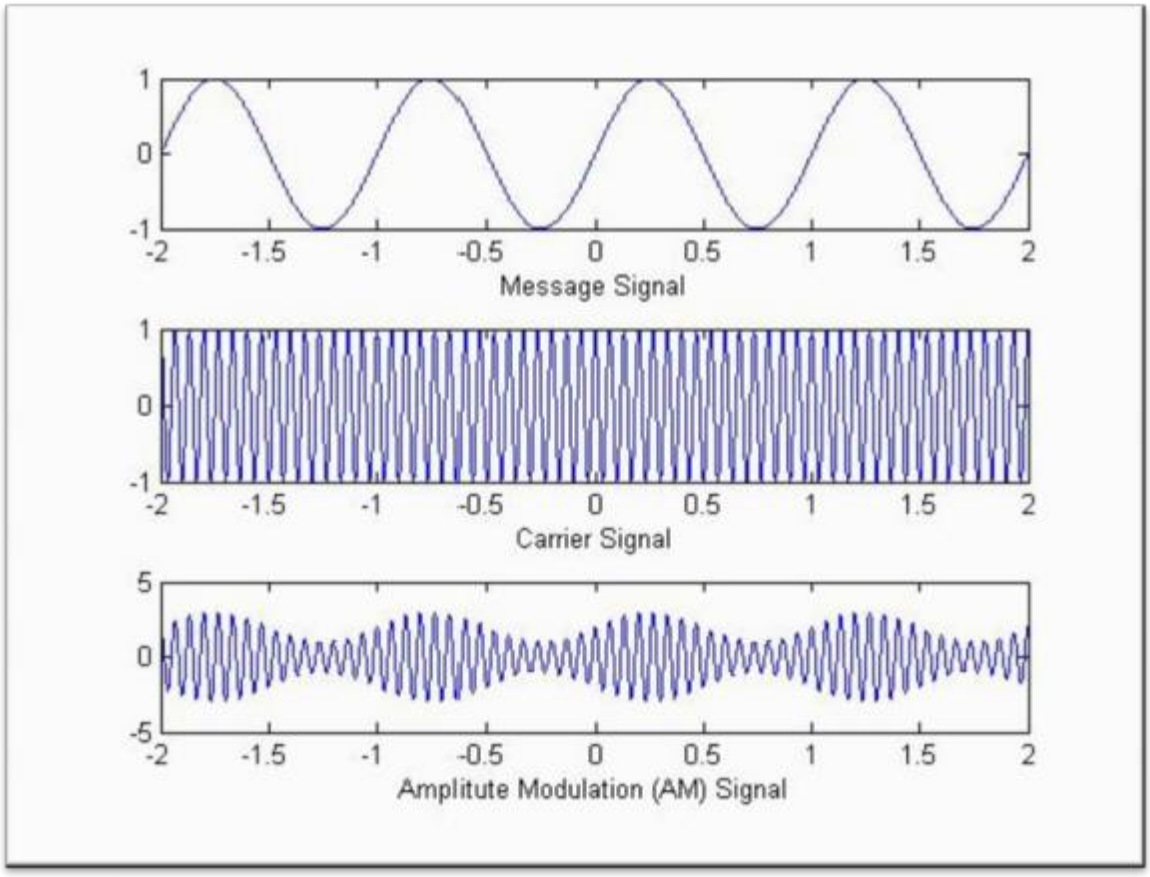


We implement the AM technique using MATLAB and Simulink.

```
%Matlab code for implementing Amplitude Modulation

ts=1e-3;

a=2;    % DC offset

t=-2:ts:2;

fm=1;           %Frequency of Message signal

fc=15;          % Frequency of Carrier Signal

wm=2*pi*fm;

wc=2*pi*fc;

message=sin(wm*t);              % Message Signal
```

```
carrier=cos(wc*t);              % Carrier Signal
am=(2+message).*carrier;        % Modulated signal
subplot(311);
plot(t,message);
xlabel('Message Signal');
subplot(312);
plot(t,carrier);
xlabel('Carrier Signal');
subplot(313);
plot(t,am);
xlabel('Amplitute Modulation (AM) Signal');
```

Output is shown below.

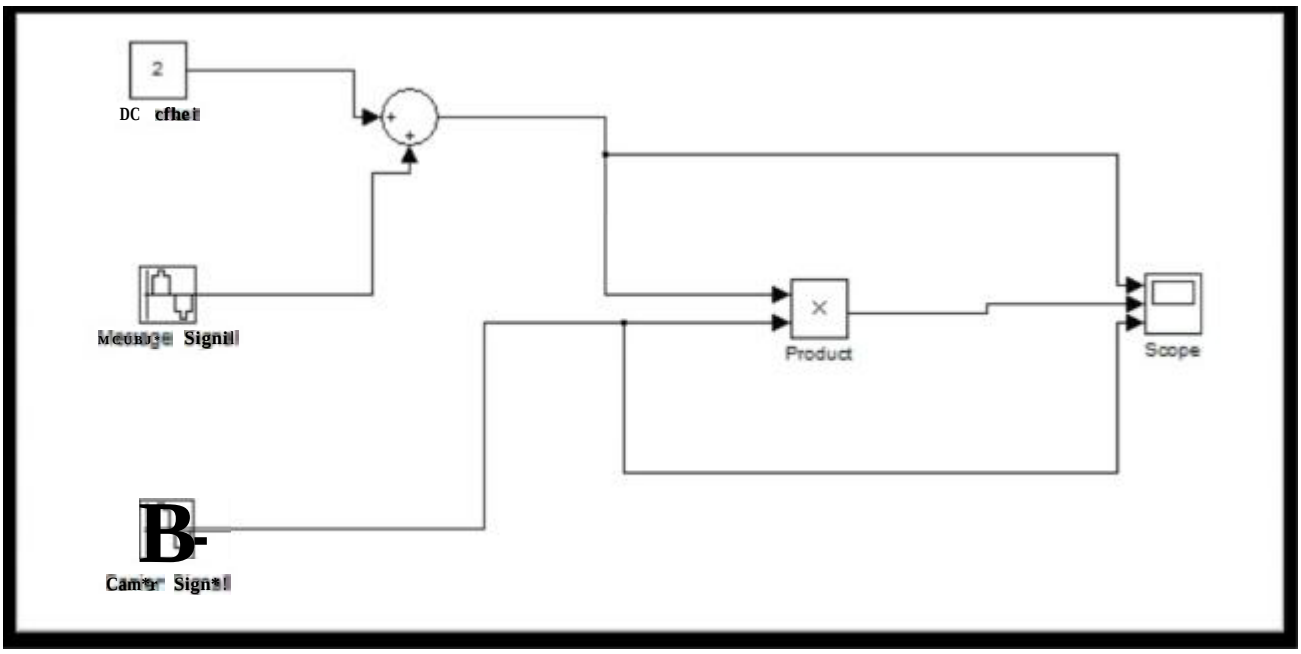


Implementation of Amplitude Modulation using Simulink is shown below.

We need the following blocks for the AM modulation

1. Constant block for DC offset
2. Sine wave for message signal
3. Sine wave for carrier signal
4. Sum for addition of the dc offset and message signal
5. Product for the multiplication of message signal and carrier signal
6. Scope for viewing the output.

Here is the Simulink block diagram for the AM model.



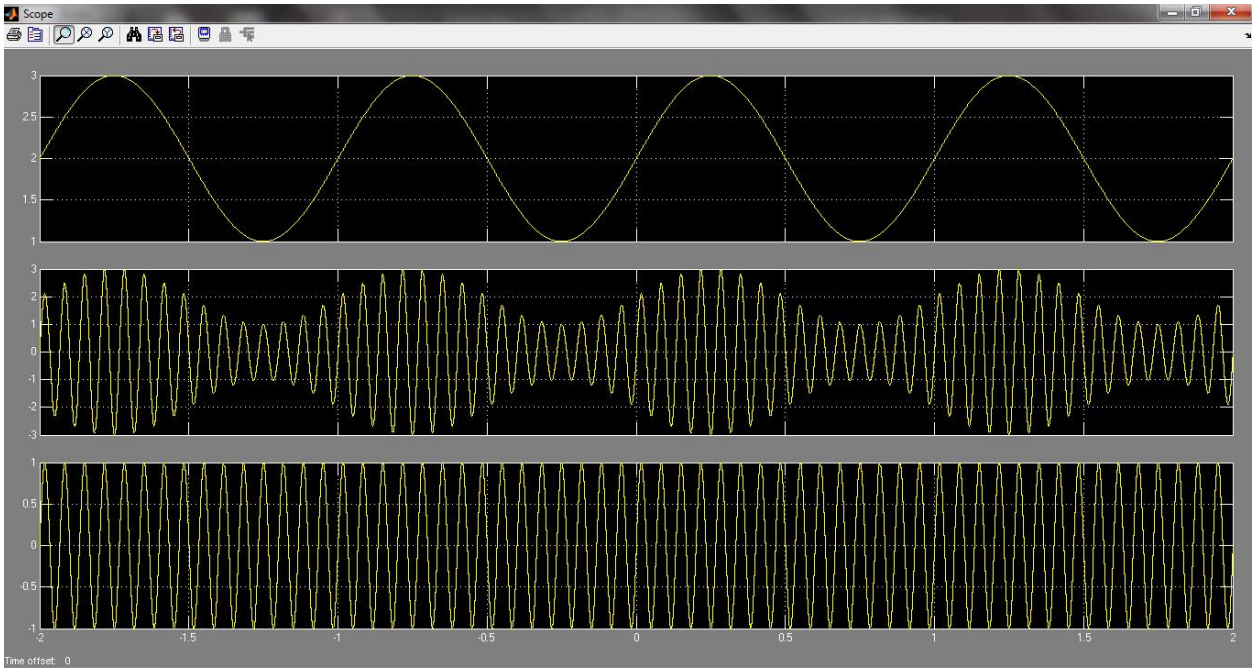
Followings are the parameter settings for each block.

Constant: Set the constant value to 2.

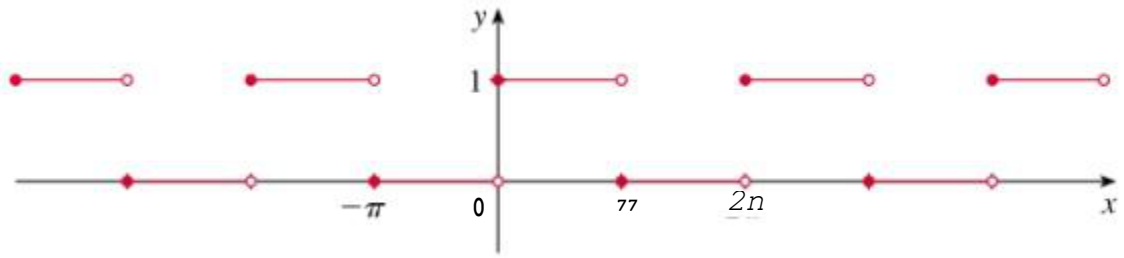
Message Signal: amplitude=1, frequency=2*pi*1, bias=0, phase=0, sample time=0.01

Carrier Signal: amplitude=1, frequency=2*pi*15, bias=0, phase=pi/2,sample time=0.01

Save and simulate the system. You will get the following output.



Example: Implement the Fourier series of the following signal using Simulink.



Coefficients for the Fourier series are given below.

$$a_0 = \frac{1}{2}$$

$$an = \frac{2}{n}(1 - con(n\ ))$$

$$bn =$$

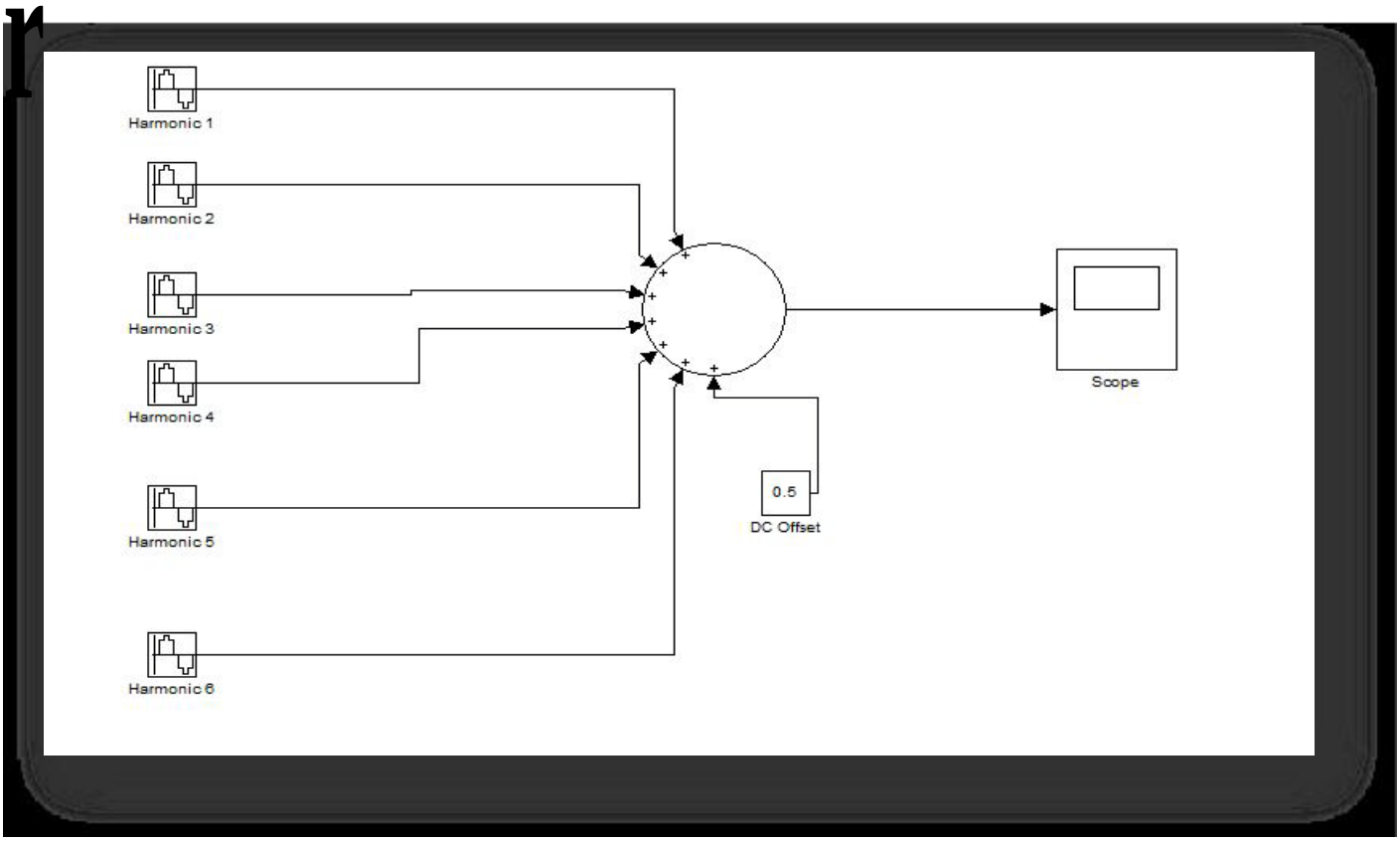
We can write the Fourier series expansion as below

$$f(x) = \frac{12}{2} + \frac{}{\pi}\cos wt \quad \frac{2}{3\pi}\cos 3wt \quad \frac{2}{}\cos 5wt \quad \frac{2}{7}\cos 7wt \quad \ldots \quad +$$

By this, we need the following blocks

Sine wave, Constant, Scope

Construct the model as below.



Set the following parameters for each block.

Sine wave

Amplitude=2/pi,-2/3pi,2/5pi.-2/7pi,2/9pi

Frequency=1,3,5,7,9

Bias=0

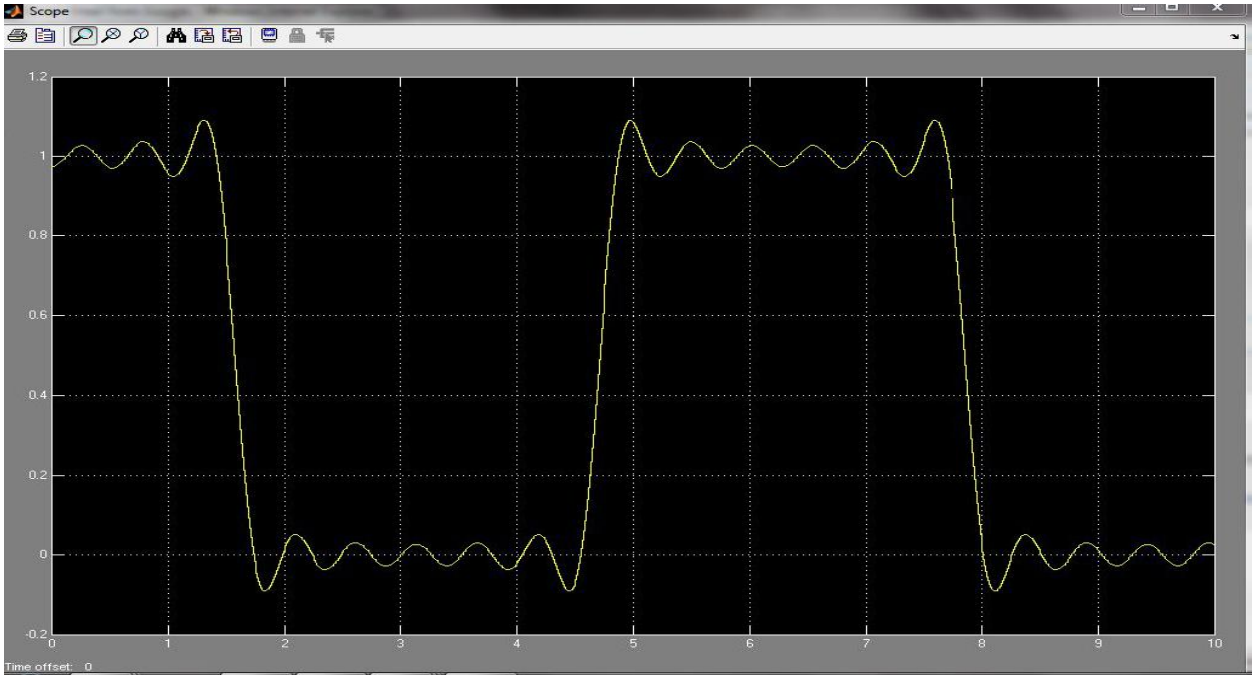
Phase=pi/2

Sample time=0.01

Constant

Constant=0.5

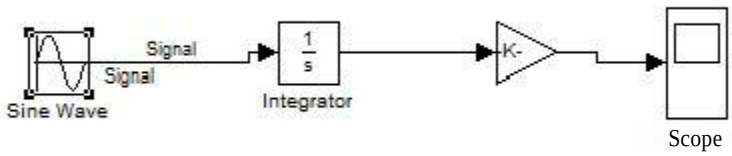
Save and simulate. You will get the following output.



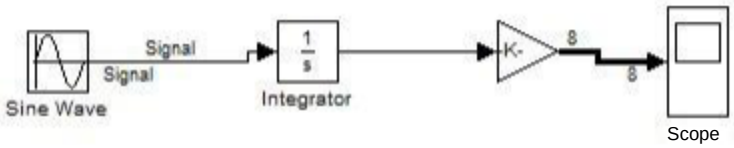
Using Vectors and Discrete parts in Simulink

We can use the vectors in Simulink as well.

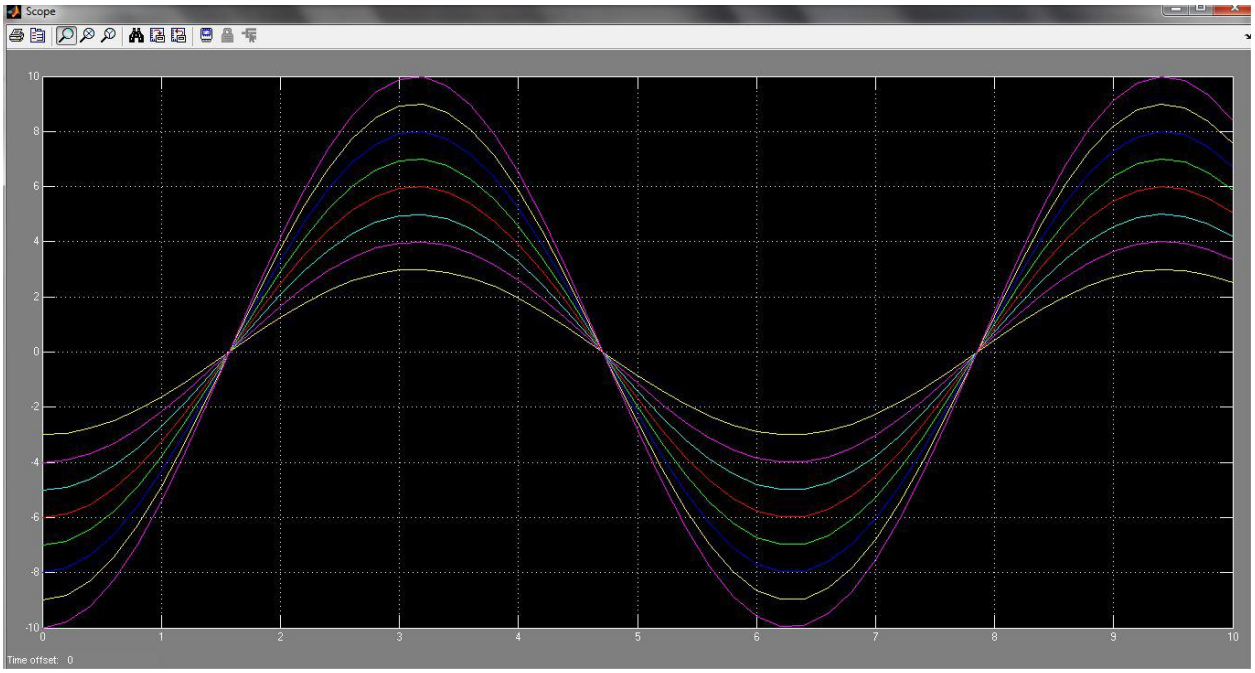
Example: Suppose we want to implement a system which integrates an input sine wave and the output is multiplied by a gain factor of 1. We draw the model as below.



Here gain is set to a constant value of 1. Now change the constant value of gain from 1 to a vector [3:1:10]. This gives a vector in a gain form. From Menu select **Format>signal/port display>signal dimensions**. This will show you the length of the vector in model and bold arrow to represent the vector.

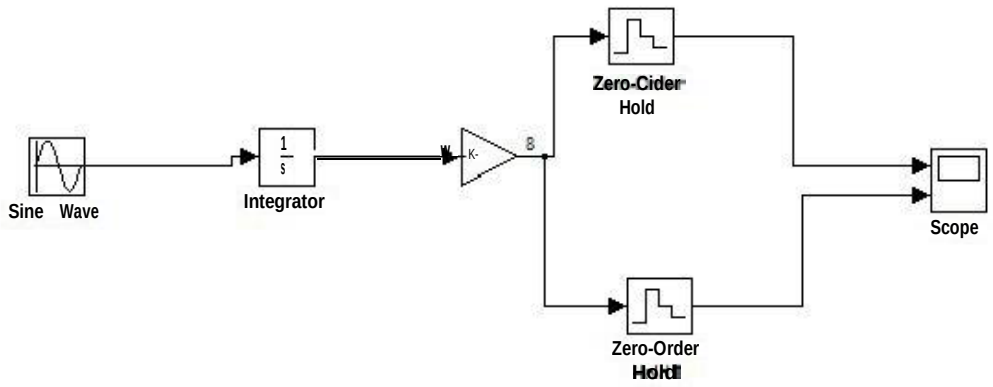


Simulate the system. You will get vector of outputs.



Analog to digital conversion system (ADC)

To implement ADC, we sample thee analog data using sample and hold block. See below the model for analog to digital conversion.



Zero order hold block

The Zero-Order Hold block samples and holds its input for the specified sample period. The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are held for the same sample period.

You specify the time between samples with the Sample time parameter. A setting of -1 means the Sample time is inherited. This block provides a mechanism for discretizing one or more signals in time.

Set the parameters for both zero-order hold blocks as follows.

Block1: Sampling time=ts1

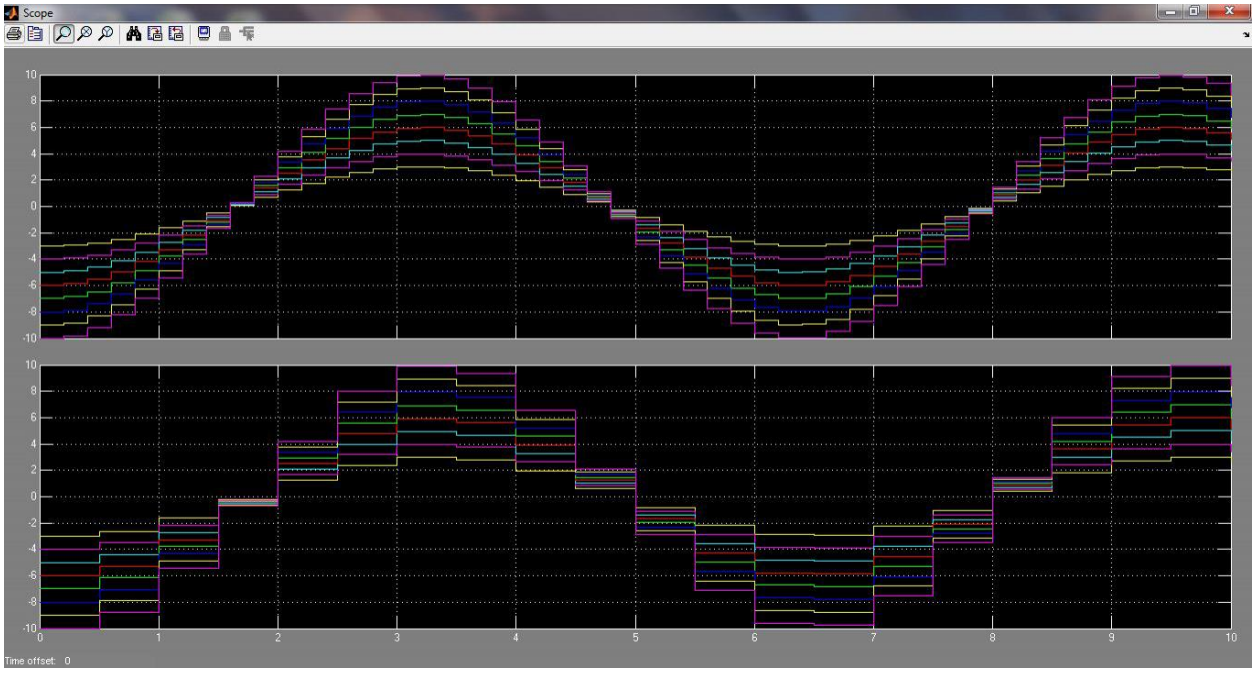
Block2: Sampling time=ts2

In command prompt define these variables as

ts1=0.2;

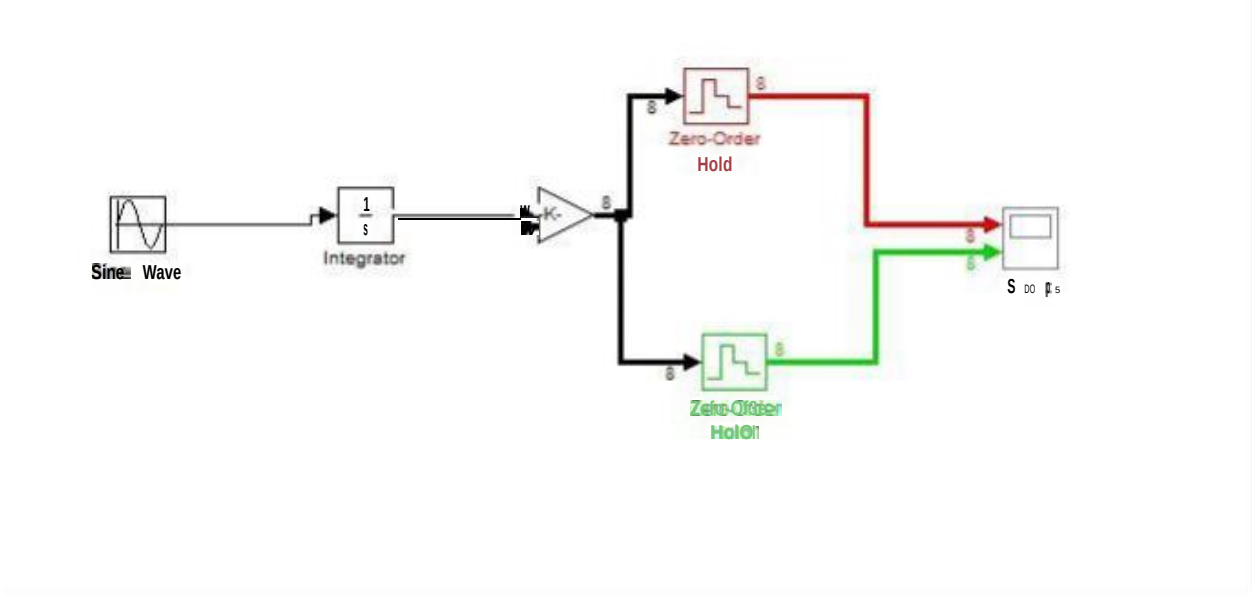
ts2=0.5;

Save and simulate the system. You will get the following output.



Sampling time differences can be observed from both plots.

From the menu choose **Format>port/signal display>sample time colors**. This option will show the discrete part of the system in colors as below.



Using Functions in Simulink

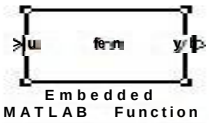
User defined functions can also be implemented in Simulink. Such a function is called embedded function. The function can be introduced no such functional block is available in Simulink.

Take the example of ADC system above. Suppose we want to use the outputs from the discrete system and generate some signaling error. We write the function for the signaling error as below.

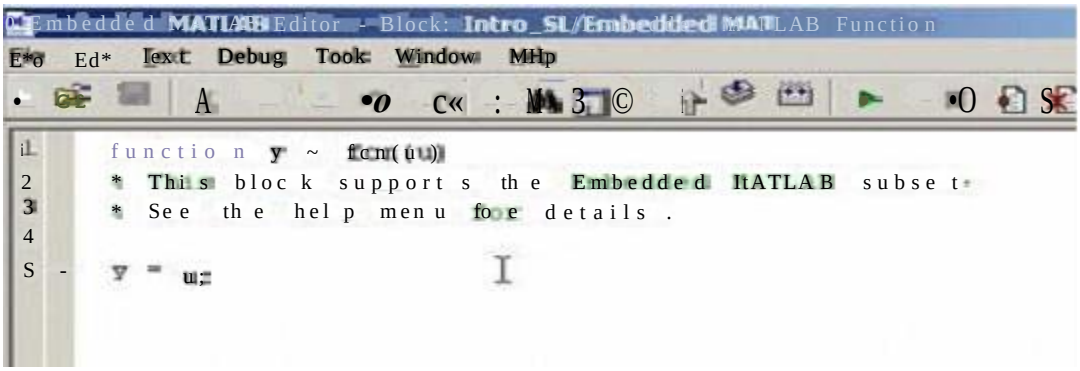
```
%#eml

function y=sampling_error(u1,u2)

weights=1./[3:1:10];

s1=(u1.*weights')'*u1;

s2=(u2.*weights')'*u2;

y=s1-s2;

end
```

Embedded function uses the keyword %#eml as embedded directive for Matlab.

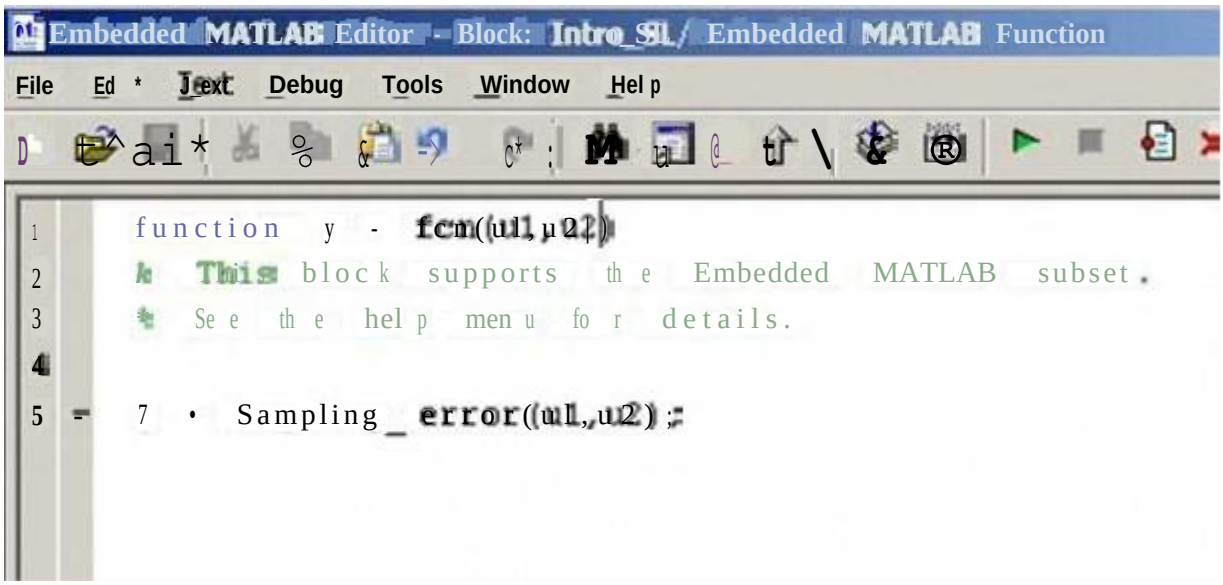
Save the function. And from library import the **Embedded Matlab function** block to your model window.



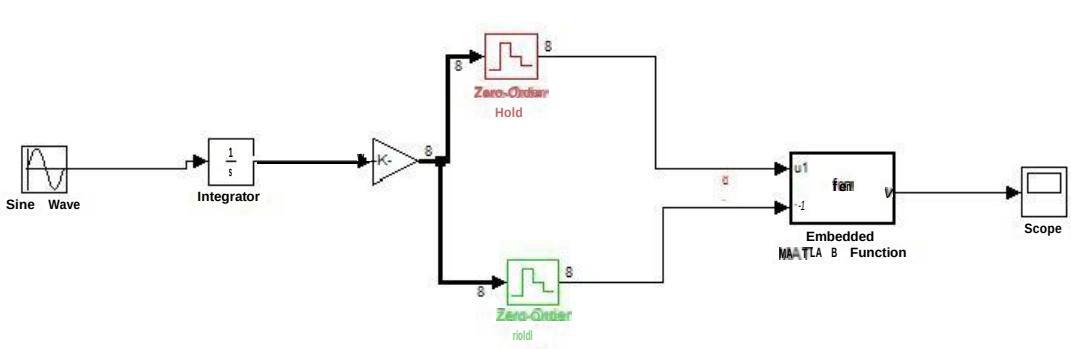
Double-click on the Embedded MATLAB function give us the standard template for an embedded function:



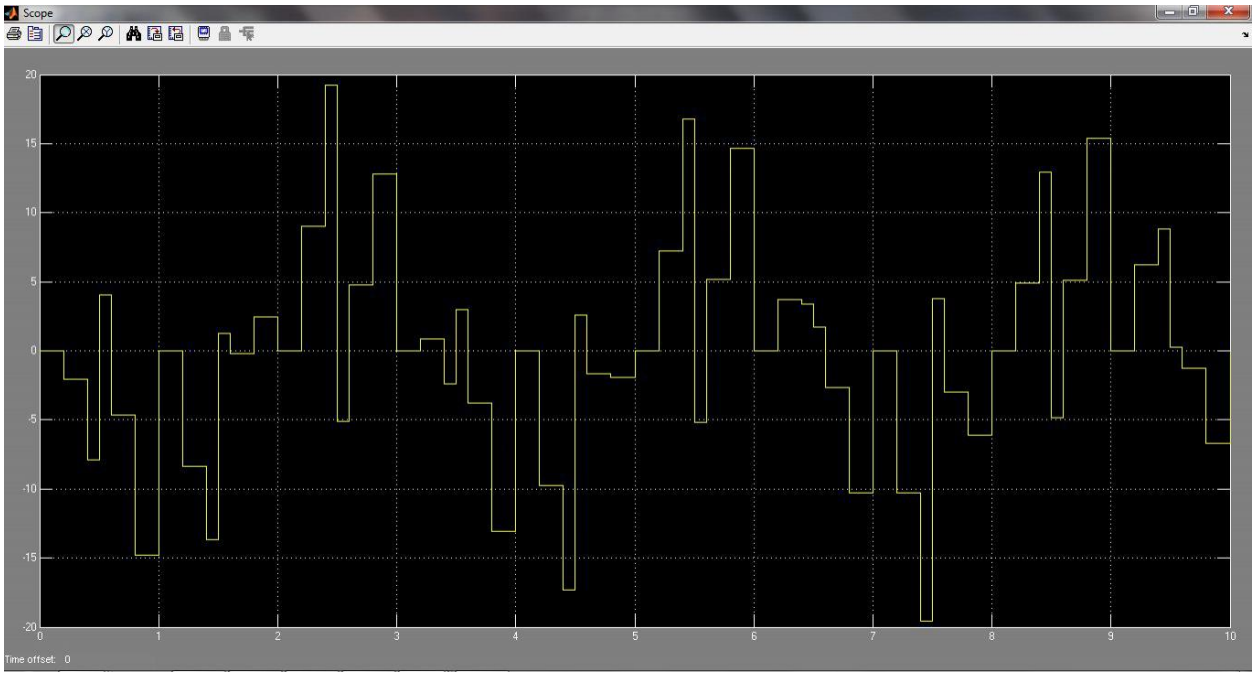
Modify the template so it calls your MATLAB function:



Wire system like this:



Save and Run the system. Output is as below.

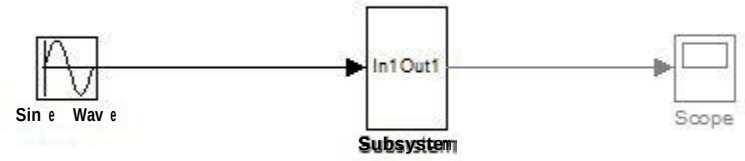


Creating a Subsystem

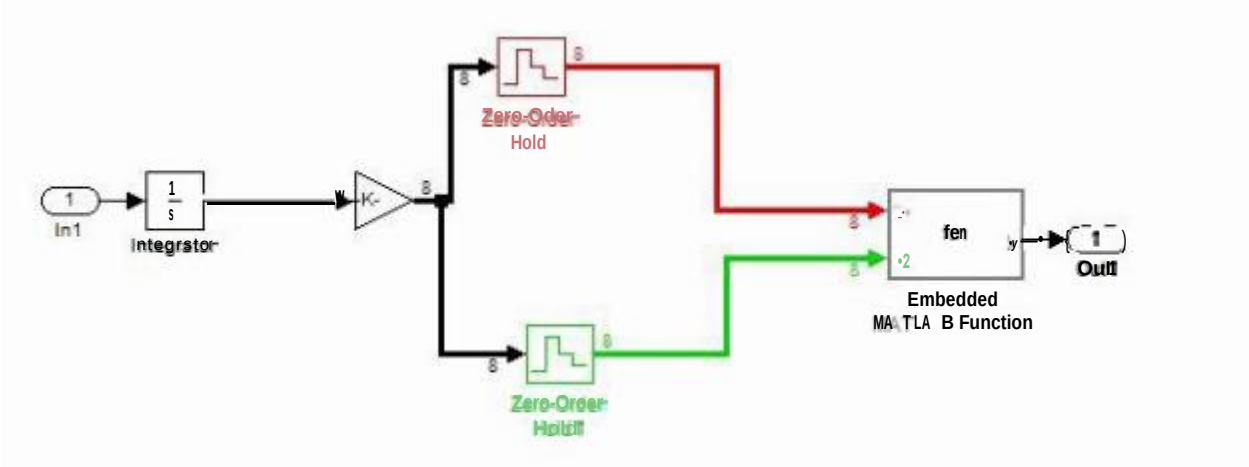
Simulink model or part of the model can be converted to subsystems to show a summary of the blocks as a single block.

In above model, select all blocks except the input sine wave and output scope.

Right click and choose create subsystem. The whole model will be shown as below.

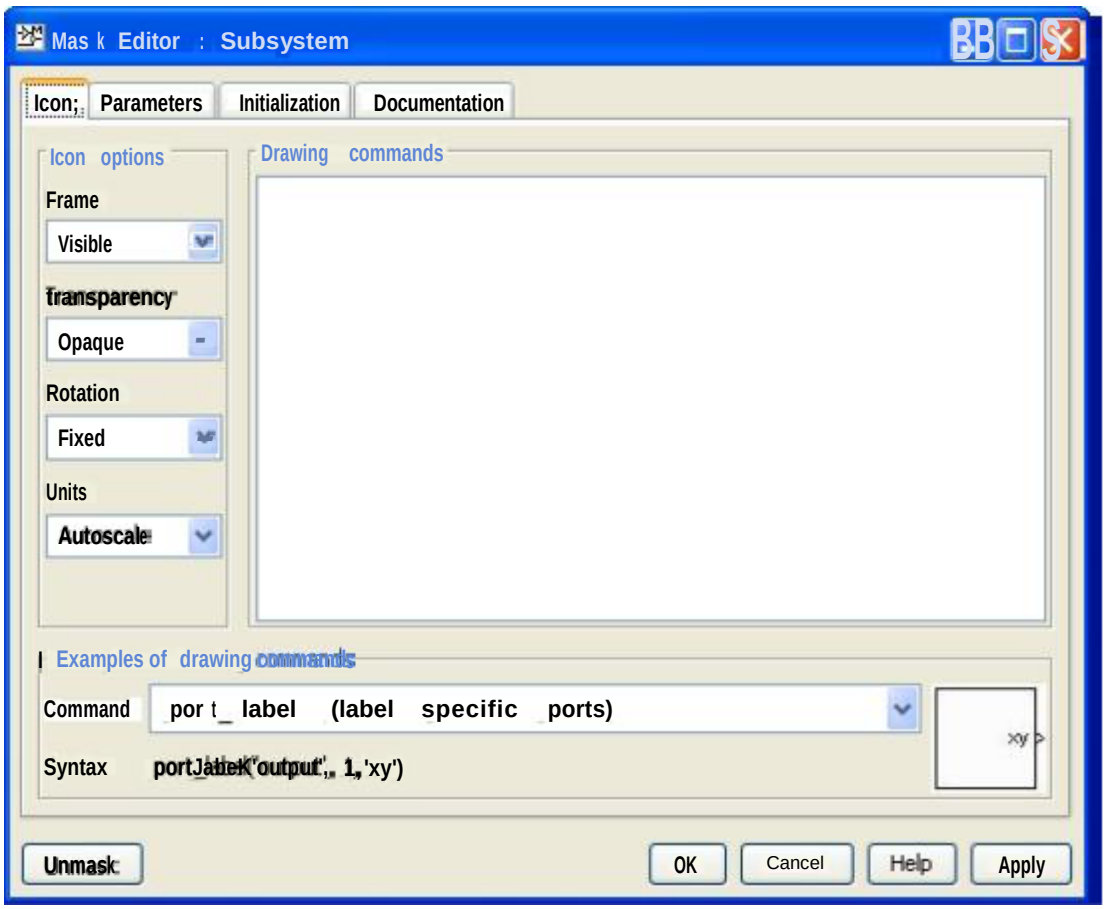


Double clicking the subsystem block will show the internal blocks as follows:

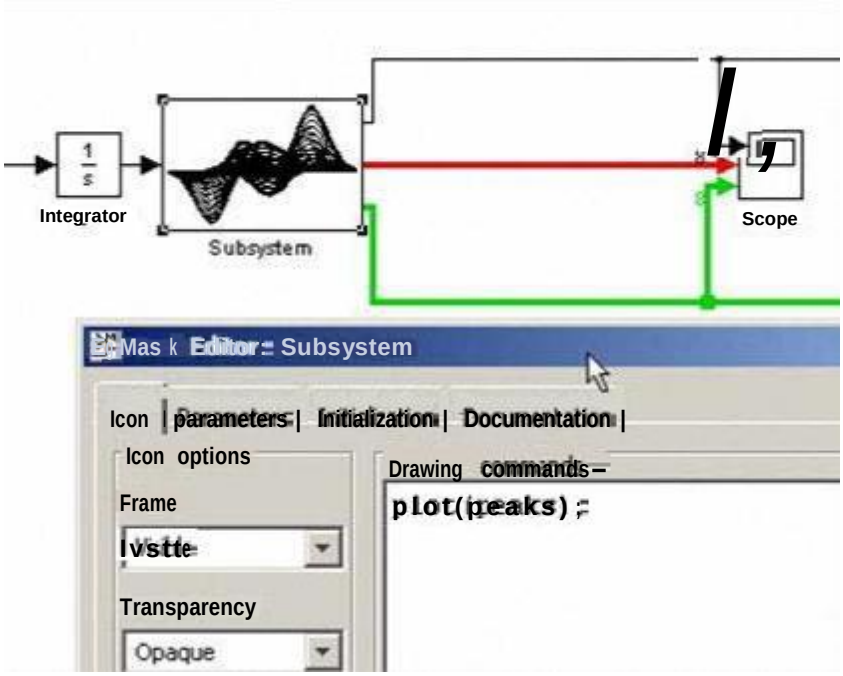


You can even create a subsystem for any number of blocks.

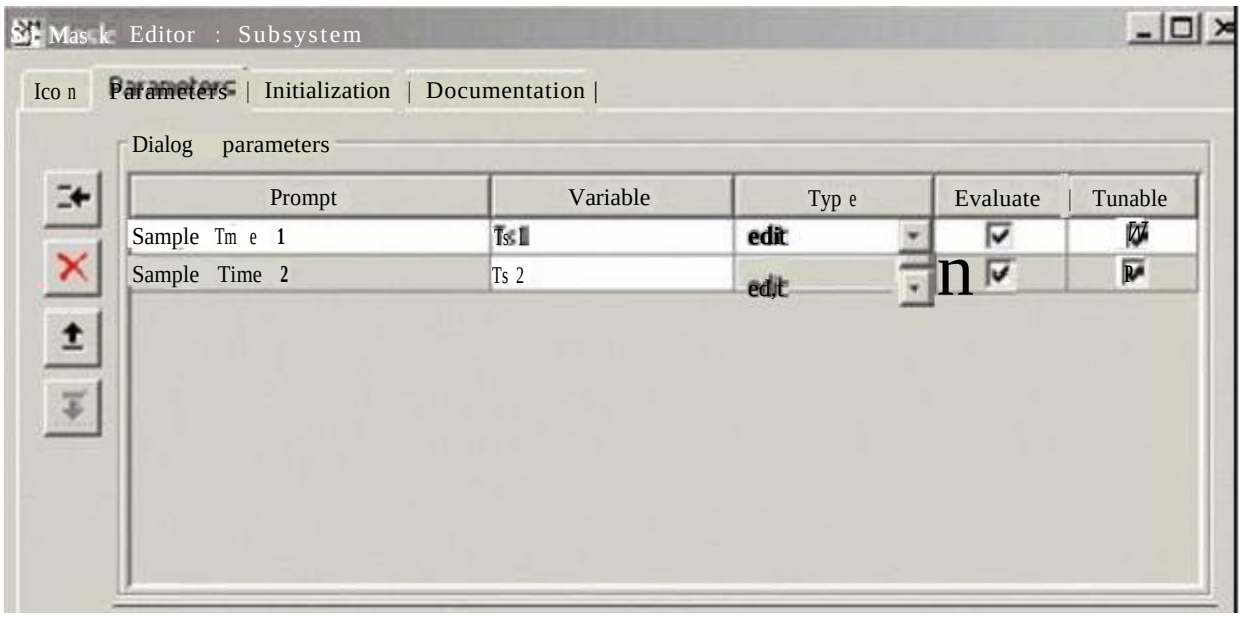
Right-click on the block and select "Edit Mask " in order to open the Mask Editor:



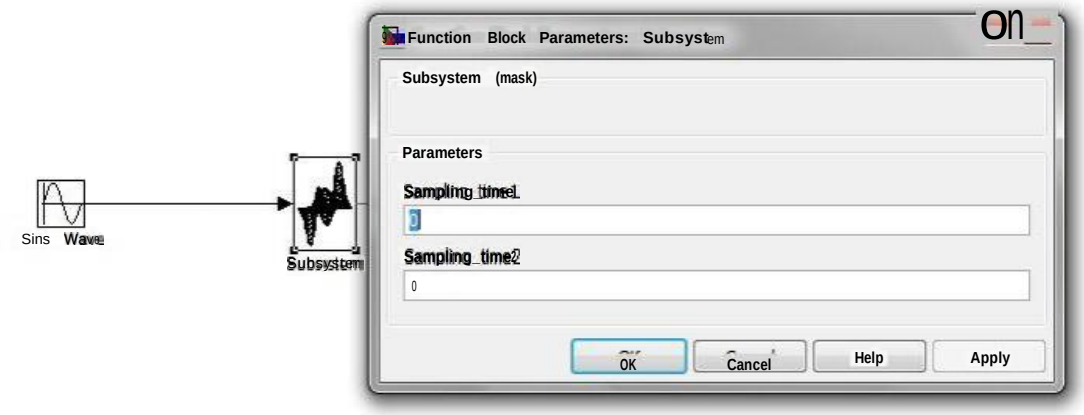
The Mask Editor allows you to change how the subsystem should look, e.g., the subsystem icon



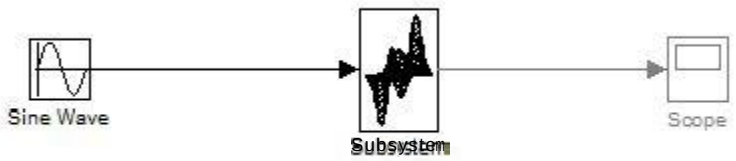
Set the parameters as follows.



Doublecick on the subsystem block to set the parameters.



The whole system looks like below with same output.



Exercise

NOTE: You can assume appropriate value for the amplitude, frequency of the signal (where required). But do mention them in Report.

E1: Take White Noise Signal from Source, and split it into high frequency and low frequency components. Use the Transfer Function block from Continuous and use these transfer functions:

Hook up scopes to the input and the two outputs. Also send the two outputs to the workspace by using to Workspace block from Sink. Show all the results from scopes.

E2:

a) Generate a square signal using Signal Generator and amplify the signal 4 times the original one. The amplified signal is passed through the following systems simultaneously

- An Op-amp based Integrator circuit
- A RC circuit having R=10 K and C=0.1 nF
- A saturating circuit causing saturation (bounds between 0.5 and -0.5)

Outputs from the three circuits are added to give Resultant signal. The resultant signal is passed through the system having transfer function (s+1)/ (s2+1.5s+2) to give the final output. Output should be a sine wave.

Draw the whole system in Simulink. Label the wiring with name of the signal (you can double click on the wire to name it). Show individual output from each system, final output, and amplified output.

b) Generate the sine wave using saw tooth wave at the input using same scenario.

ISRA University Islamabad Campus

Signals & Systems Lab

EXPERIMENT # 11: Graphical User Interface

Name of Student: ……..

Roll No.: ……

Date of Experiment: ……..

Report submitted on: ……..

Marks obtained: ……

Remarks: ……

Instructor's Signature: ……

Lab11

Graphical User Interface

Objective

Objective of the lab is to have basic interface of Matlab GUI, its usage and developing a user desired GUI according to requirement.

GUI

Graphical user interface is a graphical display in one or more windows containing controls called components that enable a user to perform interactive tasks.

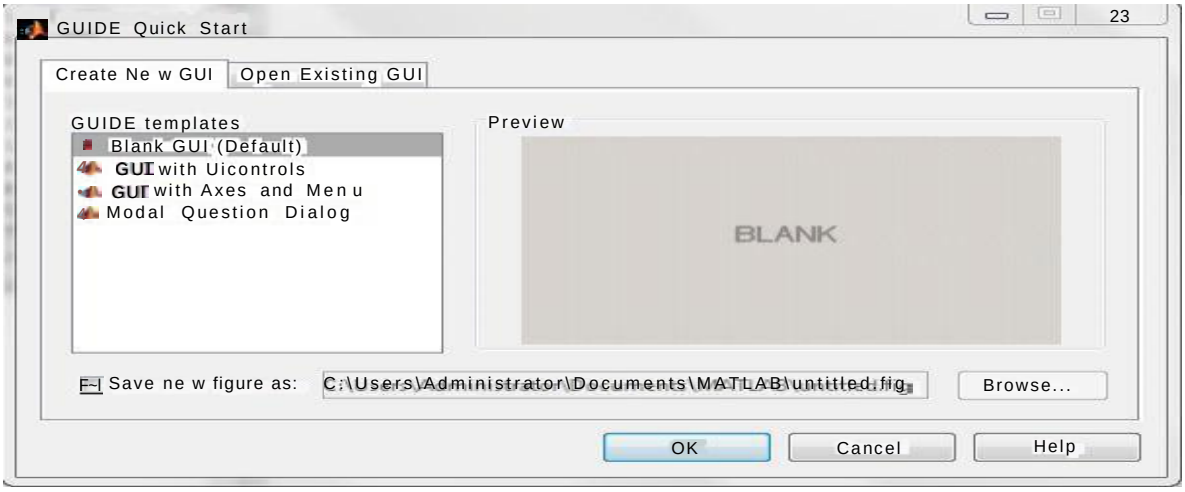
In Matlab, GUI provide a graphical environment in which user can create its own interactive system. Functional approach in MATLAB GUI is component based. Components involve different types of buttons, sliders, axes etc.

Accessing GUI in Matlab

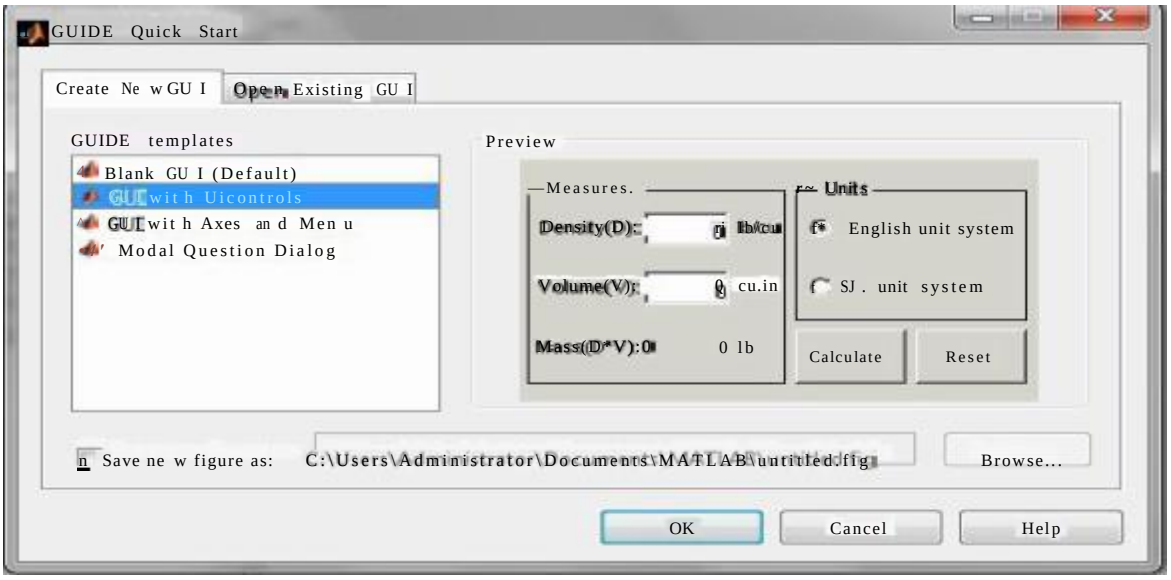
Matlab has separate GUI environment that is called GUIDE (Graphical User Interface development Environment). It provides a set of tools for creating graphical user interfaces (GUIs). In Matlab command window type the following command

>>GUIde

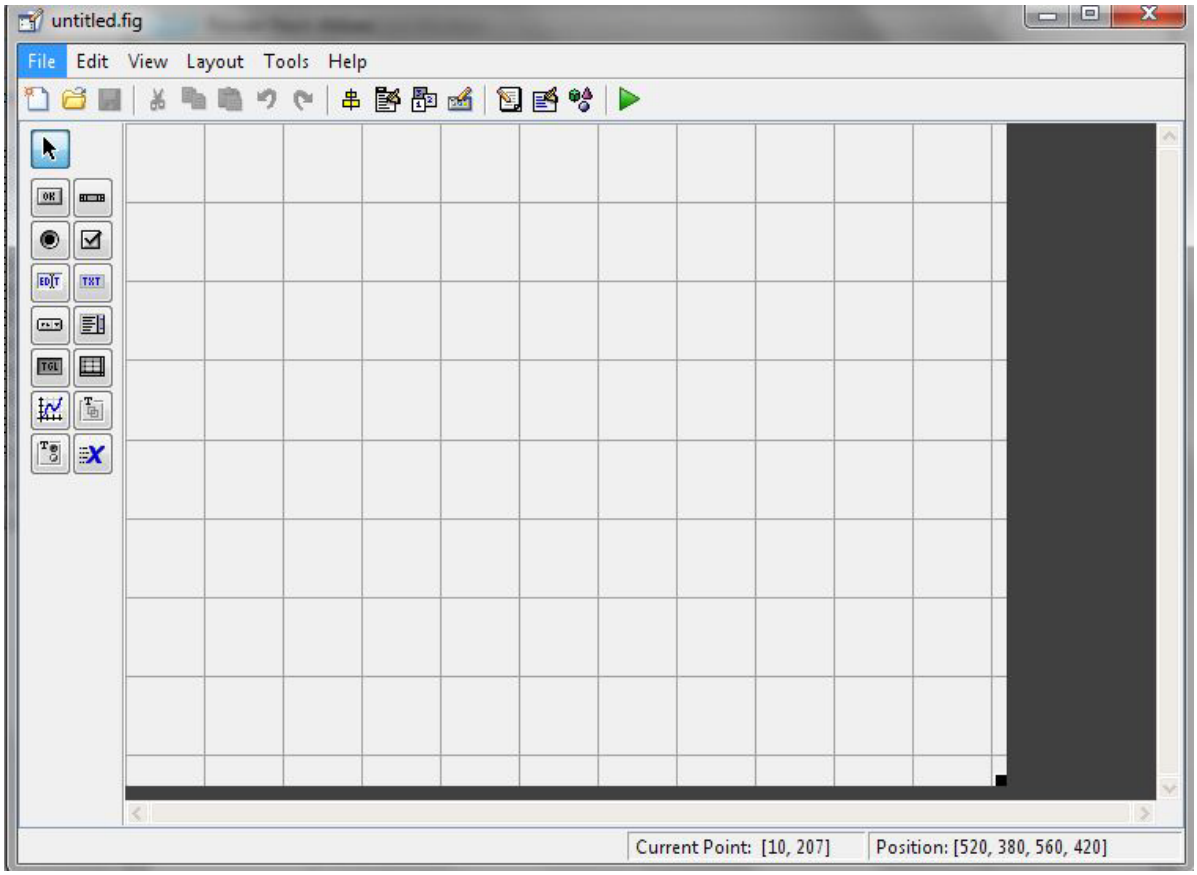
You will get the following window open.



The 1st option prompts you to open a blank GUI while below are some example GUIs available in Matlab library.

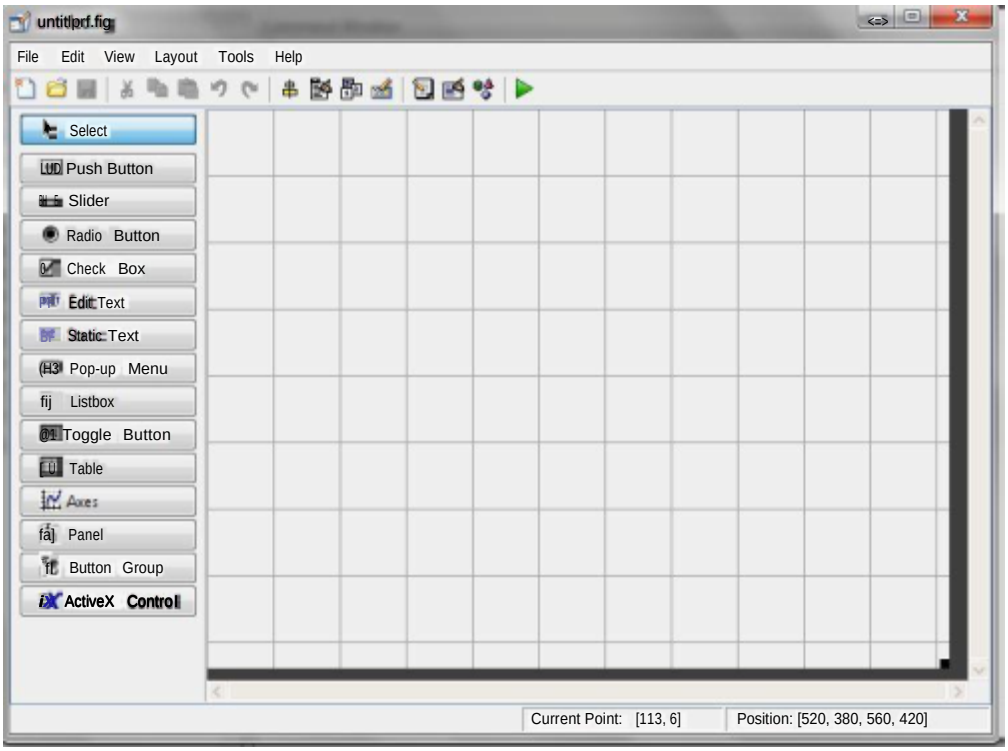


Select Blank GUI from the window. You will get the following window open.



Window opened is called **GUI Layout Editor**. Left side of the window contains different types of buttons. This area is called **component palette**. In order to show names of the components, choose **File>Preferences>Show names in component palette.**

Components will be shown with their names.



Building a Simple GUI

Suppose we want to build a GUI which takes a frequency input from the user and plots sine wave of that frequency.

Followings are the steps to create GUI.

Step1: Understanding the Problem/Requirement

For this GUI we need a user given frequency and in return a sine wave is to be generated. From this we will layout our GUI

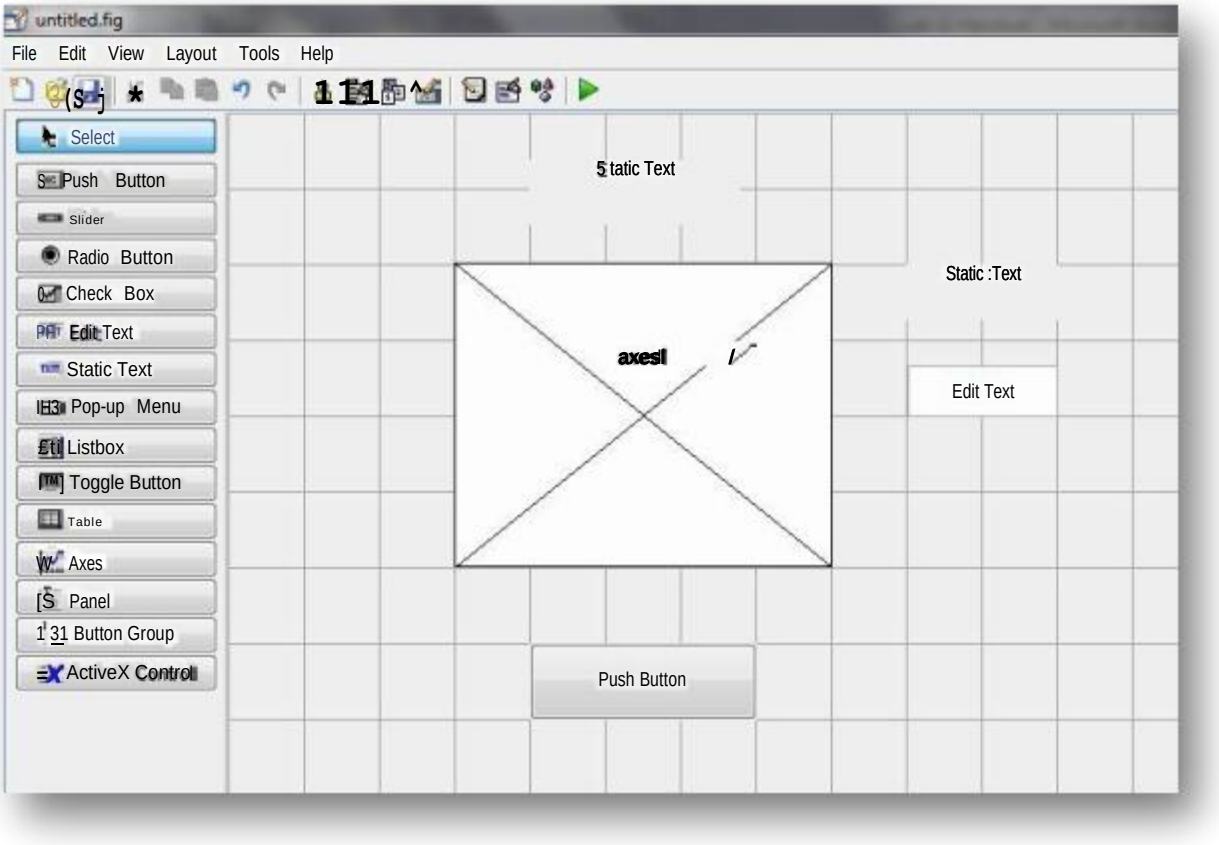
Step 2: Preparing a Draft for GUI.

Second prepare a draft our GUI as per requirement. We need a user given frequency and in return a plot of sine wave. So we need something to take input from the user and a place to plot the wave. In addition we may need an executer button which plots the wave or performs the operation.

Step 3: Laying out GUI components

Second step is to layout your GUI as per draft.

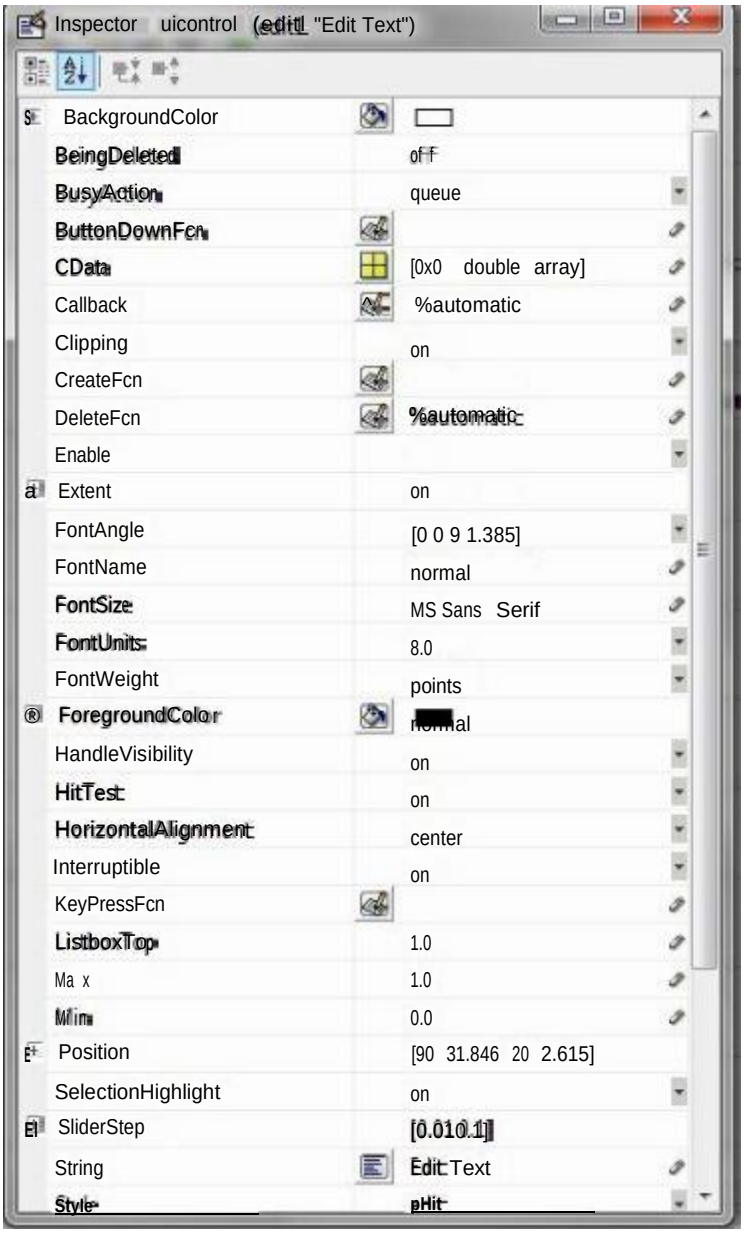
a. For getting an input on run time, we use **Edit text** button. From the component palette select Edit Text and drag it in your layout editor.
b. For plotting we use **Axes** component. So drag the Axes component and place it on layout editor.
c. Drag a **Push Button** and place it on editor, it will be used as executor.
d. In order to show some title for the GUI and input frequency , drag two **Static Text** buttons and place them in layout in the following manner.



Step 4: Setting Parameters

Next step is to set the parameters for each block. Choose **View>Property Inspector**

A new window opens called **Property Inspector** as shown below.



Window contains two columns, 1st column contains the property name while 2nd contains its value. We set the parameters one by one.

Push Button

Double click on the push button; its property inspector window will be opened.

a. From the list, choose **String** property and edit its value from pushbutton to *Plot sin(x)* . It will be shown as visual name of the button.
b. Now select **Font** and change its value to 20.

c. Select **color** and choose use your own choice.
d. Below the **string** there is a property called **Tag**, change its value to *Plotter*. Tag is the name used for programming a function for GUI.

Edit Text

a. Set String Value to 0.
b. Font size to 15.
c. Color of your own choice
d. Set Tag to *input_freq*

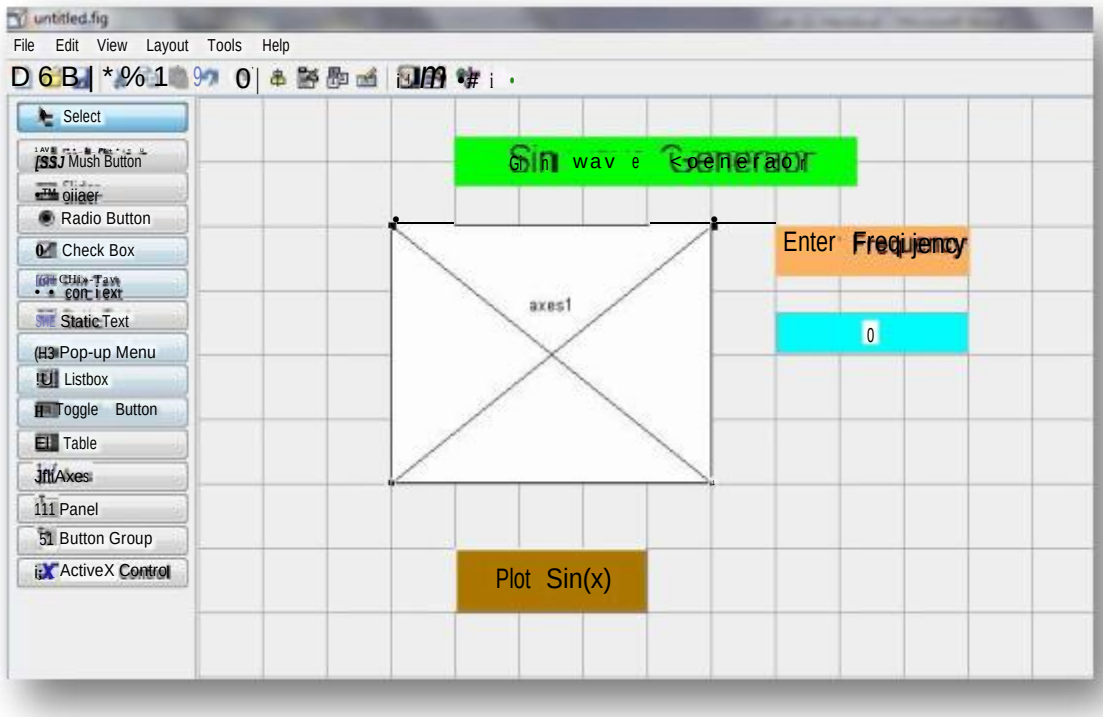
Static Text1

a. Set String to *Sine Wave Generator*
b. Font size to 20.
c. Color of your own choice

Static text2

a. Set String to *Enter Frequency*
b. Set font size to 20.
c. Color of your own choice

Now the GUI looks like below.



Step 5: Saving GUI

Save the GUI by **singengui** or any name. You will get two files upon saving.

1. Fig file
2. M file

Fig file

It is a binary file that contains all information of the blocks used in GUI and saves it as a figure.

M file

When you save your GUI layout, GUIDE automatically generates a file of MATLAB code for controlling the way the GUI works. This file contains code to initialize the GUI and organizes the GUI callbacks.

Callbacks are functions that execute in response to user-generated events, such as a mouse click. Using the MATLAB editor, you can add code to the callbacks to perform the functions you want. You can also add new functions for callbacks to use.

Step 6: Programming GUI

As soon as you save it Matlab should generate the skeleton source code for you and the source code should automatically open in an editor. Before Programming GUI followings are to be kept in mind.

a. GUI in Matlab in essence is a collection of objects. Each object has a unique handle (name). During programming we access the particular object through its handle. Syntax to access the object is **handles. Tag** e.g handles.edittext
b. All handles are stored in a structure named hObject.
c. Name of each function for a block is the same as Tag name of the block in GUI.
d. Objects used in GUI have input and output type of string.
e. For getting an input in GUI from external, we use get() functions. Function is used as get(H,'property name')
f. We setting a value as output we use set() function. Function is used as set(H,'property name',property value)

In our GUI we can define the flow of the GUI as getting the input from the user, generating a sine wave and plotting it. All can be done if we program the push button which takes input, generate sine wave and plot it.

Traverse through m file and choose plotter_callback or click the f() button on the main meu which shows you the list of functions available in m file and select plotter_callback. It will take you to the code of that function. This function is for push button as its Tag was set to plotter.

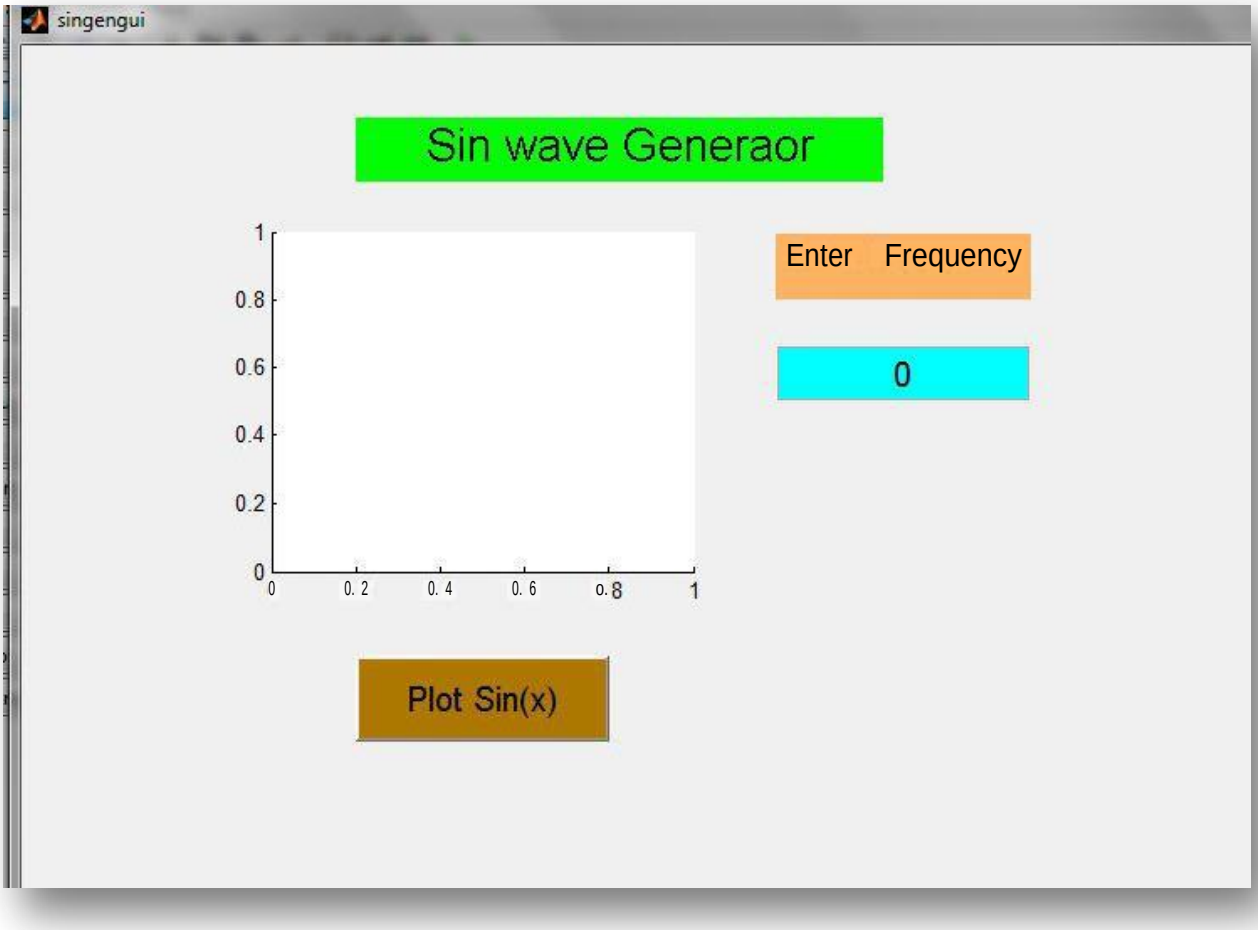
Add the following code below the function definition line under the comment lines.

```
freq=str2num(get(handles.input_freq,'string'));  %input from the Edit Text
button and converted to number type
x=0 :(2*pi)/100:2*pi;
y=sin(x*freq);
% accessing the axes block for plotting
axes(handles.axes1)
plot(y)
```

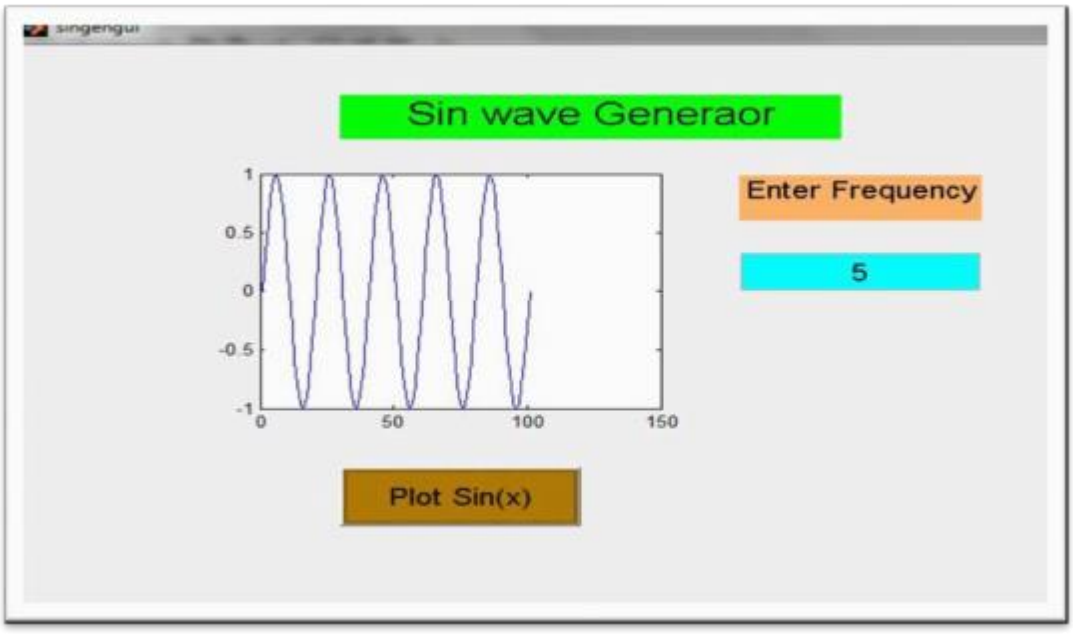
Since frequency is of numeric type so input from the user is converted to number and saved as freq. Next we generate a sine wave of freq frequency with x number of points. To plot the sine wave we access the axes through handles and plot the wave.

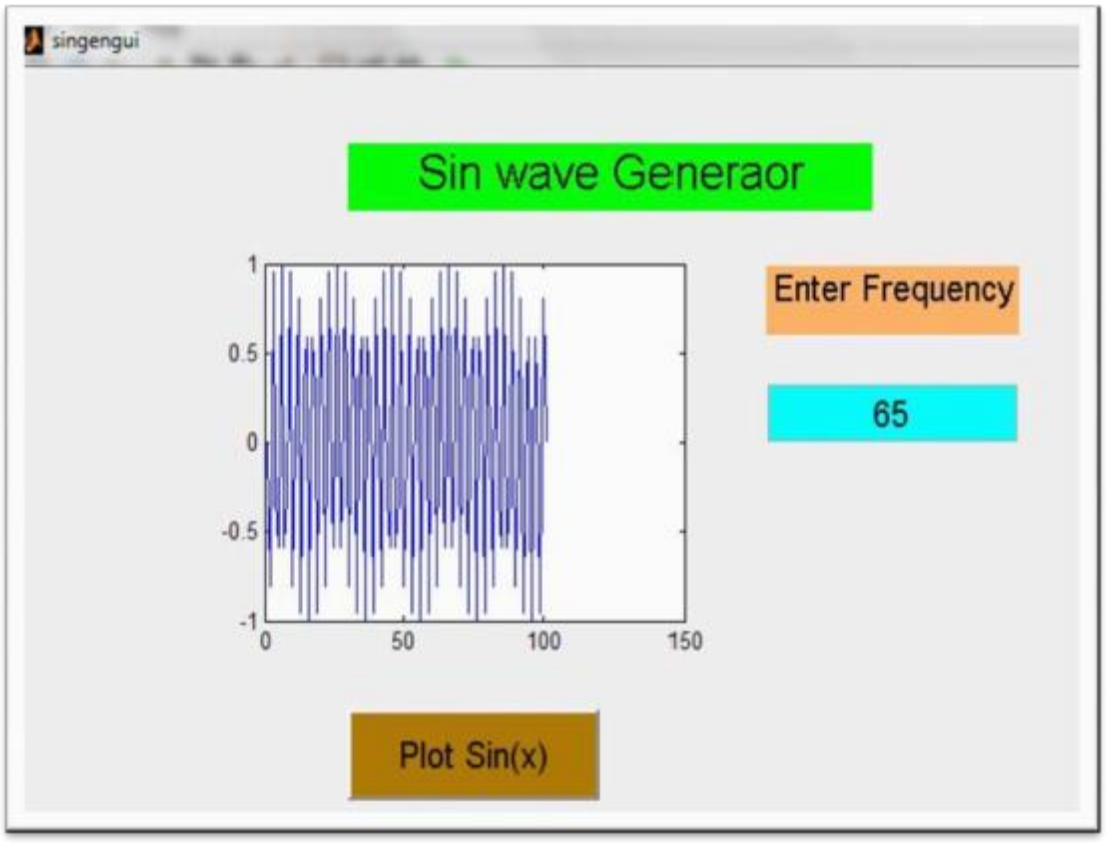
Step 7: Running GUI

Save the code and run the fig file. You will get the following GUI



Enter your desired frequency and click the push button named Plot sin(x). You will see the sine wave of that frequency.


Sin wave Generaor
Enter Frequency
5
Plot Sin(x)


singengui
Sin wave Generaor
Enter Frequency
65
Plot Sin(x)

Example

Build a Matlab GUI for addition of two user input numbers**.**

Step1: **Understanding the statement**

We need two input numbers from the user and their output as a sum

Step2: **Drafting GUI**

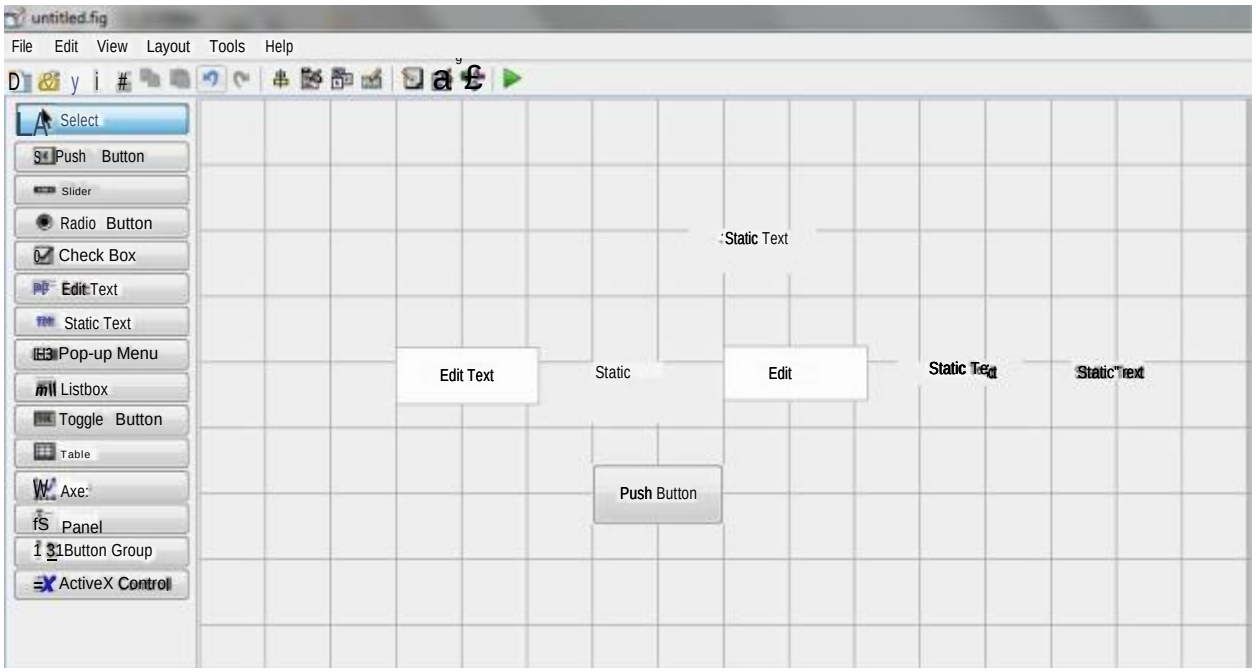
For GUI we would need two buttons for input, one for output, an executer and some for titles of GUI.

Step3: **Laying out GUI**

Import the following components in Layout editor.

a. 2 Edit Text buttons
b. 4 Static Text buttons
c. 1 Push Button

Place them in the following manner.



Step 4: Parameter Setting

Set the parameters as following.

Static text1: String=Adder, Font=20, Color=choice

Static Text2: String=+, Font=15, color=choice

Static text3: String= =, Font=15, Color=choice

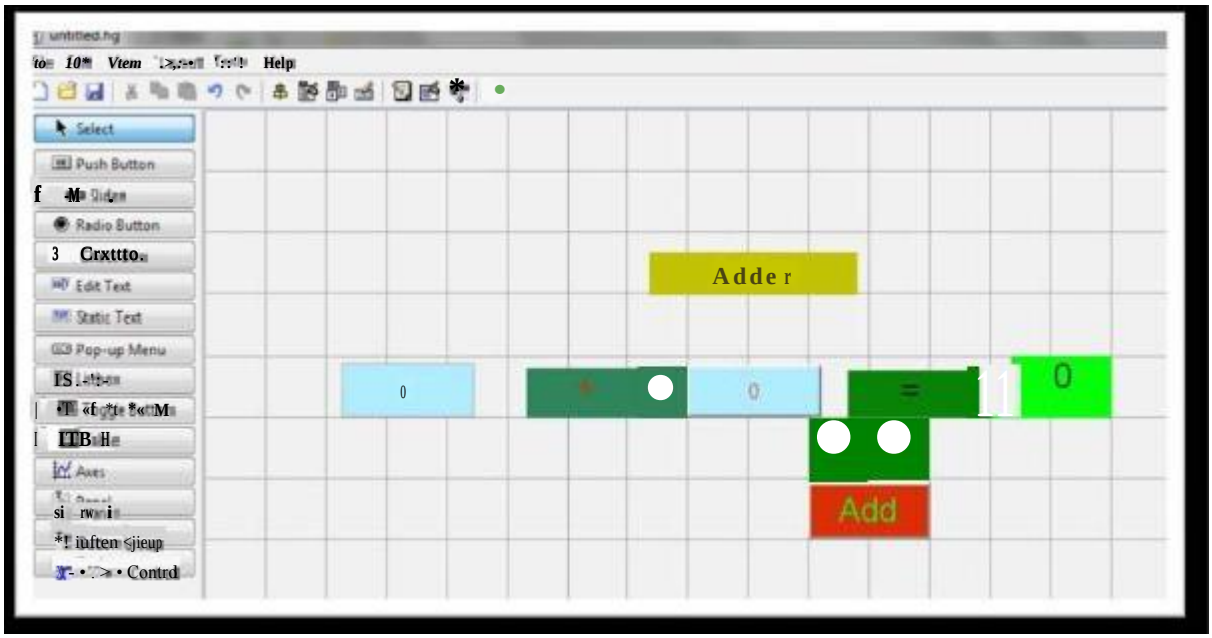
Static text4: String=0, Font=20, Color=choice, Tag=output

Edit text1: String=0, Font=20, Color=choice, Tag=input1

Edit text2: String=0, Font=20, Color=choice, Tag=input2

Push Button: String=Add, Font=20, Color=choice, Tag=adder

Now the GUI looks like it.



Step 5: Programming GUI

Flow of the GUI shows that push button takes input from the user from text buttons, add them and set it as output. If user doesn't input any number, it should be taken as zero.

In m file, select input1_callback function from the functions list and add the following code below the comment lines.

```
input=get(hObject,'string'); %input from the user
if(isempty(input))
    set(hObject,'string','0'); %if input is empty/no input,
consider it zero
end
guidata(hObject,handles); %saves and update all the handles
```

This piece of code simply makes sure that the input is well defined. We don't want the user to put in inputs that aren't numbers! The last line of code tells the gui to update the handles structure after the callback is complete. The handles stores all the relevant data related to the GUI. This function is used so that the handles are always updated after each callback.
Copy the same code to input2_callback.
Now access the adder_callback function for the pushbutton and add the following code.

```
a=str2num(get(handles.input1,'string'));
b=str2num(get(handles.input2,'string'));
c=a+b;
total=num2str(c);
set(handles.output,'string',total);
guidata(hObject,handles);
```

Explanation

1. The first two lines of code above take the strings within the Edit Text components, and stores them into the variables a and b. Since they are variables of String type, and not number type, we cannot simply add them together. Thus, we must convert a and b to number type before MATLAB can add them together.

   We can convert variables of String type to number type using the MATLAB command str2num**(**String argument**)**. Similarly, we can do the opposite using num2str**(**Number argument**)**.
2. The following line of code is used to add the two inputs together.
   C=a+b;

3. The next line of code converts the sum variable to String type and stores it into the variable total.
   total = num2str(c);

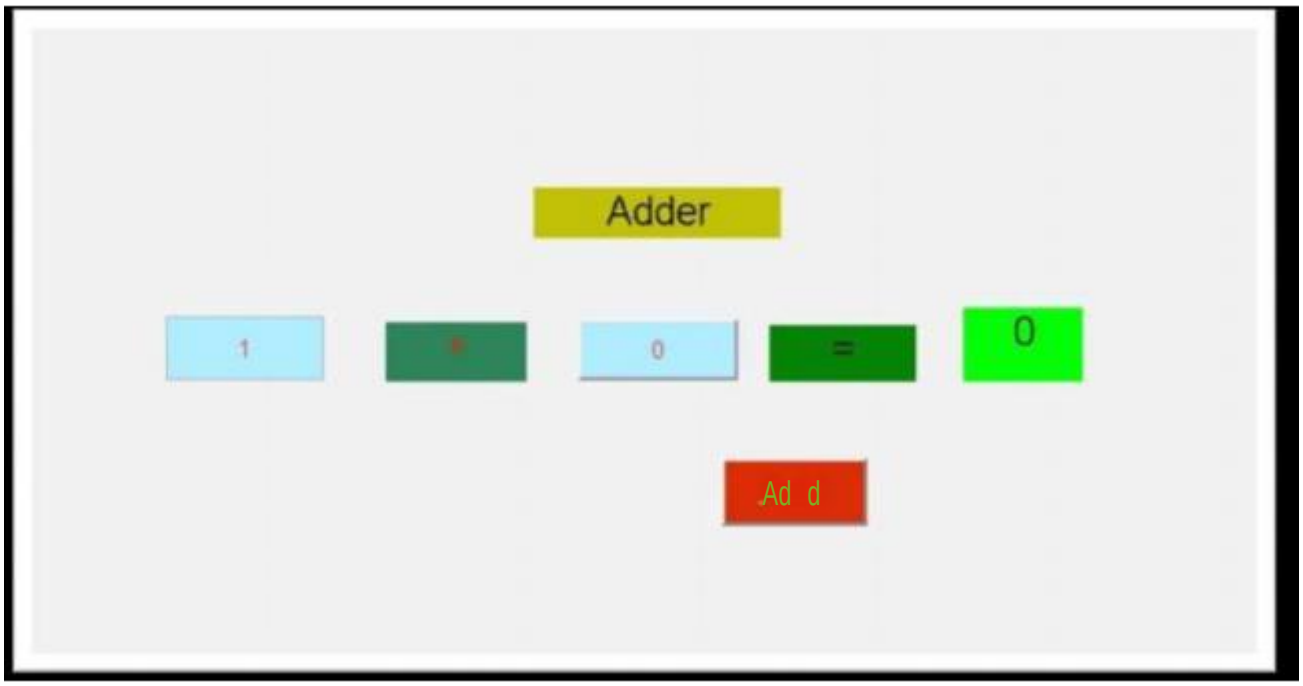
The reason we convert the final answer back into String type is because the Static Text component does not display variables of number type. If you did not convert it back into a String type, the GUI would run into an error when it tries to display the answer.

4. Now we just need to send the sum of the two inputs to the answer box that we created. This is done using the following line of code. This line of code populates the Static Text component with the variable total.
   set(handles.output,'String',total);

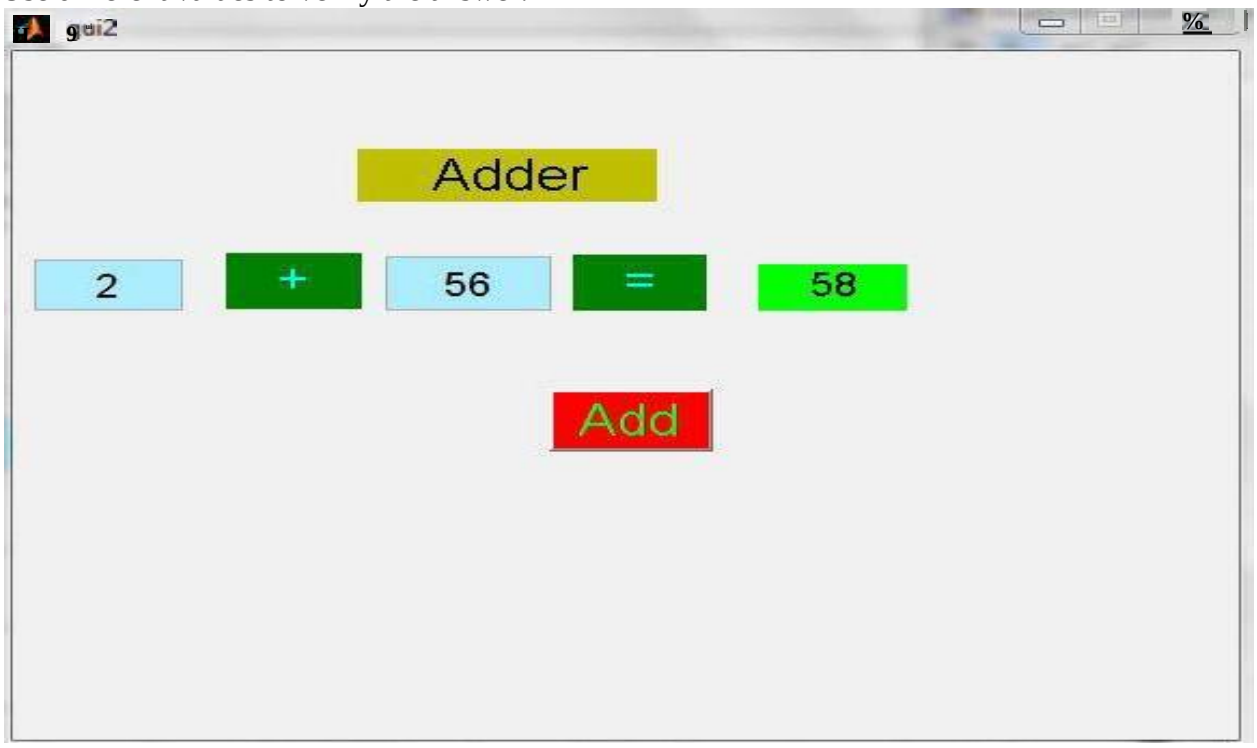
5. The last line of code updates the handles structures as was previously mentioned.

guidata(hObject, handles);

Step 6: Save the GUI and Run it.
Following window will open with GUI.



Use different values to verify the answer.



Exercise

E12.1 Build a Matlab GUI which takes input from user, a temperature value in Celsius and returns the fornhight temprature. Forn hight temprature should be displayed as a stem of hight equal to the fornhight temprature. Mention the code, output figure and properties of the used functional blocks.

E12.2: Build a Matlab GUI that performs the following operation.

1. Generation of a square wave of given time period
2. Generation of a cosine wave of given frequency

GUI should contain a single interface for all the operations which will be selected by the user. Mention the code, output figures and Properties of the blocks used.