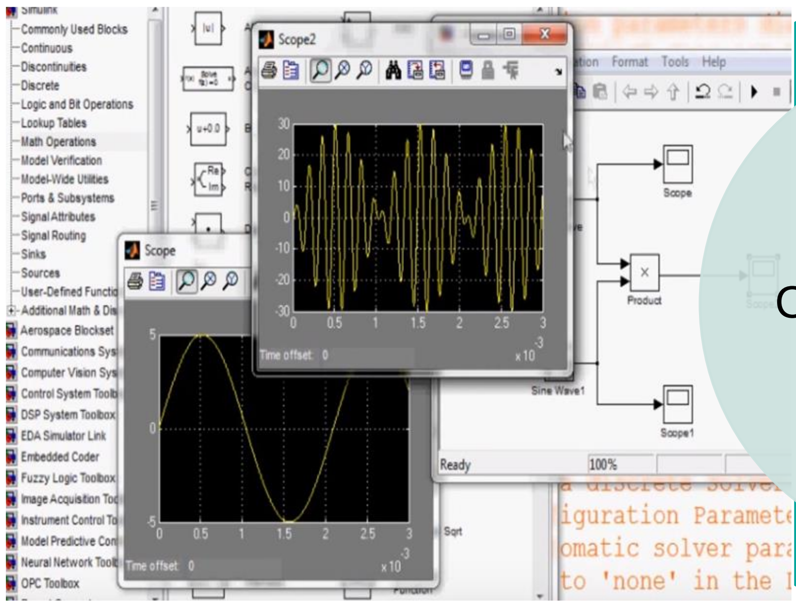


Digital Communications LAB

This is a fourth-year students' laboratory, it represents the practical side of the theoretical study of Digital Communications, which is given for the Information engineering branch during two semesters.

LABORATORY OBJECTIVES.

1. Introduce the basics of Matlab and how to use it for programming and dealing with variables, matrices, functions and graphs.
2. Teach the student how to use Simulink to design models for different communication systems and then simulate the behavior of these systems.
3. Teaching students the basics of digital communication systems through the implementation of the following circuits:
   a. DSB Modulation
   b. DSSS Direct Sequence Spread Spectrum
   c. Frequency Modulation FM
   d. Frequency-shift keying
   e. Additive white Gaussian noise (AWGN)
   f. Amplitude modulation (AM)
   g. BPSK Modulation & BER

Digital Communication Lab/ Four Class

UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Introduction to MATLAB

Assist Lec. Noor Abdul Khaleq

MATLAB (=Matrix laboratory) is an interactive matrix-based software package intended for complex scientific computations. MATLAB is an abbreviation for MATRIX LABORATORY, so it is well suited for matrix manipulation and problem solving related to Linear Algebra. MATLAB offers lots of additional Toolboxes for different areas such as Control Design, Image Processing, Digital Signal Processing, etc. MATLAB is the de facto software used in simulation of communications systems and digital signal processing. It has several advantages over high-level programming languages, such as C/C++, Fortran etc.

Why MATLAB

- High-Level Language
- Widely used as a visualization and teaching tool
- Prototype for Researchers
- Tremendous number of functions
- Documentation which is really helpful
- Comprehensive Toolboxes for easy access to standard algorithms in a variety of specializations, like Image Processing, DSP, Neural Networks, Statistics and many more.

What is MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

Advantages:

- A large set of toolboxes is available. A toolbox is a collection of MATLAB functions specific for a subject, e.g. the signal processing toolbox or the communications toolbox.
- At any time, variables (results of simulations) are stored in the workspace for debugging and inspection,
- Excellent visualization (plots) capabilities,
- Easy programming, e.g. it is not required to define variable types (unless needed). All variables are usually of type double (64 bit representation).
- Quick code development.

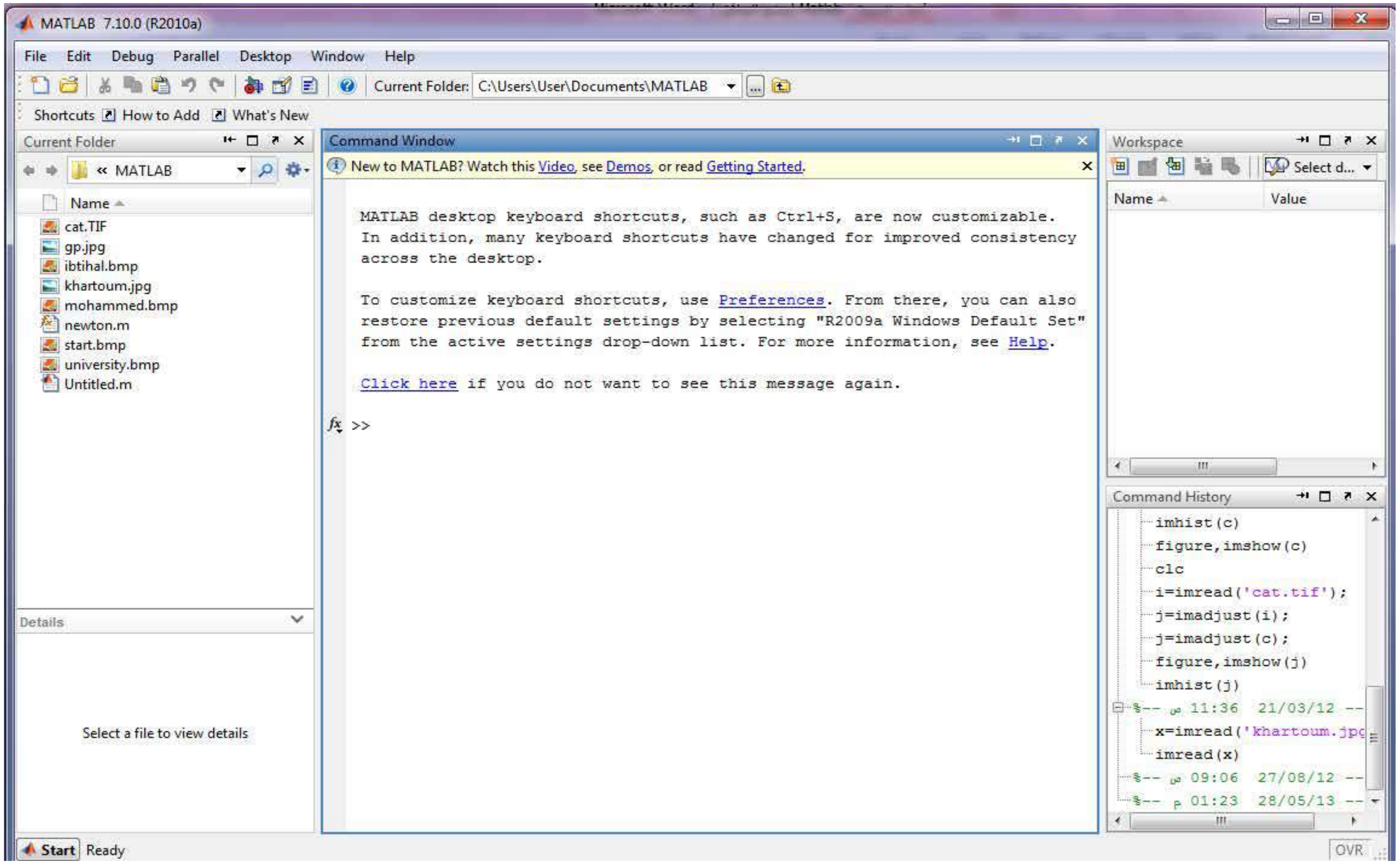
Disadvantages:

- ❖ Very expensive for non-students, although there are some free clones, such as Octave or Sscilab that are MATLAB compatible (but not 100%).

- ❖ Code execution can be slow if programmed carelessly without vectorization.

- ❖ Algorithms developed in MATLAB will need to be translated in C or assembly if to be used in DSPs.

When you start MATLAB®, the desktop appears in its default layout



New to MATLAB? Watch this Video, see Demos, or read Getting Started.

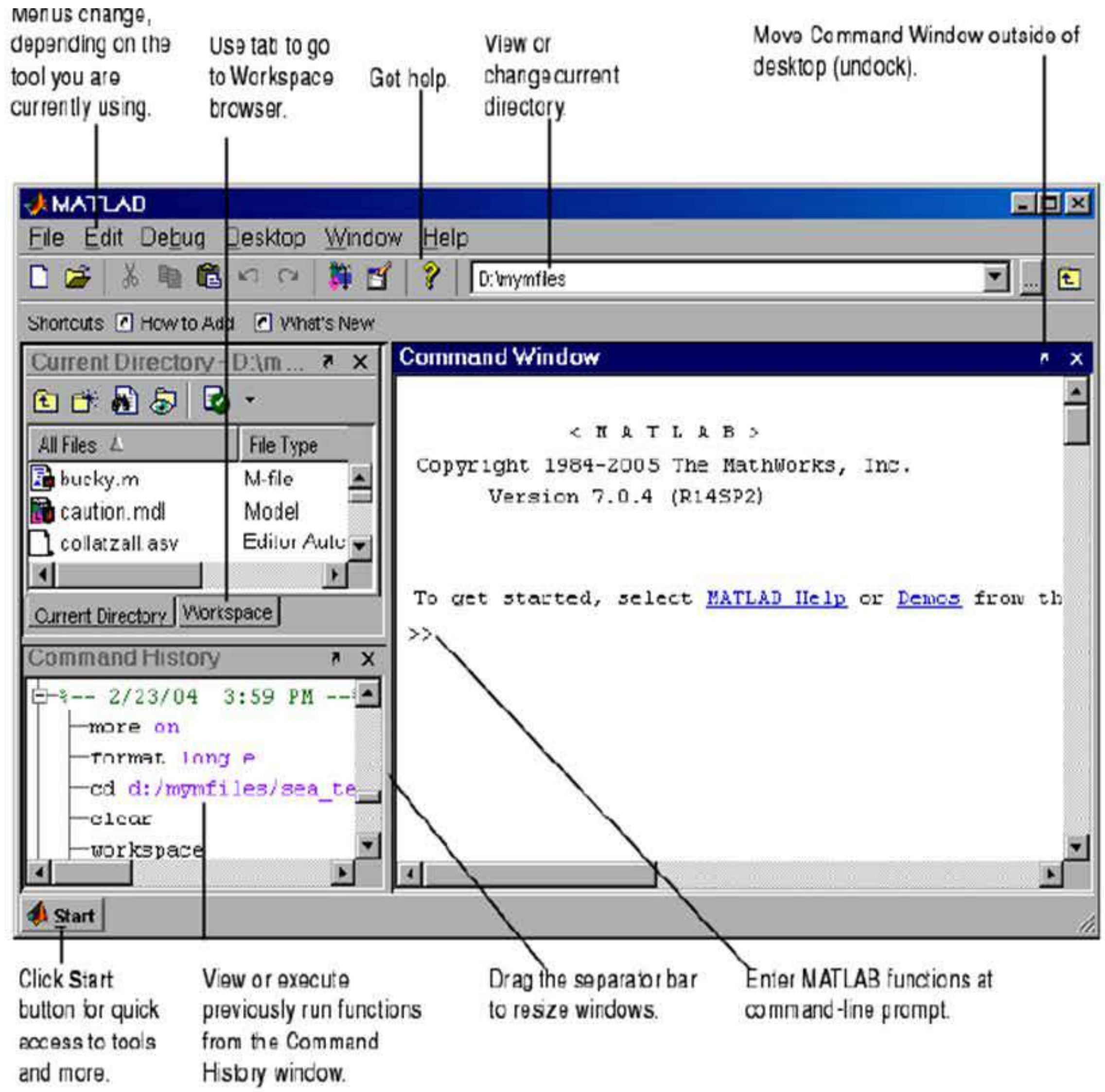
```
MATLAB desktop keyboard shortcuts, such as Ctrl+S, are now customizable.
In addition, many keyboard shortcuts have changed for improved consistency
across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you can also
restore previous default settings by selecting "R2009a Windows Default Set"
from the active settings drop-down list. For more information, see Help.

Click here if you do not want to see this message again.

fx >>
```

```
imhist(c)
figure,imshow(c)
clc
i=imread('cat.tif');
j=imadjust(i);
j=imadjust(c);
figure,imshow(j)
imhist(j)
%-- ص 11:36  21/03/12 --
x=imread('khartoum.jpg
imread(x)
%-- ص 09:06  27/08/12 --
%-- م 01:23  28/05/13 --
```

The desktop includes these panels:

- **Current Folder** — Access your files.
- **Command Window** — Enter commands at the command line, indicated by the prompt (>>).
- **Workspace** — Explore data that you create or import from files.
- **Command History**

As you work in MATLAB, you issue commands that create variables and call functions. The Command window is the main window. Use the Command Window to enter variables and to run functions and M-files scripts.


Menus change, depending on the tool you are currently using.
Use tab to go to Workspace browser.
Get help.
View or change current directory.
Move Command Window outside of desktop (undock).
MATLAB
File Edit Debug Desktop Window Help
D:\mymfiles
Shortcuts How to Add What's New
Current Directory - D:\m...
All Files
File Type
bucky.m
M-file
caution.mdl
Model
collatzall.asv
Editor Auto
Current Directory
Workspace
Command History
Command Window
Start
Click Start button for quick access to tools and more.
View or execute previously run functions from the Command History window.
Drag the separator bar to resize windows.
Enter MATLAB functions at command-line prompt.

```
                < M A T L A B >
  Copyright 1984-2005 The MathWorks, Inc.
        Version 7.0.4 (R14SP2)


 To get started, select MATLAB Help or Demos from th
>>
```

```
%-- 2/23/04   3:59 PM --
    more on
    format long e
    cd d:/mymfiles/sea_te
    clear
    workspace
```

Matrices and Arrays

MATLAB is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.

All MATLAB variables are multidimensional *arrays*, no matter what type of data. A *matrix* is a two-dimensional array often used for linear algebra.

.

Digital Communication Lab/ Four Class

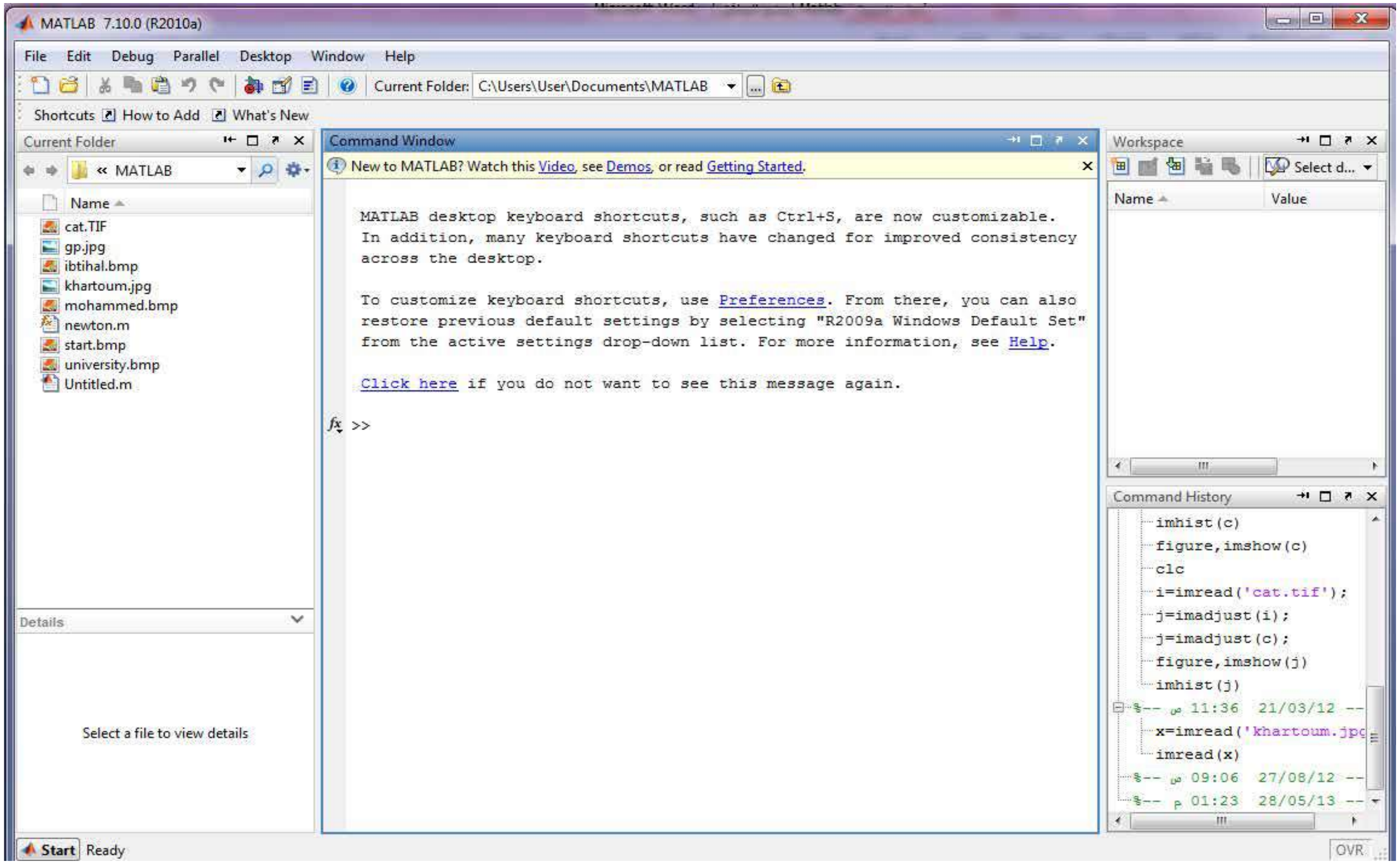
UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Assist Lec. Noor Abdul Khaleq

Desktop Basics

When you start MATLAB®, the desktop appears in its default layout


MATLAB 7.10.0 (R2010a)
File Edit Debug Parallel Desktop Window Help
Current Folder: C:\Users\User\Documents\MATLAB
Shortcuts How to Add What's New
Current Folder
« MATLAB
Name
Details
Select a file to view details
Command Window
Workspace
Select d...
Name Value
Command History
Start Ready
OVR

cat.TIF
gp.jpg
ibtihal.bmp
khartoum.jpg
mohammed.bmp
newton.m
start.bmp
university.bmp
Untitled.m

New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
MATLAB desktop keyboard shortcuts, such as Ctrl+S, are now customizable.
In addition, many keyboard shortcuts have changed for improved consistency
across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you can also
restore previous default settings by selecting "R2009a Windows Default Set"
from the active settings drop-down list. For more information, see Help.

Click here if you do not want to see this message again.

fx >>
```

```
imhist(c)
figure,imshow(c)
clc
i=imread('cat.tif');
j=imadjust(i);
j=imadjust(c);
figure,imshow(j)
imhist(j)
%-- ص 11:36  21/03/12 --
x=imread('khartoum.jpg
imread(x)
%-- ص 09:06  27/08/12 --
%-- م 01:23  28/05/13 --
```

As you work in MATLAB, you issue commands that create variables and call functions.
For example, create a variable named (a) by typing this statement at the command line:

```
a = 1
```

MATLAB adds variable (a) to the workspace and displays the result in the Command Window.

```
a =
1
```

Create a few more variables.

```
b = 2
b =
2
c = a + b
c =
3
d = cos (a)
d =
0.5403
```

When you do not specify an output variable, MATLAB uses the variable (ans), short for *answer*, to store the results of your calculation.

```
Sin (a)
ans =
0.8415
```

If you end a statement with a semicolon, MATLAB performs the computation, but suppresses the display of output in the Command Window.

```
e = a*b;
```

You can recall previous commands by pressing the up- and down-arrow keys, ↑ and ↓. Press the arrow keys either at an empty command line or after you type the first few characters of a command. For example, to recall the command b = 2, type b, and then press the up-arrow key.

Matrices and Arrays

MATLAB is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.

All MATLAB variables are multidimensional *arrays*, no matter what type of data. A *matrix* is a two-dimensional array often used for linear algebra.

Array Creation

To create an array with four elements in a single row, separate the elements with either a comma (,) or a space.

```
V = [1 2 3 4]        or        V = [1, 2, 3, 4]
V =
1    2    3   4
```

This type of array is a (*row vector*).

To create a matrix that has multiple rows, separate the rows with semicolons.

```
a = [1 2 3; 4 5 6; 7 8 10]
a =
1    2     3
4    5     6
7    8   10
```

Another way to create a matrix is to use a function, such as ones, zeros, or rand. For example, create a 5-by-1 column vector of zeros.

```
z = zeros (5, 1)
z =
0
0
0
0
0
```

create a 3-by-2 column matrix of ones.

```
 x=ones (3, 2)
x =
1 1
1 1
1 1
```

Vector, Matrix and Array Operations

- **Vectors:**

The vectors is sum of numbers can be a create in one row or one column

```
V= [1, 2, 3, 4]
V=
2   3  4       To add new element to an vector
V=[V(1:4),5]
V=
1   2  3  4  5
V= [-1, V(1:5)]
V=
-1   1  2  3  4  5
V= [ V(1:2) , 10, V(3:4)]
V=
-1  1  10  2  3
```

The operation can be doing on the vector is:

Length: calculate the number of vector elements

V=[2 5 0 1 4 -1]   Length (v)

Ans=

6

Sum: calculate the sum of vector elements   W= Sum (v)

W=

11

Max: to find the maximum number in the vectors   W= max (v)

W=

4

Min: to find the minimum number in the vectors   W= min (v)

W=

-1

5) Sort: To arrange vector elements an ascending order R= [9 7 5 8 3]

S= sort(r)

S=

3 5 7 8 9

Mathematical calculations: (+, -, *, ^)

MATLAB allows you to process all of the values in an array using a single arithmetic operator or function.

```
X= [1, 3, 5];
Y= [2, 4, 6];
Z= x+y
Z=
7  11
M= y-x
M=
1  1
P= x. *y
P=
12  30
P= x.^2
P=
1  9  25
```

Matrices:

The matrix consists of rows and columns (m x n) where m is a columns and n is a rows

```
a =[1,2,3,;4,5,6;7,8,9]
a =
1   2   3
4   5   6
7   8   9
```

To find the second row

```
A(2, :)
Ans=
3 5  6
```

Or find the third columns

```
A(: , 3)
Ans=
3
6
9
```

To delete row or column

```
a(:,2) = [ ]
a =
1 3
4 6
7 9
 a(2,:) = [ ]
a =
1 2 3
7 8 9
```

To find the diameter of the matrix

```
diag(a)
ans=
1
5
9
```

The operation can be doing on the matrix is:

1) Sum:

```
x= [1,2,3;4,5,6;7,8,9]
x =
1 2 3
4 5 6
7 8 9
>> A=sum(x)
A =
12 15 18
```

2) Size: It is a function that displays the dimensions of the array

```
[C,D]=size(x)
C =
    3
D =
3
```

Concatenation

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

```
A = [a, a]

A =

1   2   3    1    2    3

4   5   6    4    5    6

7   8   10   7   8   10
```

UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Assist Lec. Noor Abdul Khaleq

Complex Numbers:

Complex numbers have both real and imaginary parts, where the imaginary unit is the square root of -1.

```
sqrt(-1)
ans =
0.0000 + 1.0000i
```

To represent the imaginary part of complex numbers, use either i or j .

```
>>z=4+6i
z =
  4.0000 + 6.0000i
>> real(z)
ans =
    4
>> imag(z)
ans =
    6
>>c = [3+4i, 4+3j; -I, 10j]
c =
    3.0000 + 4.0000i      4.0000 + 3.0000i
    0.0000 - 1.0000i      0.0000 +10.0000i
```

Array Indexing

Every variable in MATLAB® is an array that can hold many numbers. When you want to access selected elements of an array, use indexing.

For example, consider the 4-by-4 magic square A:

```
A = magic(4)
A =
          16 2 3 13
           5 11 10 8
            9 7 6 12
            4 14 15 1
```

There are two ways to refer to a particular element in an array. The most common way is to specify row and column subscripts, such as

```
A(4,2)
ans =
          14
```

Less common, but sometimes useful, is to use a single subscript that traverses down each column in order:

```
A(8)
ans =
           14
```

Using a single subscript to refer to a particular element in an array is called *linear indexing*.

If you try to refer to elements outside an array on the right side of an assignment statement, MATLAB throws an error.

```
test = A(4,5)
        Index exceeds matrix dimensions.
```

However, on the left side of an assignment statement, you can specify elements outside the current dimensions. The size of the array increases to accommodate the newcomers.

```
A(4,5) = 17
A =
          16 2 3 13 0
           5 11 10 8 0
           9 7 6 12 0
           4 14 15 1 17
```

To refer to multiple elements of an array, use the colon operator, which allows you to specify a range of the form start:end. For example, list the elements in the first three rows and the second column of A:

```
A(1:3,2)
ans =
            2
            11
            7
```

The colon alone, without start or end values, specifies all of the elements in that dimension. For example, select all the columns in the third row of A:

```
A(3,:)
ans =
           9 7 6 12 0
```

The colon operator also allows you to create an equally spaced vector of values using the more general form start:step:end.

```
B = 0:10:100
B =
        0 10 20 30 40 50 60 70 80 90 100
```

If you omit the middle step, as in start:end, MATLAB uses the default step value of 1.

Mathematical functions

MATLAB offers many predefined mathematical functions for technical computing which
contains a large set of mathematical functions. These
functions are called *built-ins*. Many standard mathematical functions, such as sin(*x*), cos(*x*),
tan(*x*), *ex*, *ln*(*x*), are evaluated by the functions sin, cos, tan, exp, and log respectively in
MATLAB.

exp(x)	e^X	**acos(x)**	$\cos^{-1}x$
Sin(x)	Sin x	atan(x)	$\tan^{-1}x$
Cos(x)	Cos x	Log(x)	Ln x
Tan(x)	Tan x	Log10(x)	Log x
asin(x)	$\sin^{-1}x$	Abs(x)	$\|x\|$
		Sqrt(x)	$\sqrt{x}$

Workspace Variables

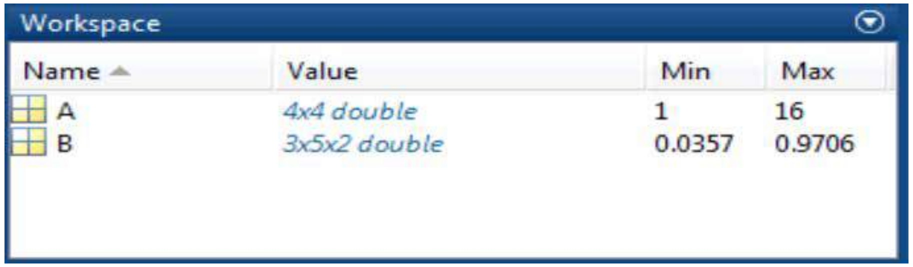
The *workspace* contains variables that you create within or import into MATLAB® from data files or other programs. For example, these statements create variables A and B in the workspace.

```
A = magic(4);
 B = [3,5,2];
```

You can view the contents of the workspace using **whos.**

```
>>whos
    Name      Size      Bytes       Class      Attributes
     A        4x4       128        double
     B        3x5x2      240        double
```

The variables also appear in the Workspace pane on the desktop.



Workspace

Name	Value	Min	Max
A	4x4 double	1	16
B	3x5x2 double	0.0357	0.9706

In order to display a list of the variables currently in the memory used who

```
>>who
    A      B
```

Workspace variables do not persist after you exit MATLAB. Save your data for later use with the save command,

save my file. Mat (any name. Mat)

Saving preserves, the workspace in your current working folder in a compressed file with a.mat extension, called a MAT-file.

To clear all the variables from the workspace, use the clear command.

Restore data from a MAT-file into the workspace using load.

load my file. Mat

Text & Characters

When you are working with text, enclose sequences of characters in single quotes. You can assign text to a variable.

```
>>Text1 = 'Hello, world';
```

If the text includes a single quote, use two single quotes within the definition.

```
>>Text2 = 'You''re right'
Text2 =
        You're right
```

myText and otherText are arrays, like all MATLAB® variables. Their class or data type is char, which is short for character.

```
>>whos Text1
        Name     Size     Bytes     Class     Attributes
        Text1    1x12     24        char
```

You can concatenate character arrays with square brackets, just as you concatenate numeric arrays.

```
>>Text3 = [Text1,' - ',Text2]
  Text3 =
   Hello, world - You're right
```

Calling Functions

MATLAB® provides a large number of functions that perform computational tasks. Functions are equivalent to subroutines or methods in other programming languages.

To call a function, such as max, enclose its input arguments in parentheses:

```
>>A = [1 3 5];
    max(A)
     ans =
              5
```

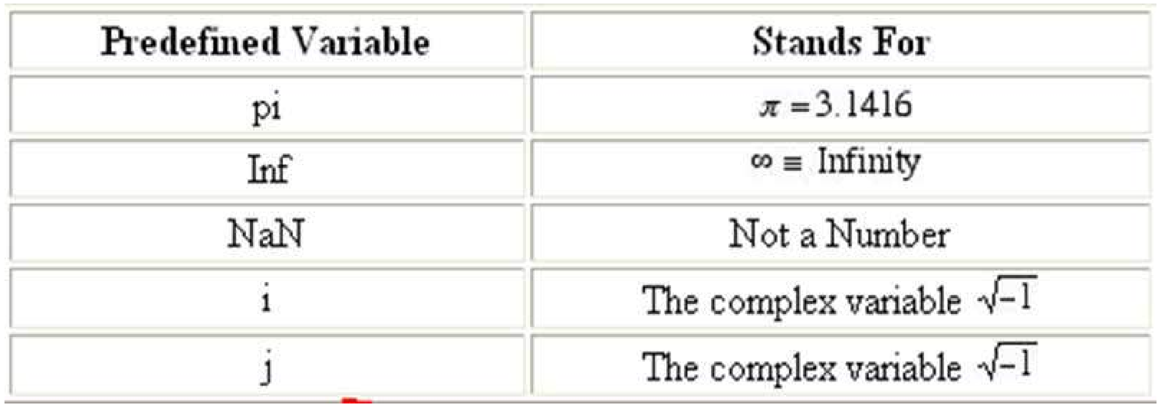
If there are multiple input arguments, separate them with commas:

```
 >>B = [10 6 4];
  >>max(A,B)
       ans =
               10 6 5
```

Return output from a function by assigning it to a variable:

```
    >>maxA = max(A)
          maxA =
                5
```

Special Variable Names and Constants:



Predefined Variable	Stands For
pi	$\pi = 3.1416$
Inf	$\infty \equiv$ Infinity
NaN	Not a Number
i	The complex variable $\sqrt{-1}$
j	The complex variable $\sqrt{-1}$

%The following command will show up the value of (pi)

```
    >>pi
    Ans=
      3.1416
```

%the following process will show the infinity

```
>> 1/0
Warning: Divide by zero
Ans=
    Inf
```

```
% the following command will show not a number
>>0/0
Warning: Divide by zero
Ans=
      NaN

% the following command will show the complex number
>> i
ans =
     0.0000 + 1.0000i
```

Input and output operations:

Input:

```
>>x=input('enter your age:')
enter your age:33
   x=
      33

>>y=input('enter your name: ','s')
enter your name: Mohamed Ali
  y =
  Mohamed Ali
```

Output:

```
>> x=9;
>> disp(x)
9
>> display(x)
x =
9
```

UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Help and Documentation:

Assist Lec. Noor Abdul Khaleq

All MATLAB® functions have supporting documentation that includes examples and describes the function inputs, outputs, and calling syntax. There are several ways to access this information from the command line:

- Open the function documentation in a separate window using the doc command.

```
>>doc mean
```

- To search for a function by specifying keywords:

```
>>doc + Search tab
```

- View an abbreviated text version of the function documentation in the Command Window using the help command.

```
>>help mean
```

Another way to get help is to use the lookfor command. The lookfor command differs from the help command. The help command searches for an exact function name match, while the lookfor command searches the quick summary information in each function for a match. For example, suppose that we were looking for a function to take the inverse of a matrix. Since MATLAB does not have a function named inverse, the command help inverse will produce nothing. On the other hand, the command lookfor inverse will
produce detailed information, which includes the function of interest, inv.

```
>> lookfor inverse
```

The general form is

```
>> lookfor FunctionName
```

Note: At this particular time of our study, it is important to emphasize one main point. Because MATLAB is a huge program; it is impossible to cover all the details of each function one by one. However, we will give you information how to get help. Here are some examples:

To check how the help, command works:

```
>> help help
```

To check toolboxes available:

```
>> help
```

To check functions available in the communications toolbox:

```
>> help comm
```

To check functions available in the signal processing toolbox:

```
>> help signal
```

To check how the function fft(discrete Fourier transform) works:

```
>> help fft
```

Type help helptools for more details.

```
>> help helptools
```

Scripts and Functions – M Files

All the commands were executed in the Command Window.
The problem is that the commands entered in the Command Window cannot be saved and executed again for several times. Therefore, a different way of executing repeatedly commands with MATLAB is:

1. to *create* a file with a list of commands,
2. *save* the file under a name that ends in ".m". and
3. *run* the file.

If needed, corrections or changes can be made to the commands in the file. The files that are used for this purpose are called script files or *scripts* for short.

M-File Scripts:

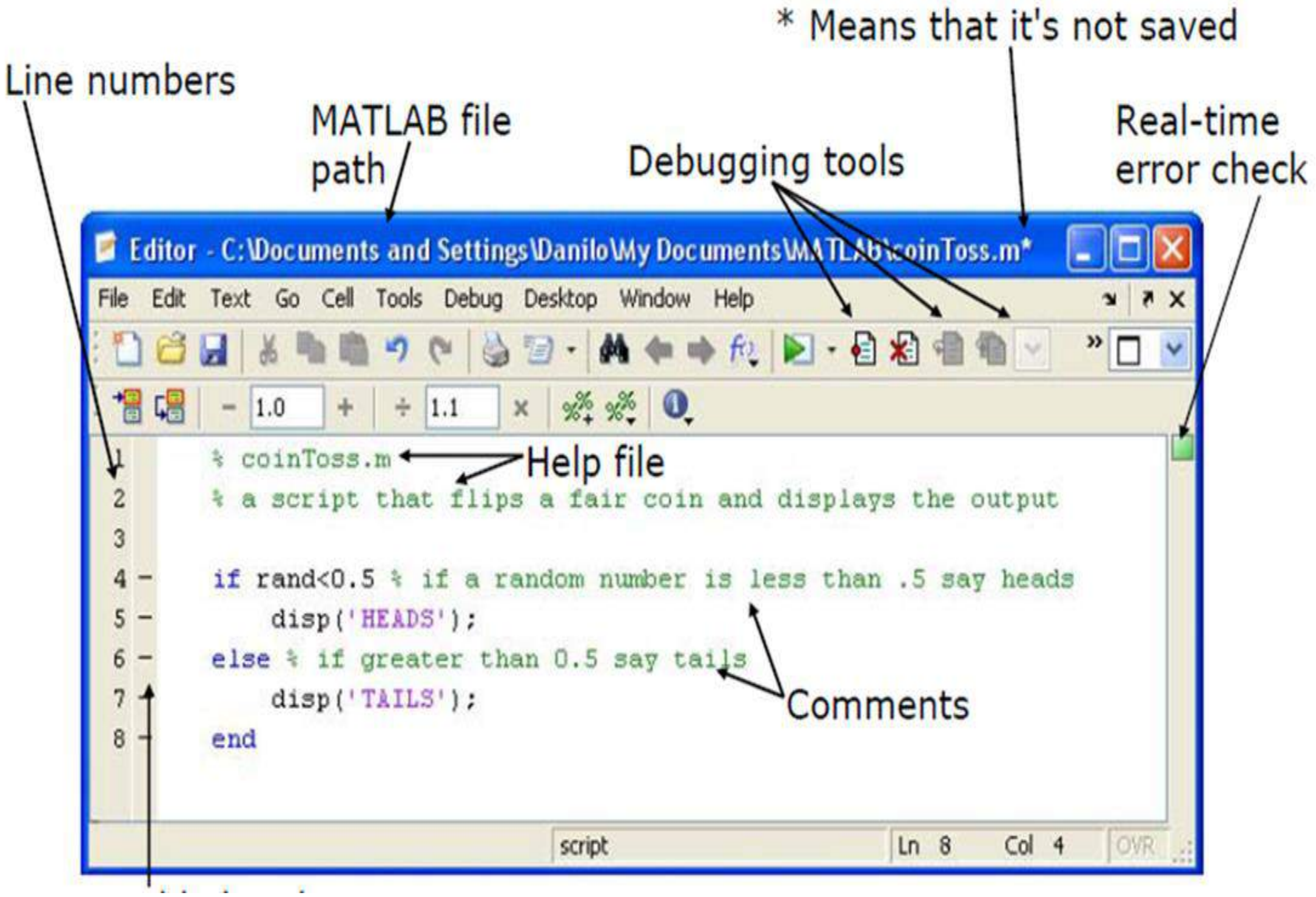
A script file is an external file that contains a sequence of MATLAB statements. Script files have a filename extension .m and are often called M-files. M-files can be scripts that simply execute a series of MATLAB statements, or they can be functions that can accept arguments and can produce one or more outputs.

Notes!

Anything following a **%** is seen as a comment

Note that scripts are somewhat static, since there is no input and no explicit output

All variables created and modified in a script exist in the workspace even after it has stopped running


* Means that it's not saved
Line numbers
MATLAB file path
Debugging tools
Real-time error check
Help file
Comments
Editor - C:\Documents and Settings\Danilo\My Documents\MATLAB\coinToss.m*
File Edit Text Go Cell Tools Debug Desktop Window Help
script
Ln 8
Col 4
OVR

```
% coinToss.m
% a script that flips a fair coin and displays the output

if rand<0.5 % if a random number is less than .5 say heads
    disp('HEADS');
else % if greater than 0.5 say tails
    disp('TAILS');
end
```

Example:

Enter the following statements in the file:

```
A = [1 2 3; 3 3 4; 2 3 3];
b = [1; 1; 2];
x = A\b
```

Save the file, for example, example1.m.

Run the file, in the command line, by typing:

```
>> example1
x =
-0.5000
1.5000
-0.5000
```

When execution completes, the variables (A, b, and x) remain in the workspace.

Script side-effects

All variables created in a script file are added to the workspace. This may have undesirable

effects, because:

- Variables already existing in the workspace may be overwritten.
- The execution of the script can be affected by the state variables in the workspace.

As a result, because scripts have some undesirable side-effects, it is better to code any complicated applications using rather function M-file.

M-File Functions

Functions are programs (or routines) that accept input arguments and return output arguments. Each M-file function (or function or M-file for short) has its own area of workspace, separated from the MATLAB base workspace.

In addition, it is important to note that function name must begin with a letter, and must be no longer than the maximum of 63 characters. Furthermore, the name of the text file that you save will consist of the function name with the extension .m.

Example:

```
function f = factorial(n)
% FACTORIAL(N) returns the factorial of N.
% Compute a factorial value.
f = prod(1:n);
```

The first line of a function M-file starts with the keyword function. It gives the function name and order of arguments. In the case of function factorial, there are up to one output argument and one input argument. Table below summarizes the M-file function.

As an example, for n = 5, the result is,

```
>> f = factorial(5)
f =
120
```

Input and output arguments

The input arguments are listed inside parentheses following the function name. The output arguments are listed inside the brackets on the left side. They are used to transfer the output from the function file. The general form looks like this

function [outputs] = function_name(inputs)

Function file can have none, one, or several output arguments.

Example:

function C=FtoC(F)   (One input argument and one output argument)

function area=TrapArea(a,b,h)   (Three inputs and one output)

Input to a script file

When a script file is executed, the variables that are used in the calculations within the file must have assigned values. The assignment of a value to a variable can be done in three ways.

1. The variable is defined in the script file.

2. The variable is defined in the command prompt.

3. The variable is entered when the script is executed.

In this case, the variable is defined in the script file. When the file is executed, the user is

prompted to assign a value to the variable in the command prompt. This is done by using

the input command. Here is an example.

This script file calculates the average of points scored in three games.

The point from each game are assigned to a variable by using the `input' command.

```
game1 = input('Enter the points scored in the first game ');
game2 = input('Enter the points scored in the second game ');
game3 = input('Enter the points scored in the third game ');
average = (game1+game2+game3)/3
```

Output commands

As discussed before, MATLAB automatically generates a display when commands are executed. In addition to this automatic display, MATLAB has several commands that can be used to generate displays or outputs.

Two commands that are frequently used to generate output are: disp and fprintf.

The main differences between these two commands can be summarized as follows:

Disp:

- Simple to use.
- Provide limited control over the appearance of output

fprintf:

- Slightly more complicated than disp.
- Provide total control over the appearance of output

Digital Communication Lab/ Four Class

UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Control flow and operators

Assist Lec. Noor Abdul Khaleq

MATLAB is also a programming language. Like other computer programming languages, MATLAB has some decision making structures for control of command execution. These decisions making or control flow structures include for loops, while loops, and if-else-end constructions. Control flow structures are often used in script M-files and function M-files. By creating a file with the extension .m, we can easily write and run programs.
MATLAB has four control flow structures: the if statement, the for loop, the while loop, and the switch statement.

The ``if...end'' structure

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 simplest form of the if statement is

```
if expression
statements
end
```

example:

```
n=input ('enter the number');
if n > 4
M = eye(n)
elseif n < 4
M = zeros(n)
else
M = ones(n)
end
```

example2:

```
discr = b*b - 4*a*c;
if discr < 0
disp('Warning: discriminant is negative, roots are imaginary');
else
disp('Roots are real, but may be repeated')
end
```

example3:

```
str='demo';
 password=input('Enter password:','s');
 if strcmpi(str,password)==1
     disp('password is right!');
 end
```

Relational and logical operators

A relational operator compares two numbers by determining whether a comparison is *true*
or *false*. Relational operators are shown in Table
Note that the equal to" relational operator consists of two equal signs (==) (with no space
between them), since = is reserved for the *assignment* operator.

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
~=	Not equal to
&	AND operator
|	OR operator
~	NOT operator
xor()	XOR, Logical EXCLUSIVE OR

The ``for...end'' loop

In the for ... end loop, the execution of a command is repeated at a fixed and predetermined number of times. The syntax is

```
for variable = expression
statements
end
```

Usually, expression is a vector of the form i:s:j. A simple example of for loop is

```
for ii=1:5
x=ii*ii
end
```

It is a good idea to indent the loops for readability, especially when they are nested. Note that MATLAB editor does it automatically. Multiple for loops can be nested. The following statements form the 5-by-5 symmetric matrix A with (*i; j*) element *i=j* for *j >= i*:

example:

```
n = 5; A = eye(n);
for j=2:n
for i=1:j-1
A(i,j)=i/j;
A(j,i)=i/j;
end
end
```

example2:

```
m=5
for n = 1:m
r(n) = rank(magic(n));
end
r
```

The ``while...end'' loop

This loop is used when the number of passes is not specified. The looping continues until a stated condition is satisfied. The while loop has the form:

```
while expression
statements
end
```

The statements are executed as long as expression is true.

Example:

```
x = 1
while x <= 10
x = 3*x
end
```

example:

```
m=5;
while m > 1
m = m - 1;
zeros(m)
end
```

Switch and Case Statement

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. There must always be an end to match the switch.

Example:

```
n=2
switch(n)
case 1
M = eye(n)
case 2
M = zeros(n)
case 3
M = ones(n)
End
```

Saving output to a file

In addition to displaying output on the screen, the command fprintf can be used for writing the output to a file. The saved data can subsequently be used by MATLAB or other softwares. To save the results of some computation to a file in a text format requires the following

steps:

1. Open a file using fopen
2. Write the output using fprintf
3. Close the file using fclose

Here is an example (script) of its use.

```
% write some variable length strings to a file
op = fopen('weekdays.txt','wt');
fprintf(op,'Sunday\nMonday\nTuesday\nWednesday\n');
fprintf(op,'Thursday\nFriday\nSaturday\n');
fclose(op);
```

This file (weekdays.txt) can be opened with any program that can read .txt file.

UNVERSITY OF TECHNOLOGY

Computer Engineering Department

2-D and 3-D Plots

Assist Lec. Noor Abdul Khaleq

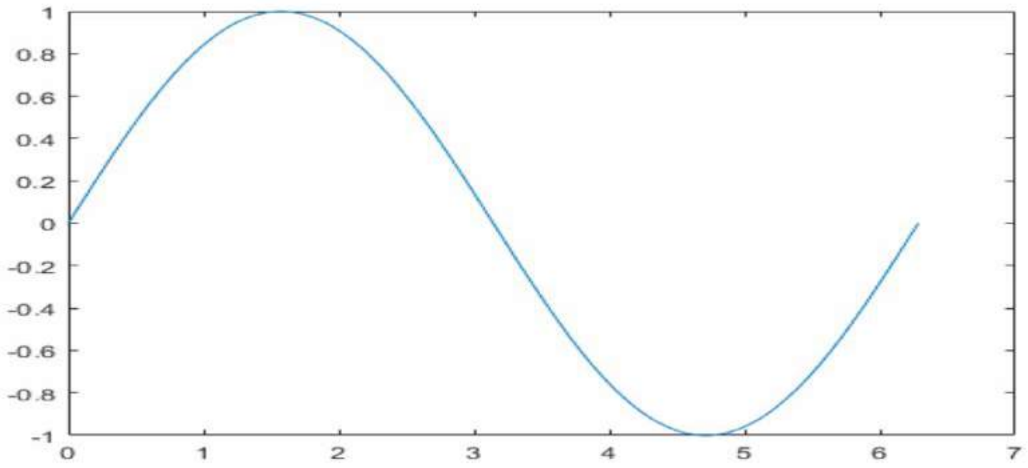
Creating simple plots

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of x- coordinates,
x = (x1, …….. , xN), and a vector of y-coordinates, y = (y, ……. , yN), locate the points (xi; yi), with i = 1; 2,…., n and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the same length. Use the plot function. Function plot can be used to produce a graph from two vectors x and y. For example, The MATLAB command to plot a graph is plot(x,y). The vectors x = (1, 2, 3, 4, 5, 6) and y = (3,-1, 2, 4, 5, 1)

```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```

Note: The plot functions have different forms depending on the input arguments. If y is a vector plot(y)produces a piecewise linear graph of the elements of y versus the index of the elements of y. If we specify two vectors, as mentioned above, plot(x,y) produces a graph of y versus x. For example, to plot the function sin (x) on the interval [0; 2pi], we first create a vector of x values ranging from 0 to 2pi, then compute the sine of these values, and finally plot the Result

```
X=0:pi/100:2*pi;
Y=sin(x);
Plot(x,y)
```

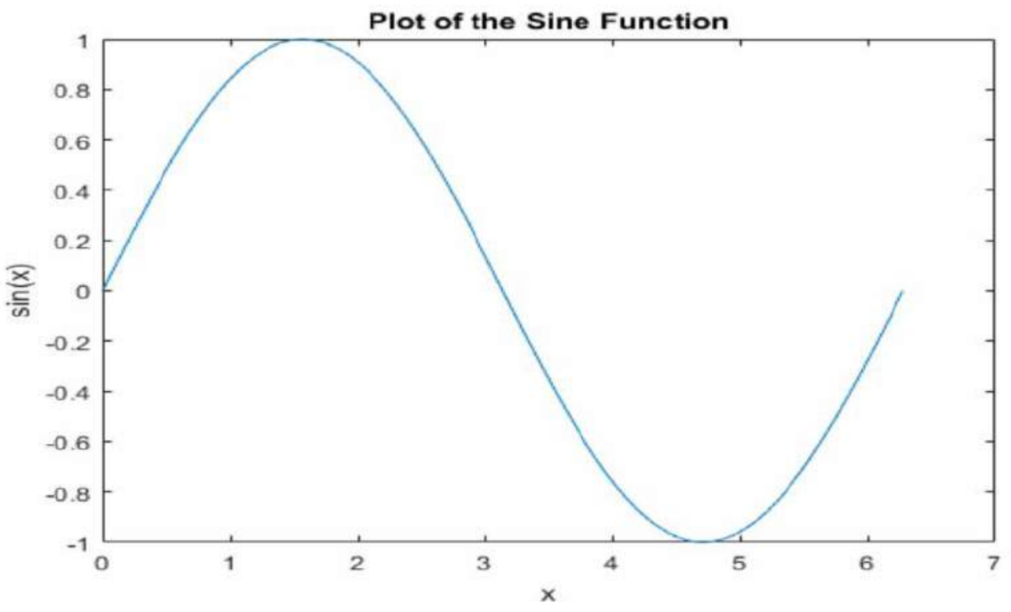


Adding titles, axis labels, and annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the

previous example, add an x- and y-axis labels.

```
xlabel('x')
ylabel('sin(x)')
title('Plot of the Sine Function’)
```



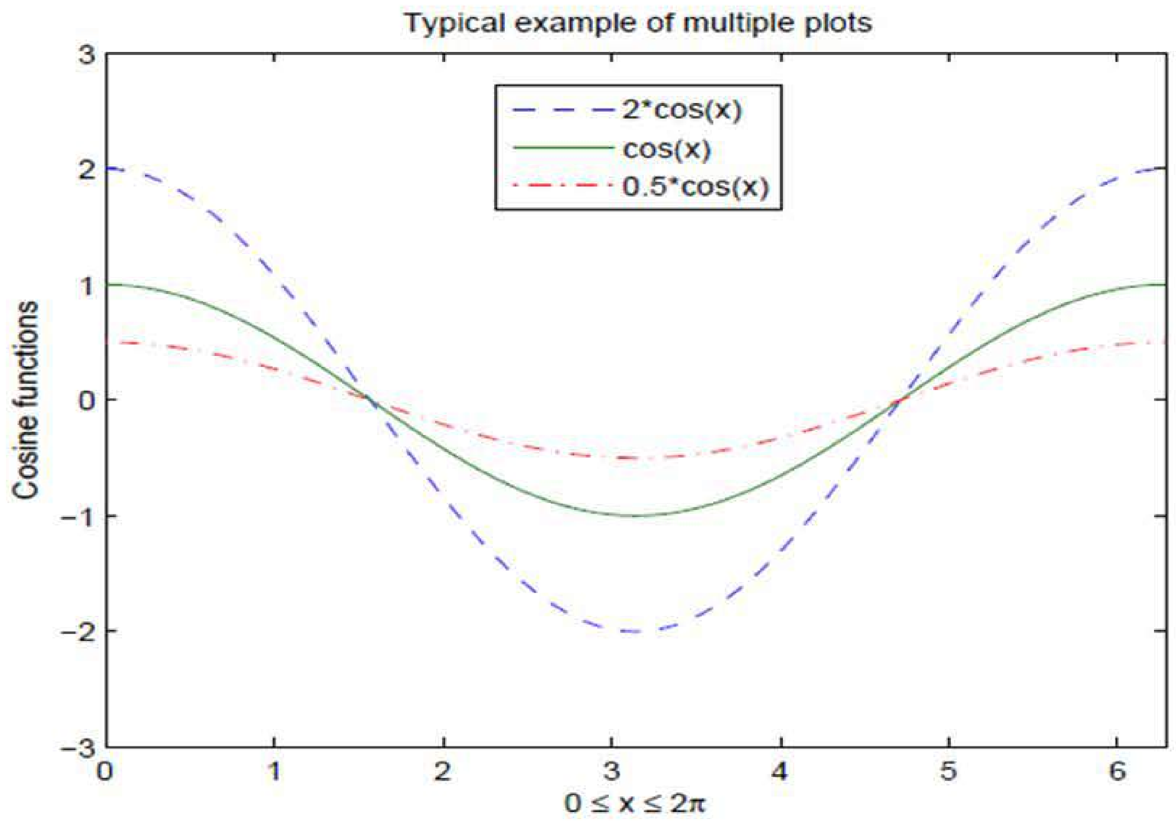
Multiple data sets in one plot

Multiple (x; y) pairs arguments create multiple graphs with a single call to plot. For example,
these statements plot three related functions of x: y1 = 2 cos(x), y2 = cos(x), and y3 =
0:5 * cos(x), in the interval 0 <= x =<2pi.

```
x = 0:pi/100:2*pi;
 y1 = 2*cos(x);
y2 = cos(x);
 y3 = 0.5*cos(x);
plot(x,y1,'--',x,y2,'-',x,y3,':')
xlabel('0 \leq x \leq 2\pi')
ylabel('Cosine functions')
legend('2*cos(x)','cos(x)','0.5*cos(x)')
title('Typical example of multiple plots')
```

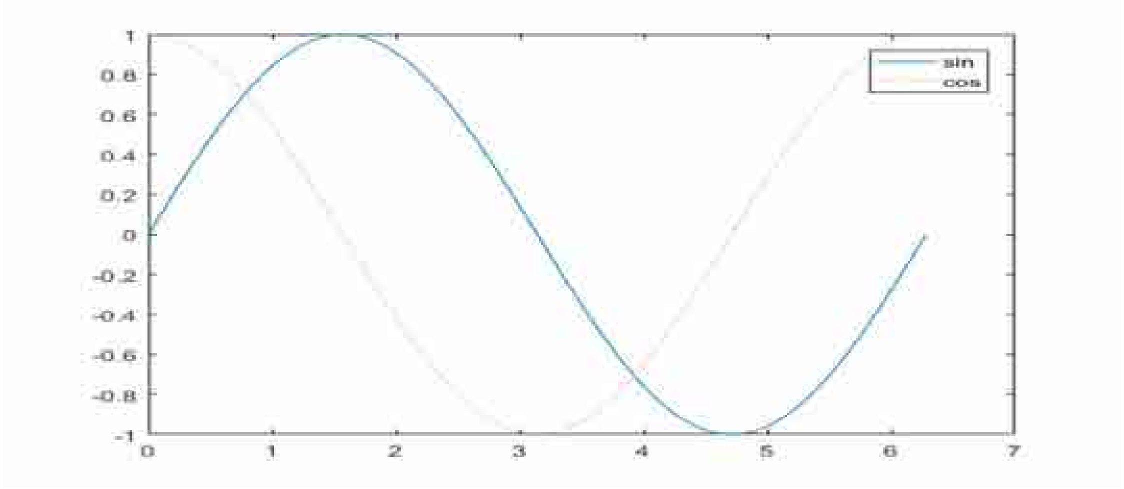
can add the number the axis you needed

```
>> axis([0 2*pi -3 3])
```


Typical example of multiple plots
2*cos(x)
cos(x)
0.5*cos(x)
Cosine functions
$0 \le x \le 2\pi$
3
2
1
0
−1
−2
−3
0
1
2
3
4
5
6

To add plots to an existing figure, use hold.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
hold on
y2 = cos(x);
plot(x,y2,':')
legend('sin','cos')
```



Specifying line styles and colors

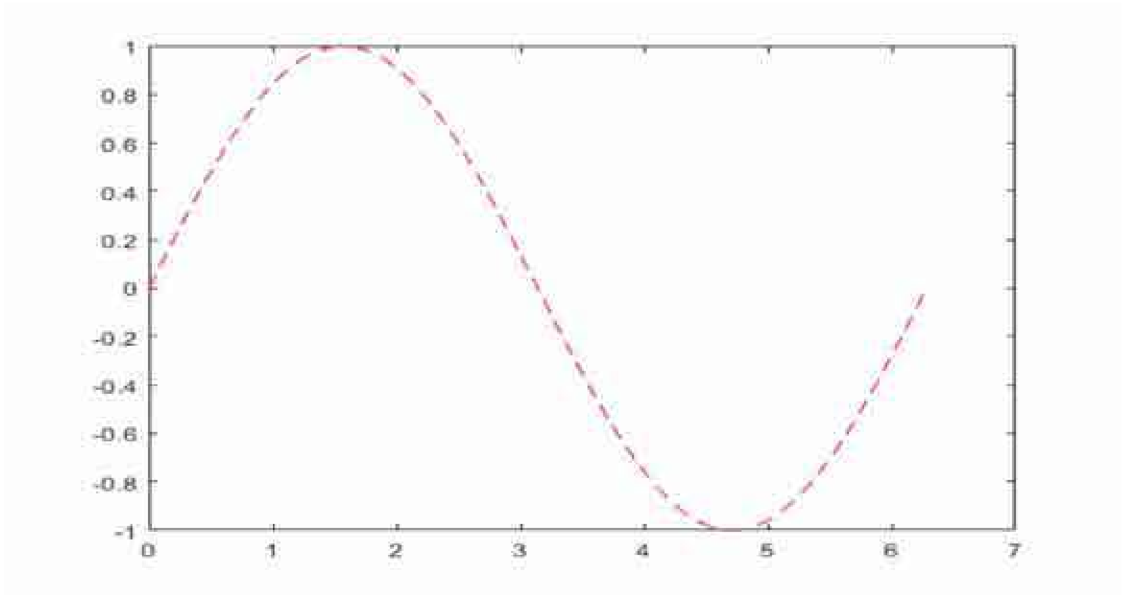
It is possible to specify line styles, colors, and markers using the plot command:

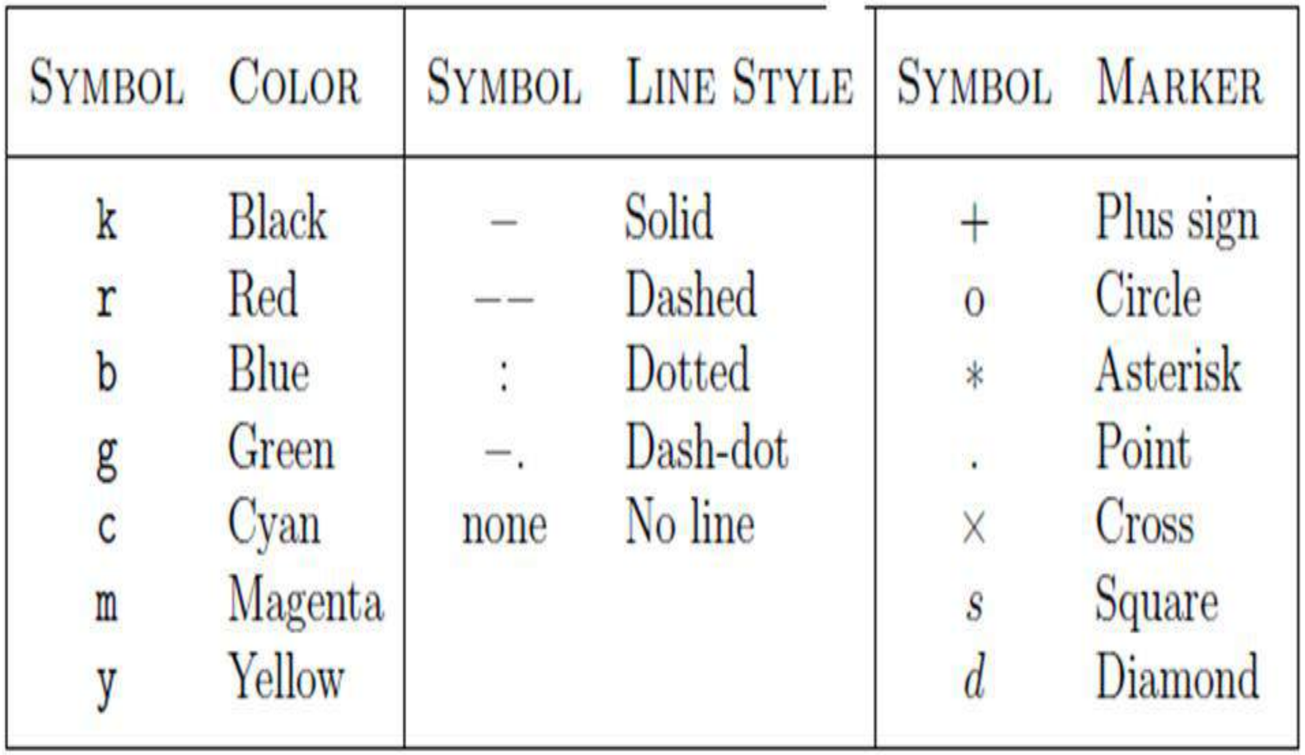
plot(x,y,'style_color_marker')

where style_color_marker is a triplet of values from Table. To find additional information, type help plot or doc plot.

plot(x,y,'r--')

plot(x,y2,':’)





SYMBOL	COLOR	SYMBOL	LINE STYLE	SYMBOL	MARKER
k	Black	–	Solid	+	Plus sign
r	Red	––	Dashed	o	Circle
b	Blue	:	Dotted	*	Asterisk
g	Green	–.	Dash-dot	.	Point
c	Cyan	none	No line	×	Cross
m	Magenta			*s*	Square
y	Yellow			*d*	Diamond

Digital Communication Lab/ Four Class

UNVERSITY OF TECHNOLOGY

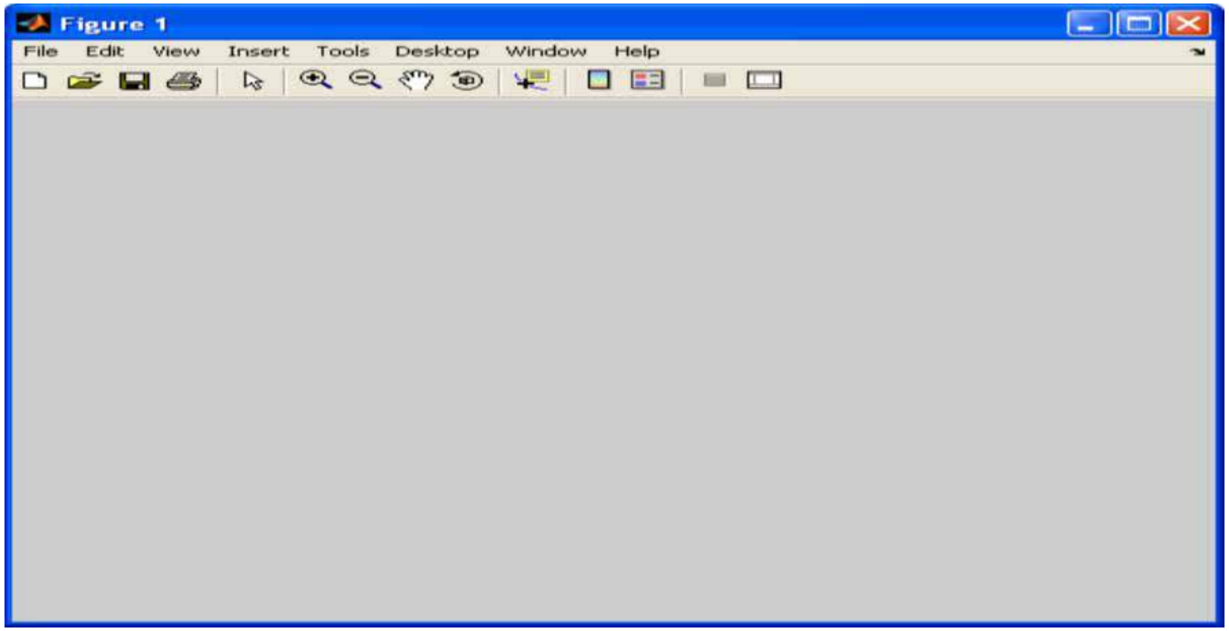
Computer Engineering Department

2-D and 3-D Plots

Assist Lec. Noor Abdul Khaleq

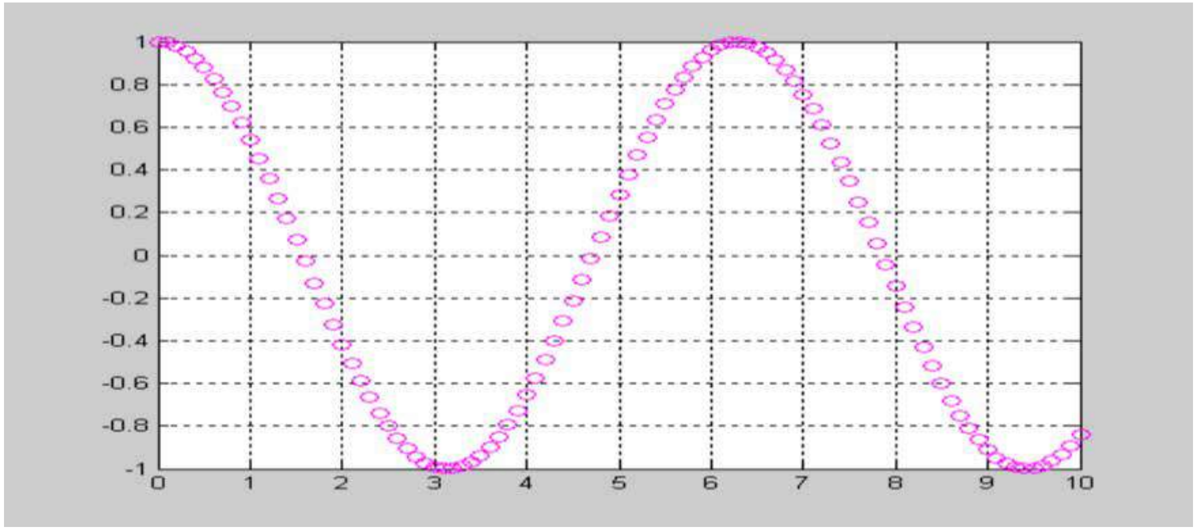
Put the plot in different window

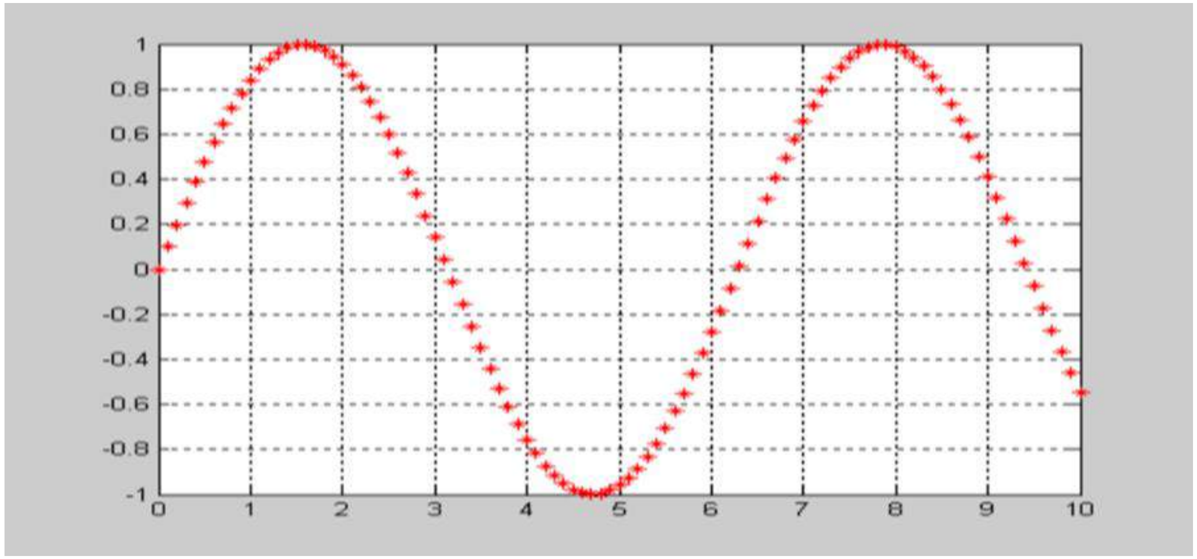
We note to put more than plot in one window we use the operation (hold on, hold off) if we need the result of more than one plot in same time put in more than window use the operation (figure). The operation figure opens null window, see example.



Example:

```
Clc
Clear
Close all
X=0:0.1:10;
Y=sin(X)
Z=cos(X)
Plot(X,Y,’r*’);
Grid
figure
Plot(X,Z,’mo’);
Grid
```


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7
8
9
10

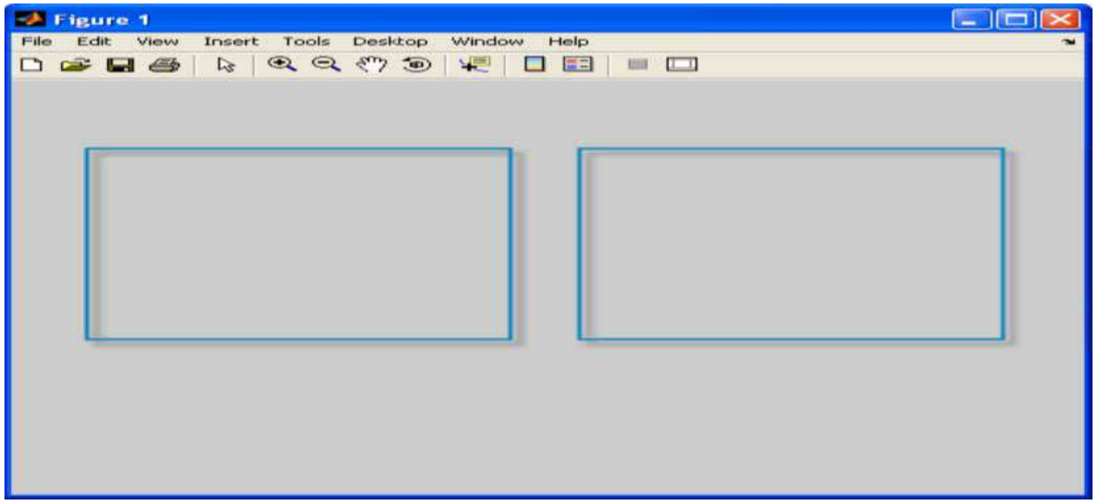

1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7
8
9
10

Subplots

You can display multiple plots in different sub regions of the same window using the subplot function. The first two inputs to subplot indicate the number of plots in each row and column. The third input specifies which plot is active. For example, create four plots in a 2-by-2 grid within a figure window.

plot(number of rows, number of column, the number of the matrix which occupy the figure)

Subplot (1,2,1)



Example:

```
Clc
Clear
Close all
X=0:0.1:10;
Y=sin(X)
Z=cos(X)
Subplot(1,2,1)
Plot(X,Y,’r*’);
Grid
Subplot(1,2,2)
Plot(X,Z,’mo’);
Grid
```

Note: must use the operation plot after subplot

The Result:

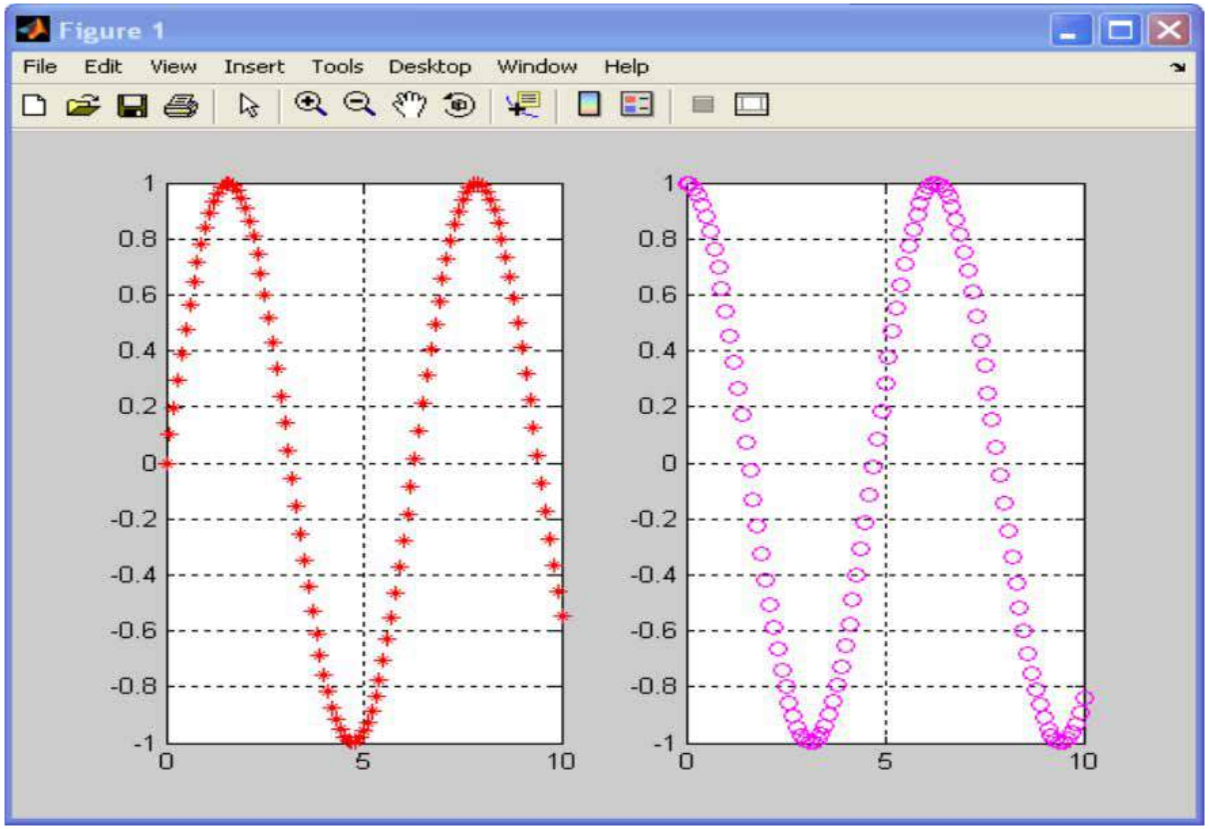
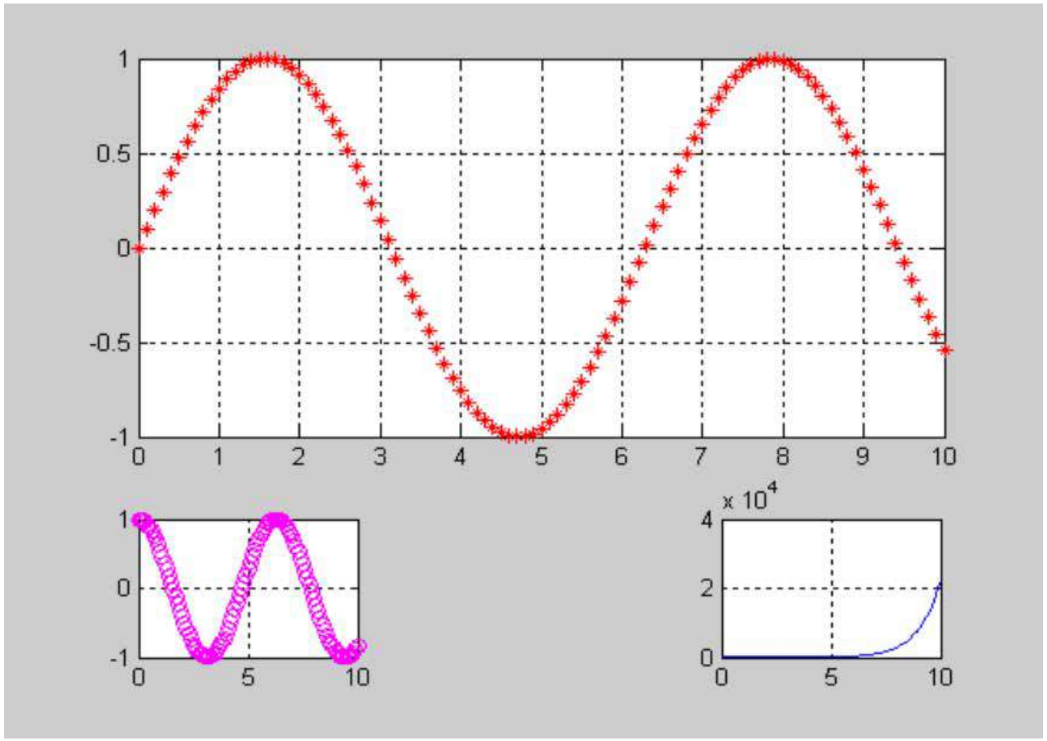


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
5
10

Note: If the plot take more than one subplot in the figure we use the [ ] and write the number inter it.

Example:

```
Close all
X=0:0.1:10;
Y=sin(X)
Z=cos(X)
V=exp(X);
Subplot(3,3,[1 2 3 4 5 6])
Plot(X,Y,'r*');
Grid
Subplot(3,3,7)
Plot(X,Z,'mo');
Grid
Subplot(3,3,9)
Plot(X,V);  grid
```



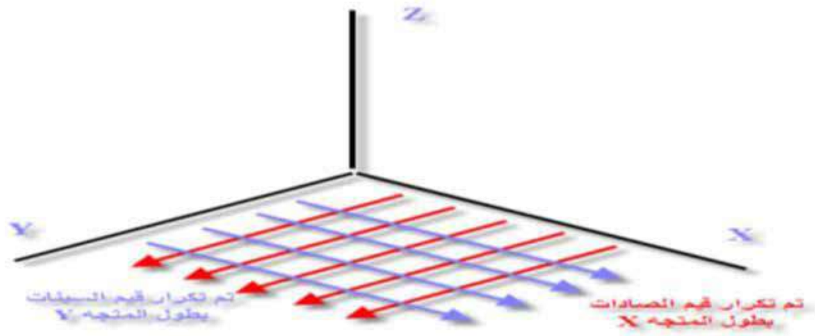
3-D Plots

As we learned that the three-dimensional drawing depends on three axes to draw, axis x, y, z and that both (x, y) represent the horizontal level (Z) is the height, but these values are the values of the points in the axes, but until any point is drawn at the horizontal level, we must define this for the MATLAB using a meshgrid where the MATLAB produces a matrix that is repeated. The x-Axis axis, with the same length as the y-Axis antagonist, replicates the values of the x-axis antibody axis with the same length as the x-Axis values, so that the matrix formed is the horizontal plane.

The form of operation meshgrid is:

```
[x,y]=meshgrid(x,y)
```

After that we use the operation mesh in 3-D instead of plot in 2-D.



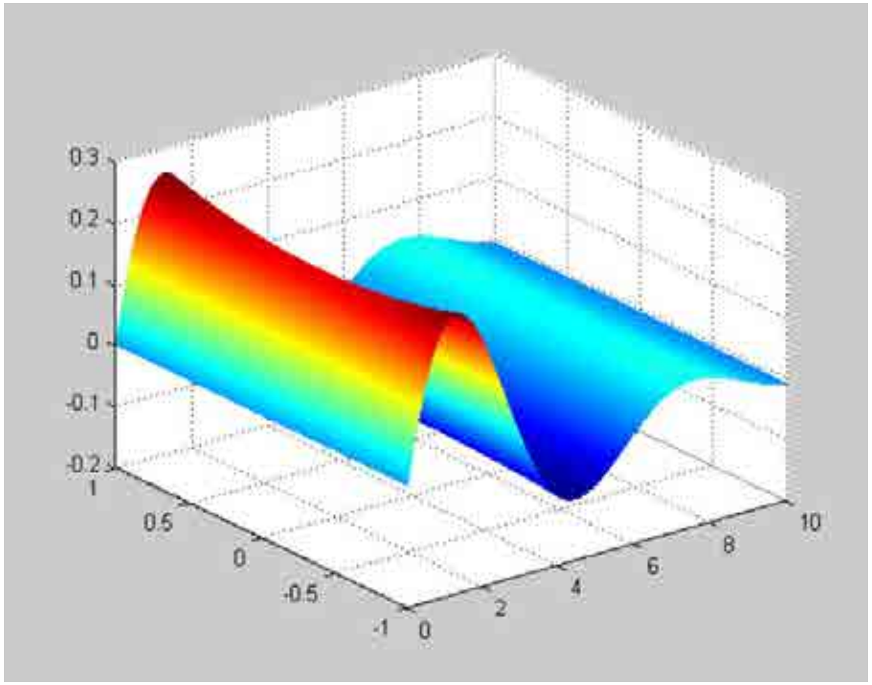
Three-dimensional plots typically display a surface defined by a function in two variables, z = f(x,y) . To evaluate z, first create a set of (x, y) points over the domain of the function using meshgrid.

Note: (Three-dimensional graphics can be produced using the functions surf, plot3 or mesh).

Example:

```
x=linspace(0,10,1000);
y=sin(x);
[x,y] = meshgrid(x,y);
z=sin(x) .* exp(-0.3*x)./ (cos(y)+2);
mesh(x,y,z)
```

.



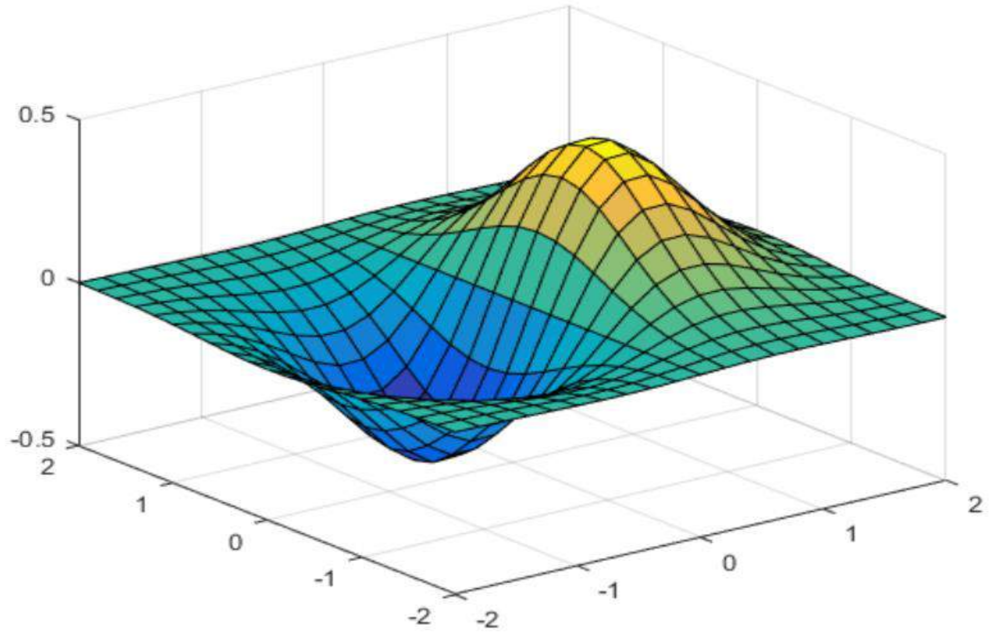
Example:

```
[X,Y] = meshgrid(-2:0.2:2);
Z = X .*exp(-X.^2 - Y.^2);
```

Then, create a surface plot.

```
surf(X,Y,Z)
```

.



Both the surf function and its companion mesh display surfaces in three dimensions. surf displays both the connecting lines and the faces of the surface in color. mesh produces wireframe surfaces that color only the lines connecting the defining points.

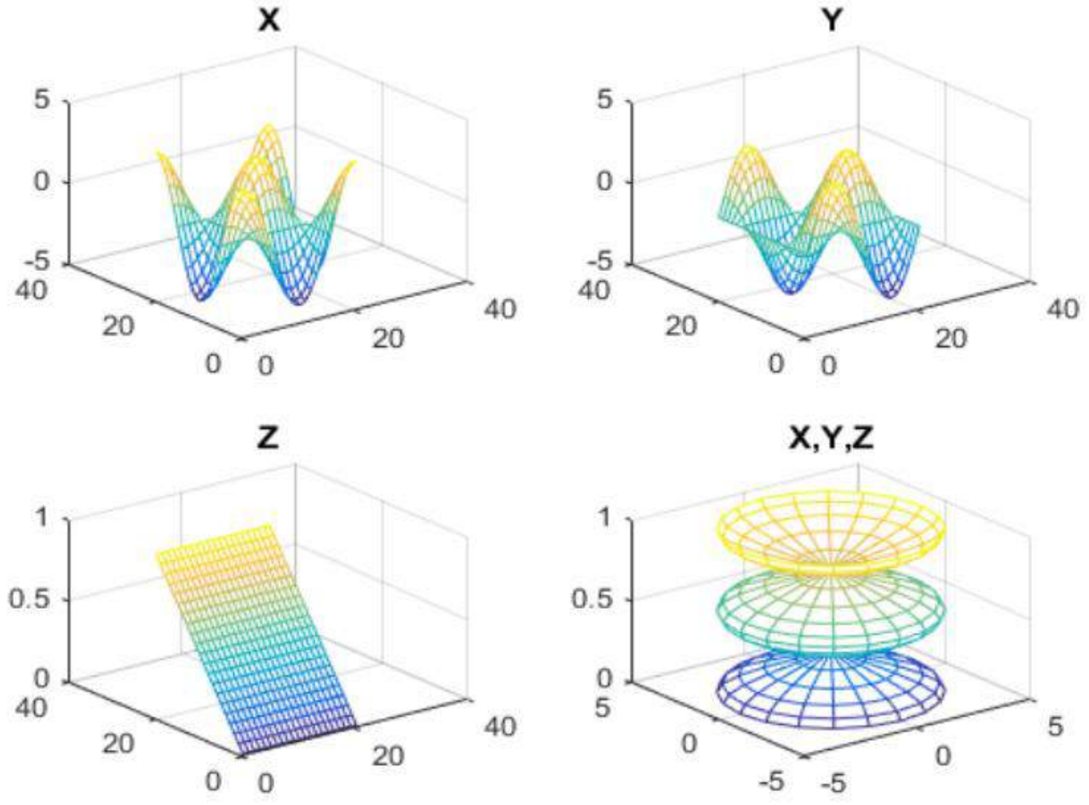
Subplot example:

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X); title('X');
subplot(2,2,2); mesh(Y); title('Y');
subplot(2,2,3); mesh(Z); title('Z');
subplot(2,2,4); mesh(X,Y,Z); title('X,Y,Z');
```

.

Digital Communication Lab/ Four Class

•



UNVERSITY OF TECHNOLOGY

Computer Engineering Department

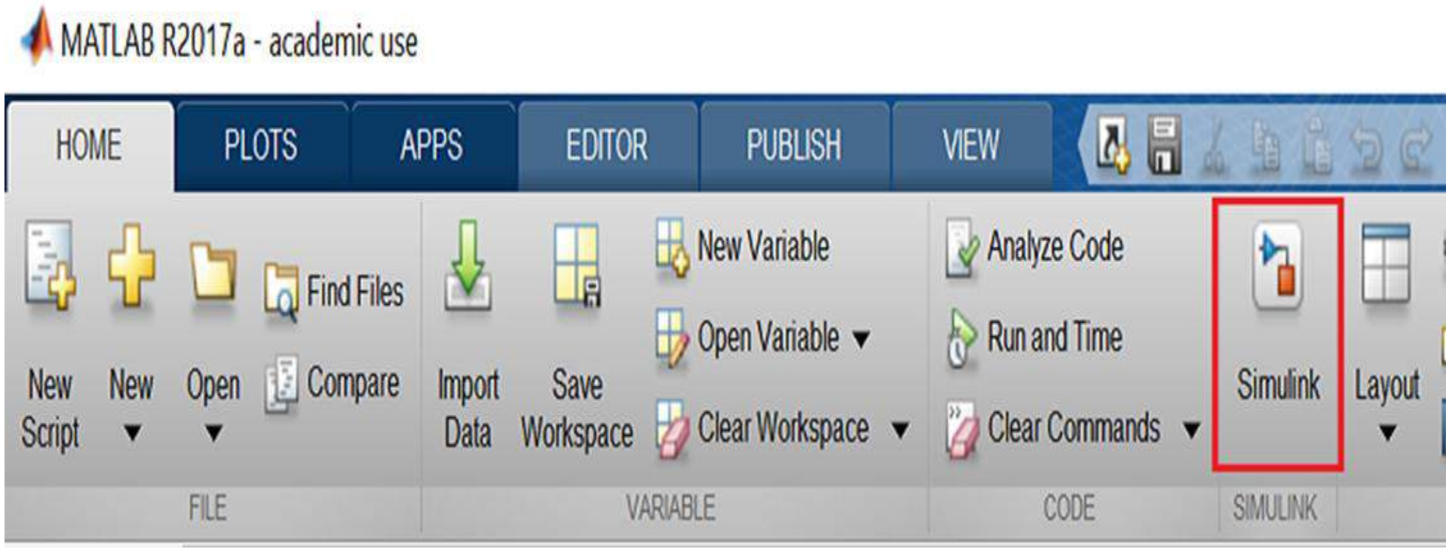
Introduction to Simulink

Assist Lec. Noor Abdul Khaleq

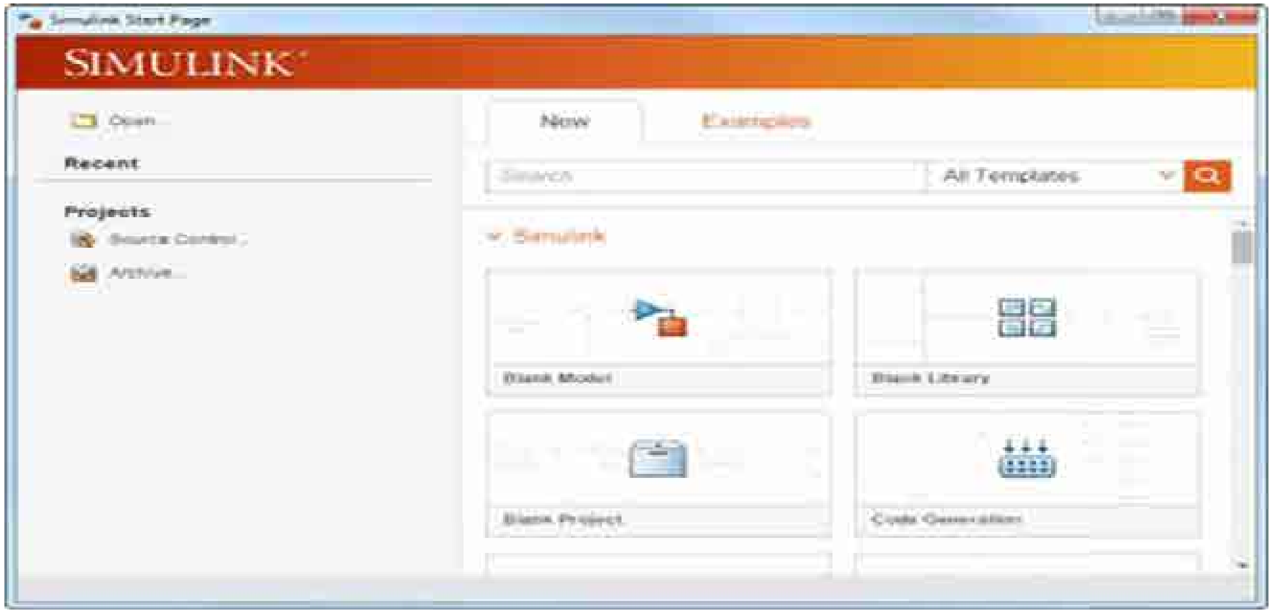
Simulink is an environment for simulation and model-based design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing.

Or Simulink is a graphical extension to MATLAB for the modeling and simulation of systems. In Simulink, systems are drawn on screen as block diagrams. Many elements of block diagrams are available (such as transfer functions, summing junctions, etc.), as well as virtual input devices (such as function generators) and output devices (such as oscilloscopes). Simulink is integrated with MATLAB and data can be easily transferred between the programs. In this tutorial, we will introduce the basics of using Simulink to model and simulate a system.

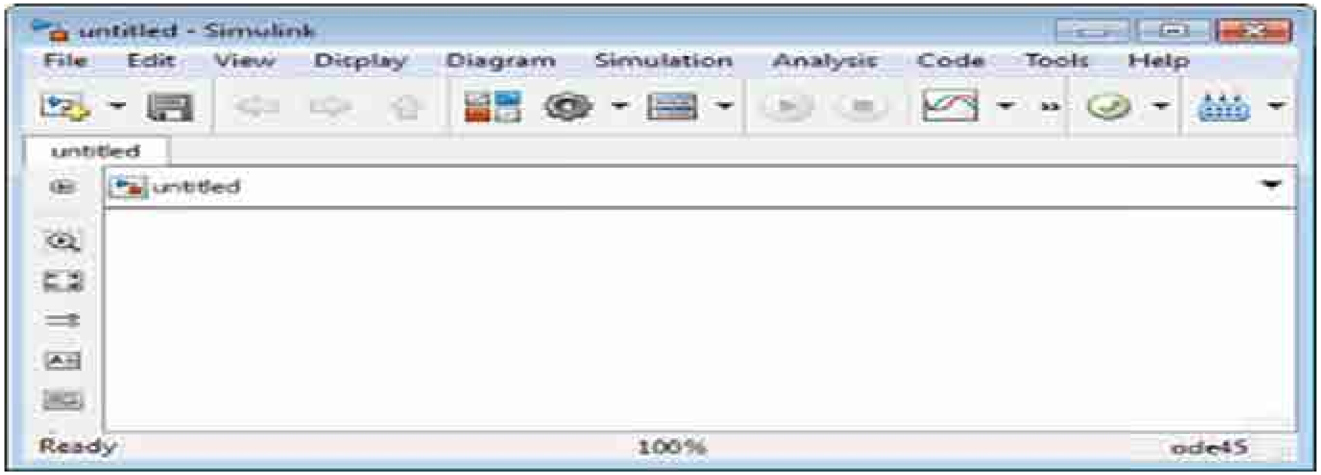
You start Simulink from the MATLAB IDE:
Open MATLAB and select the Simulink icon in the Toolbar Or type "simulink" in the Command window.



After that select the Blank Model template

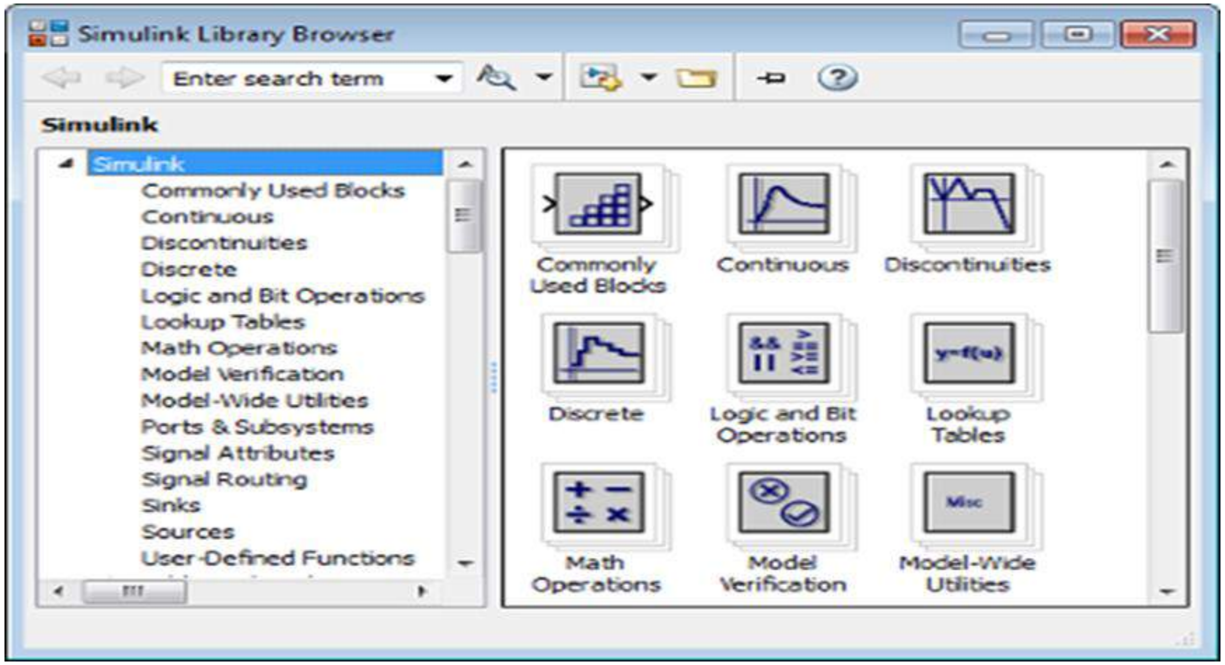


The Simulink Editor opens with a new block diagram.



From the File menu, select Save as. In the File name text box, enter a name for your model, For example, simple-model. Click Save. The model is saved with the file extension (.slx).

From the Simulink Editor toolbar, click the Library Browser button
Open this window



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:

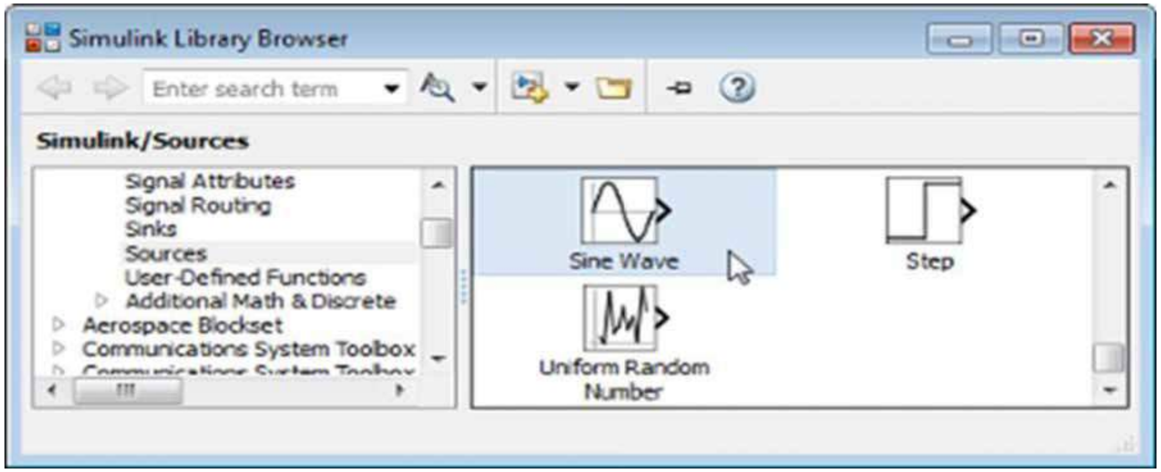
- Sources: used to generate various signals
- Sinks: used to output or display signals
- Continuous: continuous-time system elements (transfer functions, state-space models, PID controllers, etc.)
- Discrete: linear, discrete-time system elements (discrete transfer functions, discrete state-space models, etc.)
- Math Operations: contains many common math operations (gain, sum, product, absolute value, etc.)
- Ports & Subsystems: contains useful blocks to build a system

Create a new Model

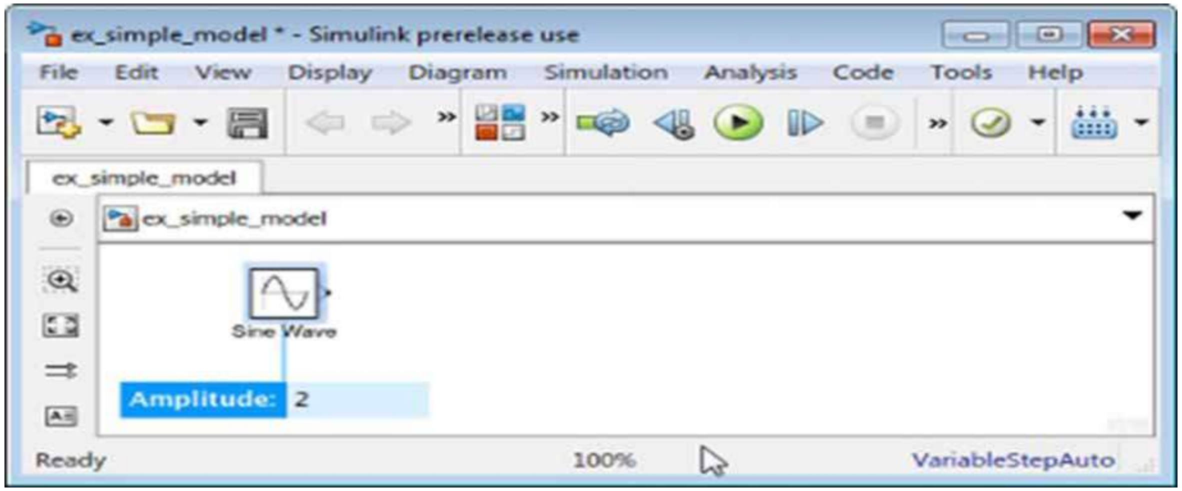
You build models by dragging blocks from the Simulink Library Browser window to the Simulink Editor or single-clicking your model and entering a search term.

Example:

Drag the Sine Wave block to the Simulink Editor.



A copy of the Sine Wave block appears in your model with a text box for entering the value of the Amplitude parameter. In the text box enter 2.


ex_simple_model * - Simulink prerelease use
File Edit View Display Diagram Simulation Analysis Code Tools Help
ex_simple_model
ex_simple_model
Sine Wave
Ready
100%
VariableStepAuto

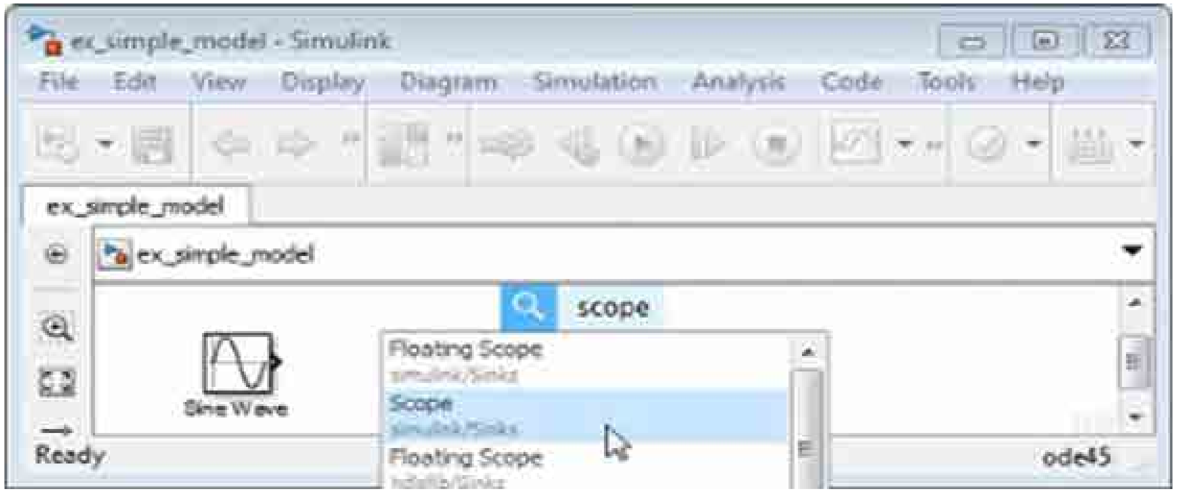
Amplitude: 2

Add a Scope block using this alternative procedure:

a. Double-click within the block diagram.

b. After the search icon appears, type scope, and then from the list, select Scope from the Simulink/Sinks library.

.



Block Connections

Before you connect the blocks in your model, arrange the block inputs and outputs to make signal connections as straightforward as possible

Most blocks have angle brackets on one or both sides. Inputs are located on the left side of the blocks, while outputs are located on the right side of the blocks. The angle brackets represent input and output ports:

- The > symbol pointing into a block is an input port.
- The > symbol pointing out of a block is an output port.

.

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

symbol.

Use the mouse to wire the inputs and outputs of the different blocks by drawing lines between output ports and input ports.

In example Position the cursor over the output port on the right side of the Sine Wave block. The pointer changes to a cross hair (+).

Click, and then drag a line from the output port to the top input port of the Gain block

Click and drag the output port from the Gain block to the Scope block

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

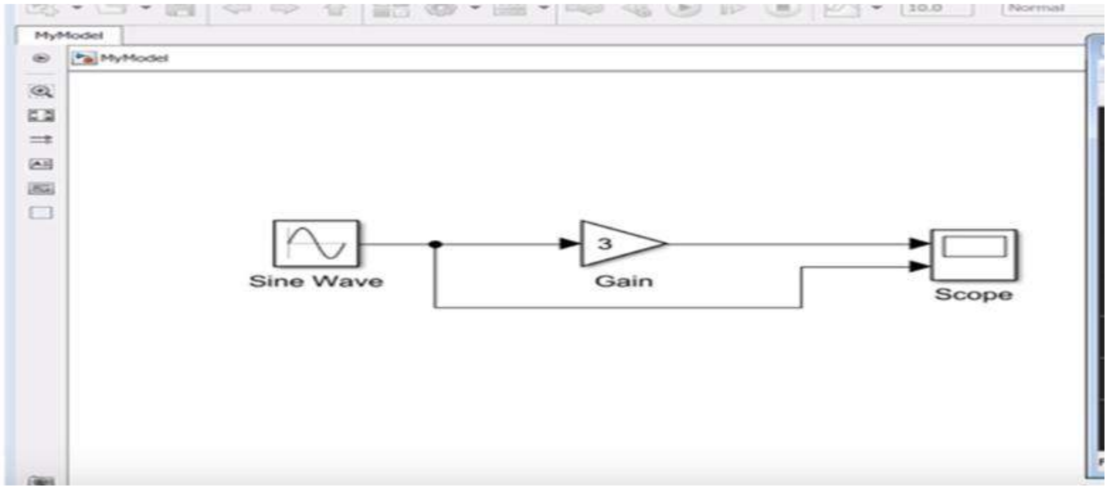
Your simple model is almost complete. To finish the model, connect the Sine Wave block to the scope block. This connection is different from the other connections,

.

which all connect output ports to input ports.

1. Position the cursor where you want to start a branch line. In this example, position the cursor over the signal line between the Sine Wave and scope blocks.
2. Right-click and drag the cursor away from the line to form a dotted line segment.
3. Continue to drag the cursor to the scope input port, and then release the mouse button.
4. The new line, called a branch line, carries the same signal that passes from the Sine Wave block to the scope block. Your model is now complete similar to the following figure

.



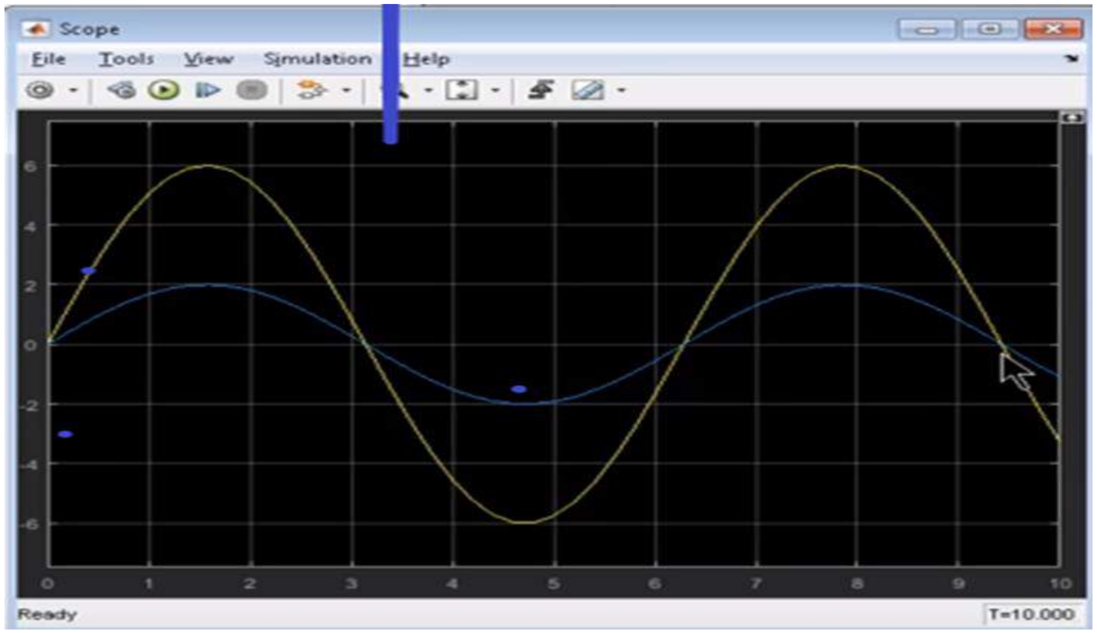
Run Simulation

From the Simulink Editor menu bar, select Simulation > Run

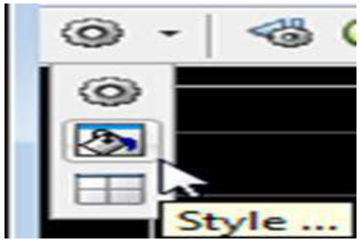
Or by clicking the Run simulation button   and Pause simulation button   on the Simulink Editor toolbar or Scope window toolbar.

you can view the simulation results in a Scope window.

Double-click the Scope block. The Scope window opens and displays the simulation results. The plot shows a sine wave signal with the resulting of Gain.



On the Scope window toolbar, click the Style button.



A Style dialog box opens with display options.

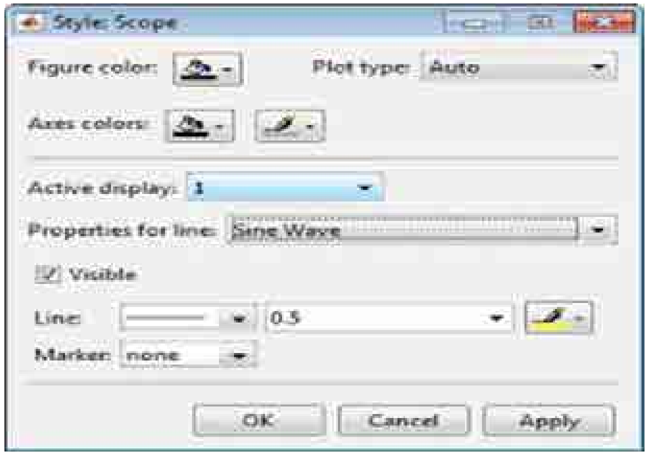

Style: Scope

Figure color:

Plot type: Auto

Axes colors:

Active display: 1

Properties for line: Sine Wave

☑ Visible

Line: 0.5

Marker: none

OK Cancel Apply

Digital Communication Lab/ Four Class

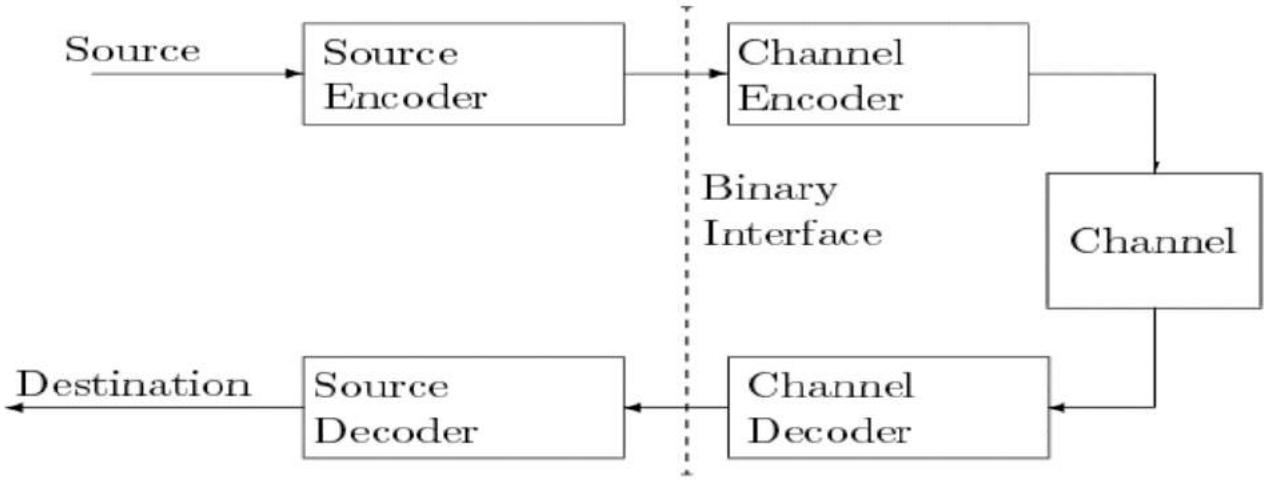
UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Assist Lec. Noor Abdul Khaleq

Communication Systems that first convert the source output into a binary sequence and then convert that binary sequence into a form suitable for transmission over particular physical media such as cable, twisted wire pair, optical fiber, or electromagnetic radiation through space.

Digital communication systems, by definition are communication systems that use such a digital sequence as an interface between the source and the channel input and similarly between the channel output and final destination.



Line Encoding Simulation RZ

Theory:

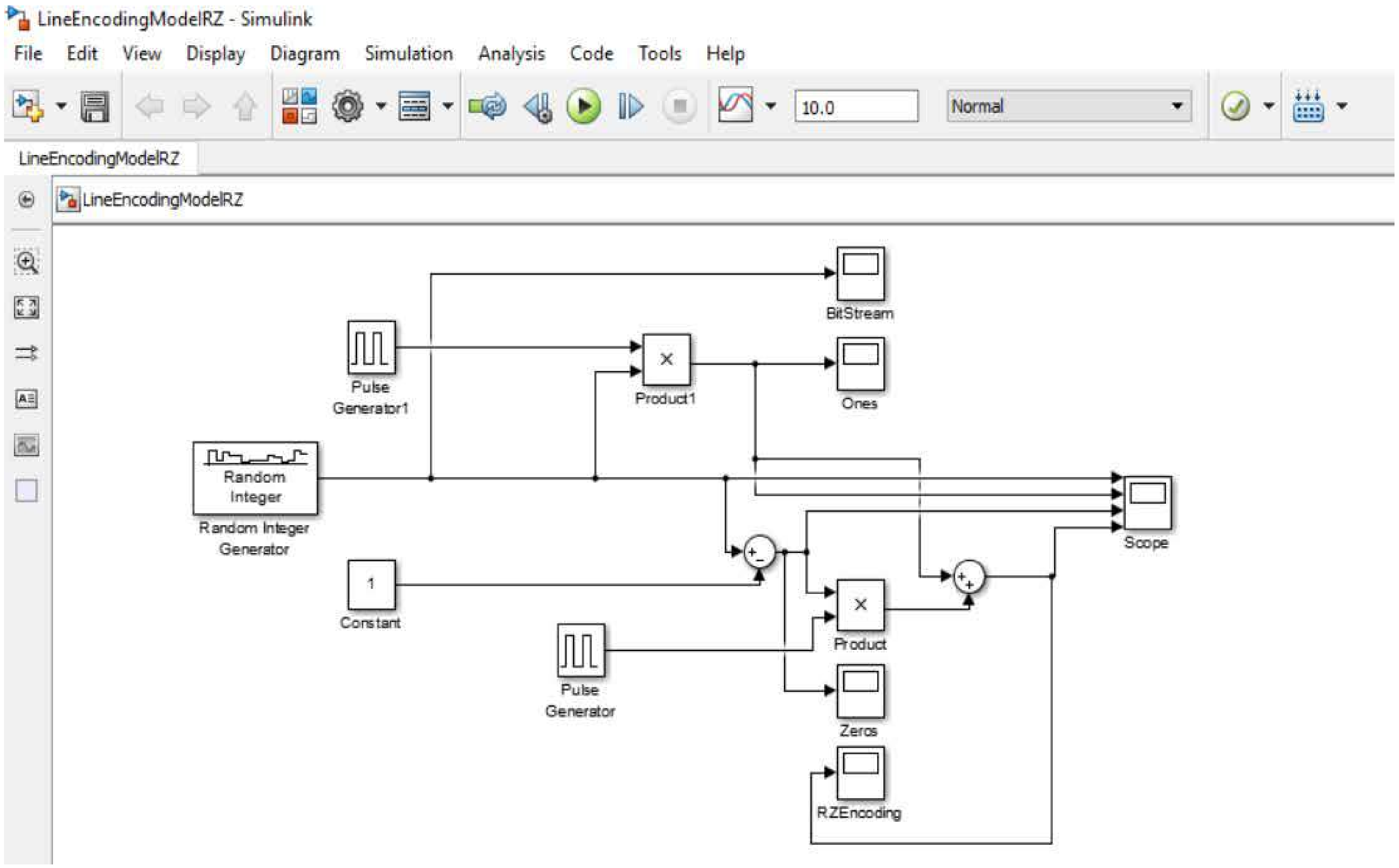
For RZ (Return to zero) encoding technique we apply the following:
+V for 1 and -V for 0.

Objective: Simulation of Line encoding for RZ

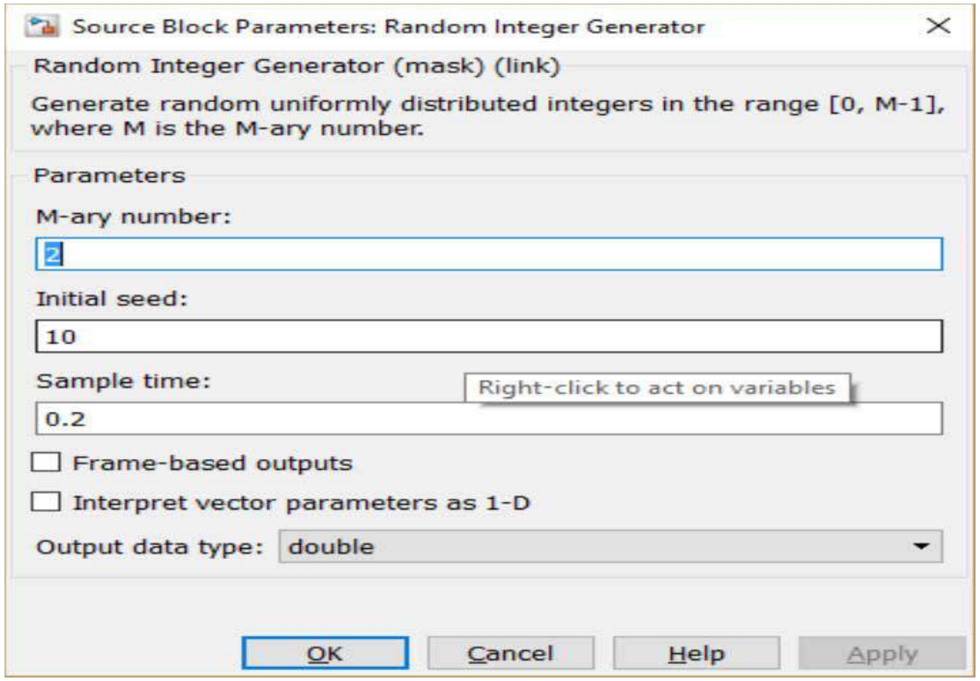
Methodology:

Using Simulink blocks to represent the conversion of binary stream into voltage levels to output the line encoded message with the following configurations:

- Using Random Integer Generator block to seed the random bit stream as input
- Pulse Generator
- Product
- Constant
- Sum
- Scope


LineEncodingModelRZ - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
LineEncodingModelRZ
LineEncodingModelRZ
BitStream
Pulse Generator1
Product1
Ones
Random Integer
Random Integer Generator
Scope
Constant
Product
Pulse Generator
Zeros
RZEncoding

double-clicked in the Random Integer Generator block to bringing up the following dialog box:


Source Block Parameters: Random Integer Generator

Random Integer Generator (mask) (link)

Generate random uniformly distributed integers in the range [0, M-1], where M is the M-ary number.

Parameters

M-ary number: 2

Initial seed: 10

Sample time: 0.2

Right-click to act on variables

☐ Frame-based outputs

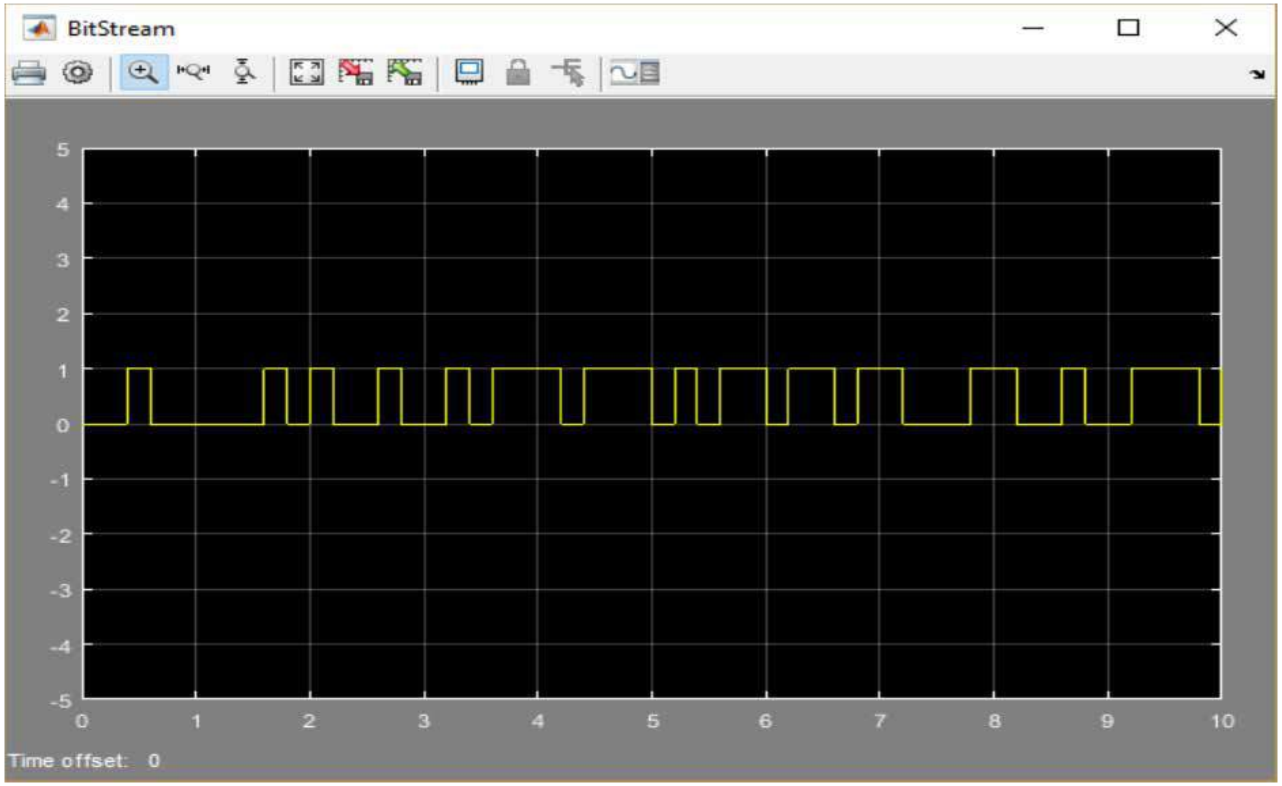
☐ Interpret vector parameters as 1-D

Output data type: double

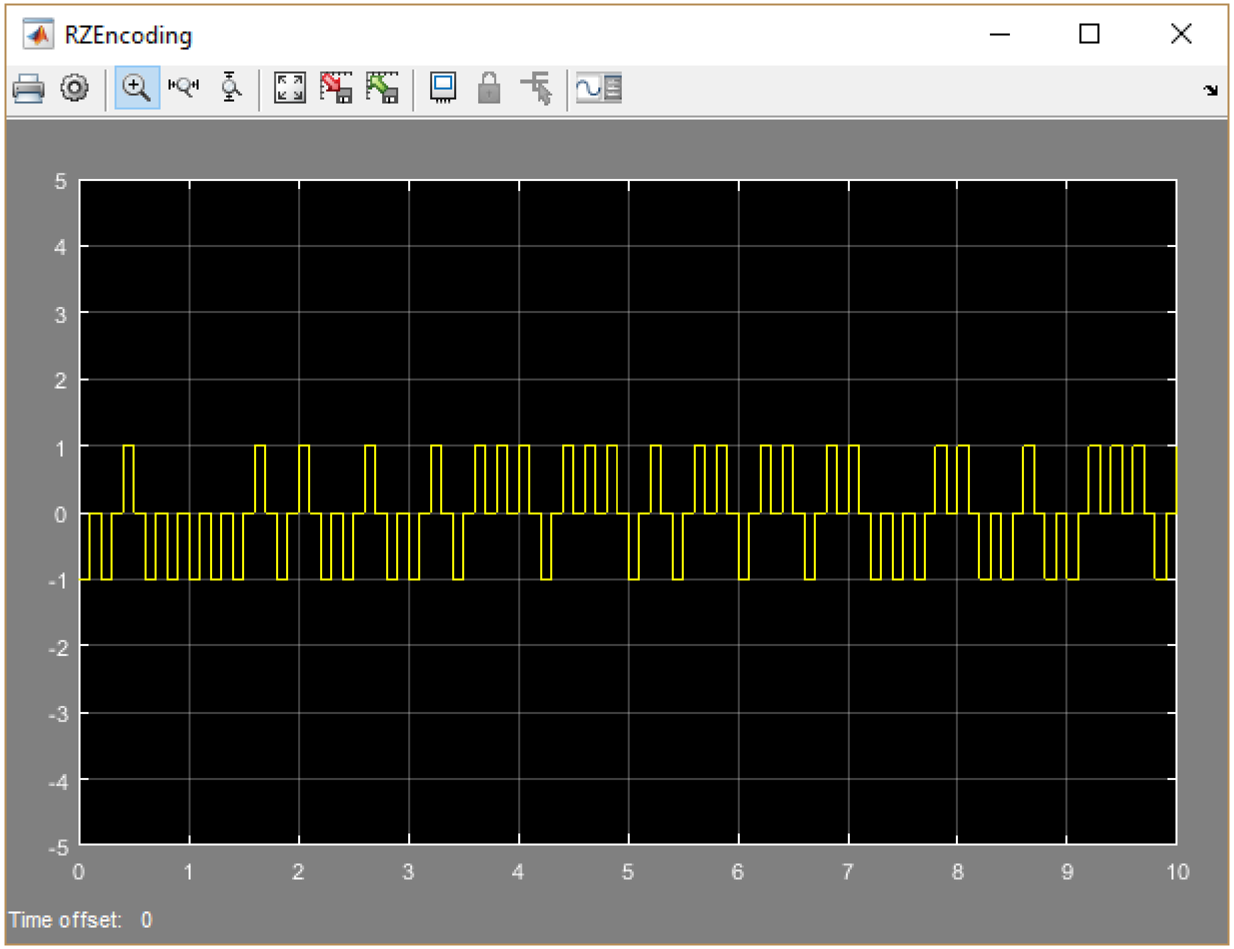
We have changed the M-ary number to (2) in order to generate random integer between 0 and M-1 which is 1 so it will represent a binary stream

Results:

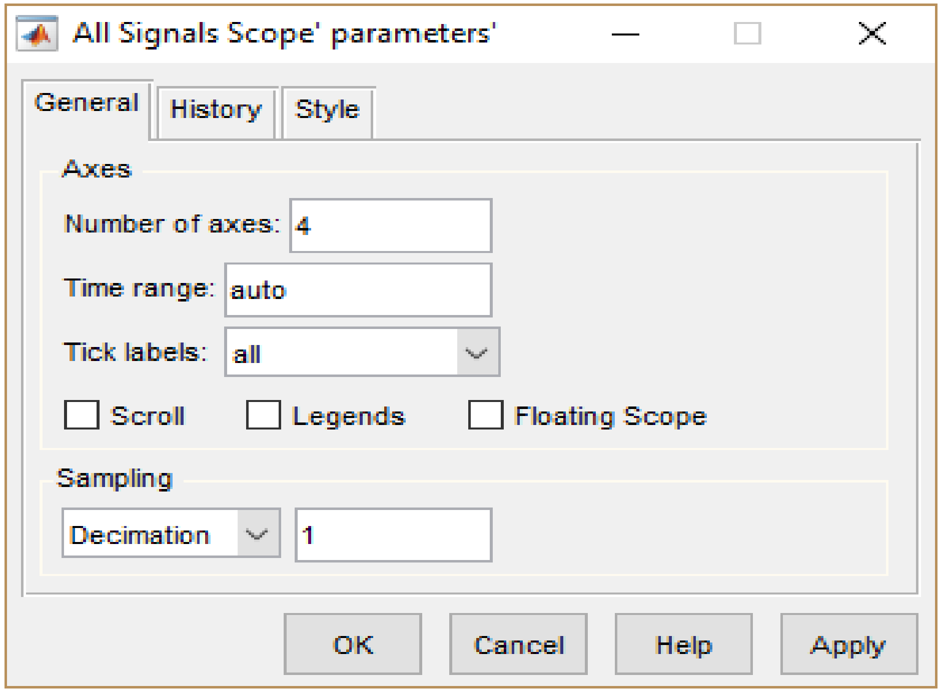
Below is the output of the random binary bit stream which is represented the message to be encoded:



Below is the Scope output of the encoded message using RZ technique



- By configure the scope block in order to show all the scopes output on the same figure, we increase the scope axes as shown below



All Signals Scope' parameters'

General | History | Style

Axes

Number of axes: 4

Time range: auto

Tick labels: all

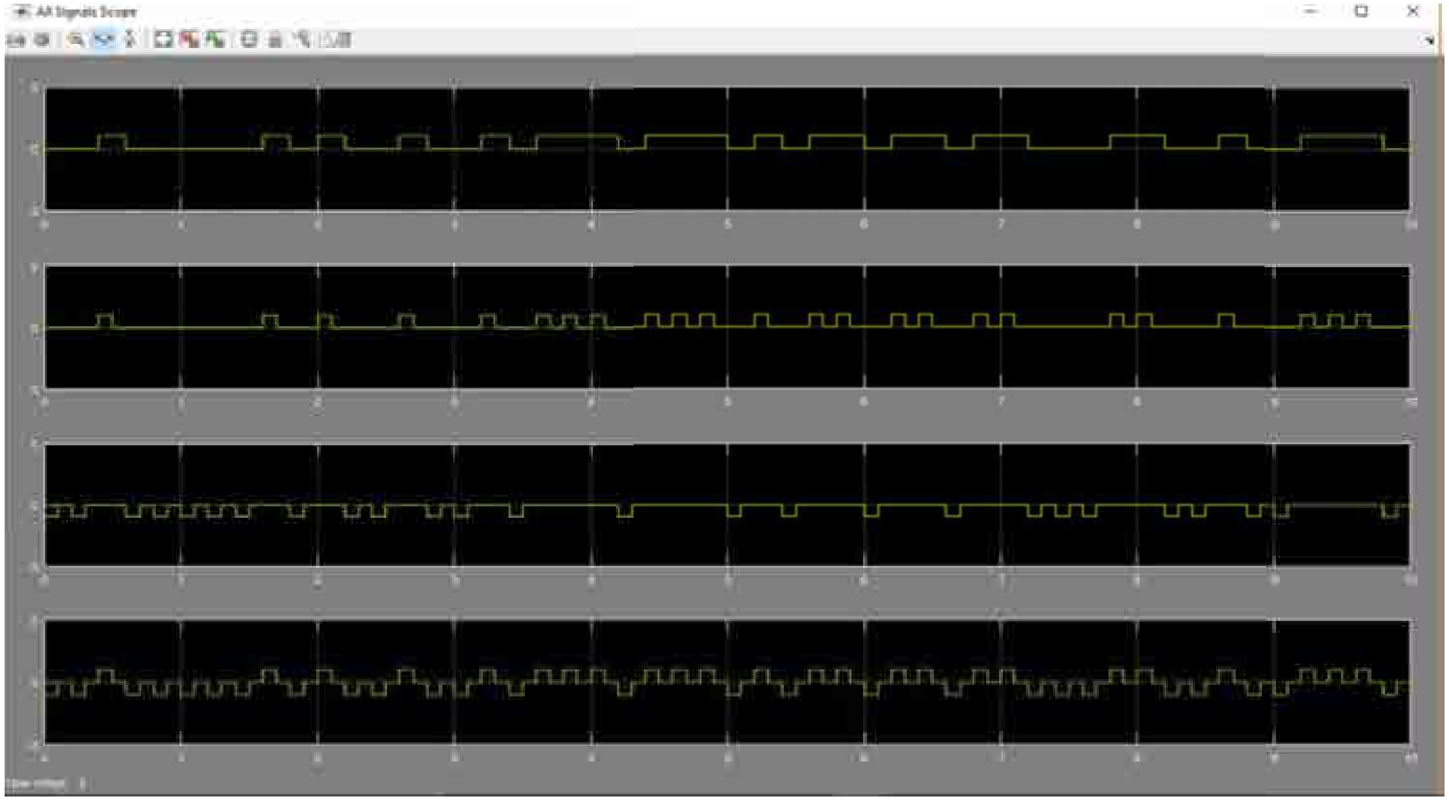
☐ Scroll ☐ Legends ☐ Floating Scope

Sampling

Decimation 1

OK Cancel Help Apply

So we can see change between the produced message and the encoded procedure:



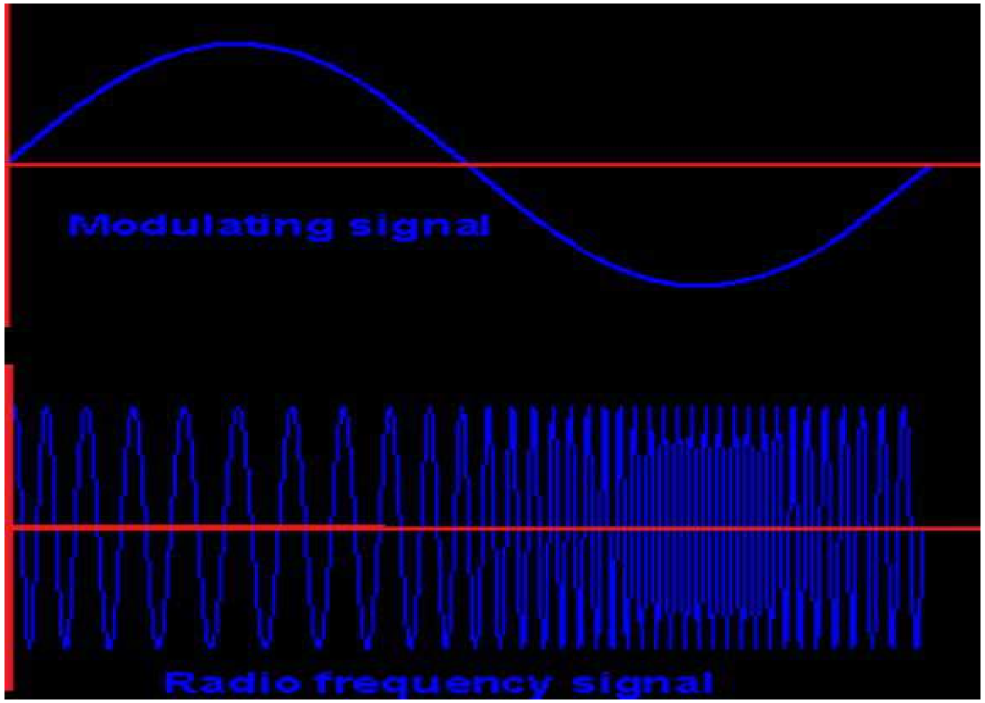
UNVERSITY OF TECHNOLOGY

Computer Engineering Department

Frequency Modulation FM

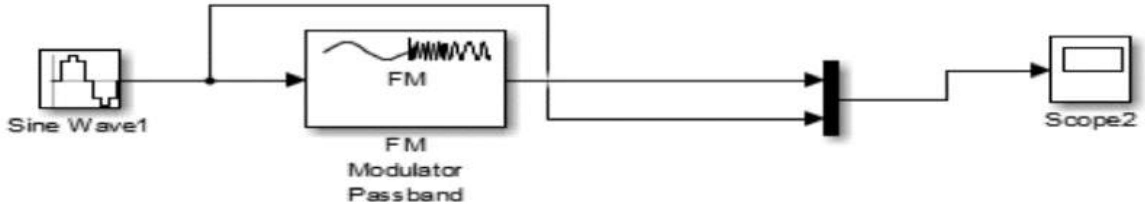
Assist Lec. Noor Abdul Khaleq

In telecommunications and signal processing, frequency modulation (FM) is the encoding of information in a carrier wave by varying the instantaneous frequency of the wave. This contrasts with amplitude modulation, in which the amplitude of the carrier wave varies, while the frequency remains constant.

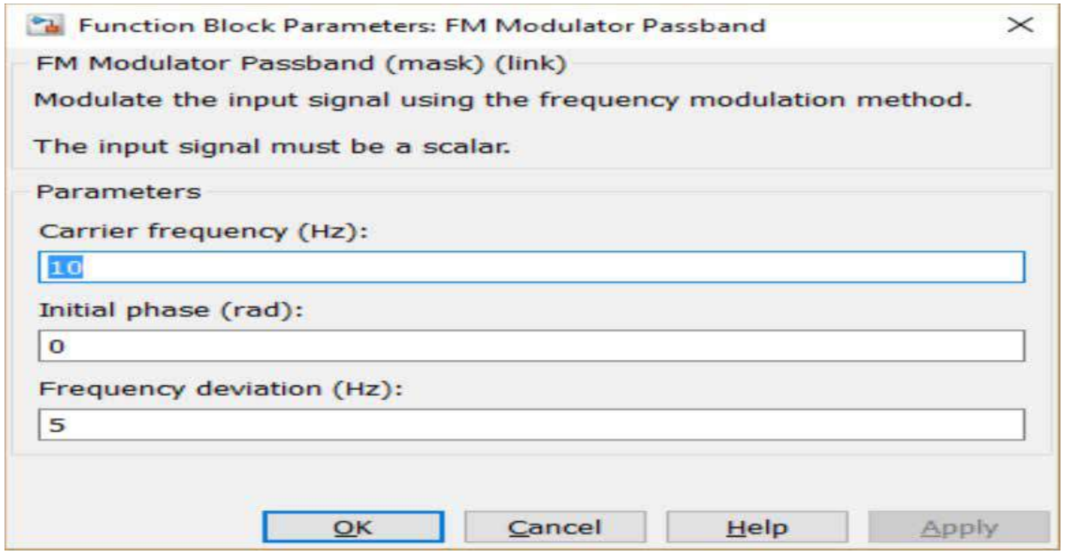


Frequency-Shift Keying:
Frequency-shift keying (FSK) is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier signal. The technology is used for communication systems such as amateur radio, caller ID and emergency broadcasts.

Modeling FM in Simulink
In order to create an FM simulation, we have to create a model in Simulink save it and add the following blocks:
-Sine Wave Input Signal
-FM Modulator Passband block
- Scope
We have also used MUX optionally for the purpose of demonstration.



By setting the parameters of the above blocks to the following settings:


Function Block Parameters: FM Modulator Passband

FM Modulator Passband (mask) (link)

Modulate the input signal using the frequency modulation method.

The input signal must be a scalar.

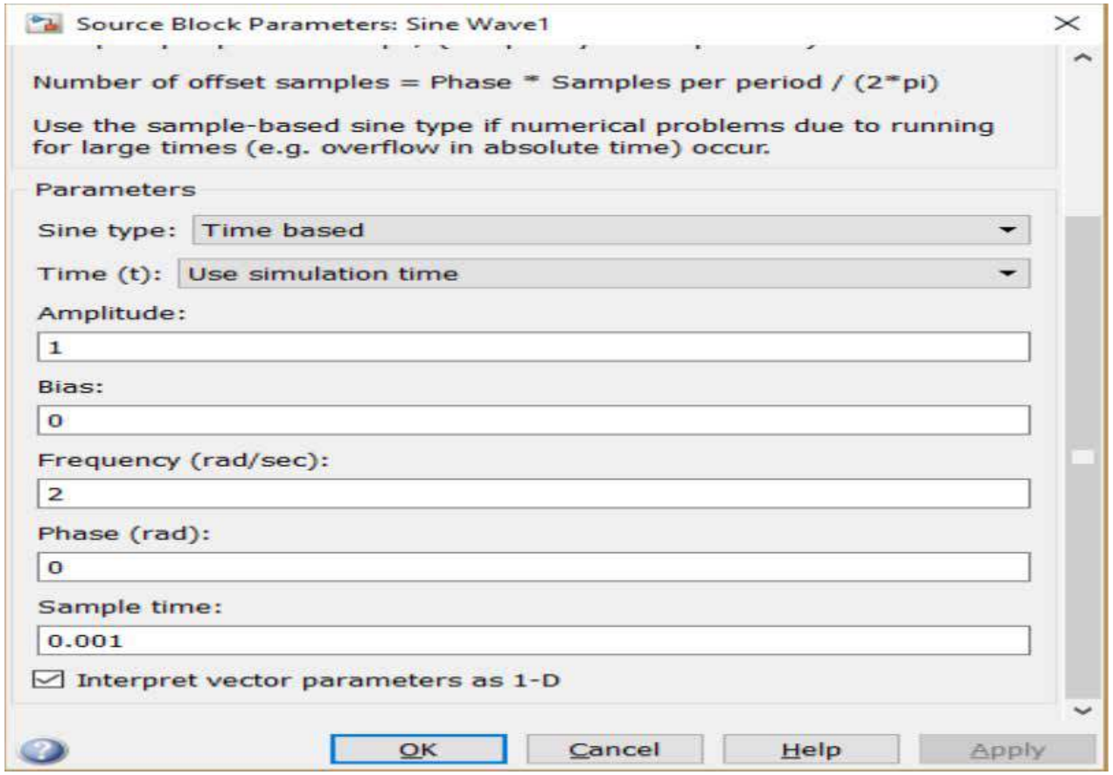
Parameters

Carrier frequency (Hz): 10

Initial phase (rad): 0

Frequency deviation (Hz): 5

And for the input signal:



Source Block Parameters: Sine Wave1

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

Frequency (rad/sec): 2

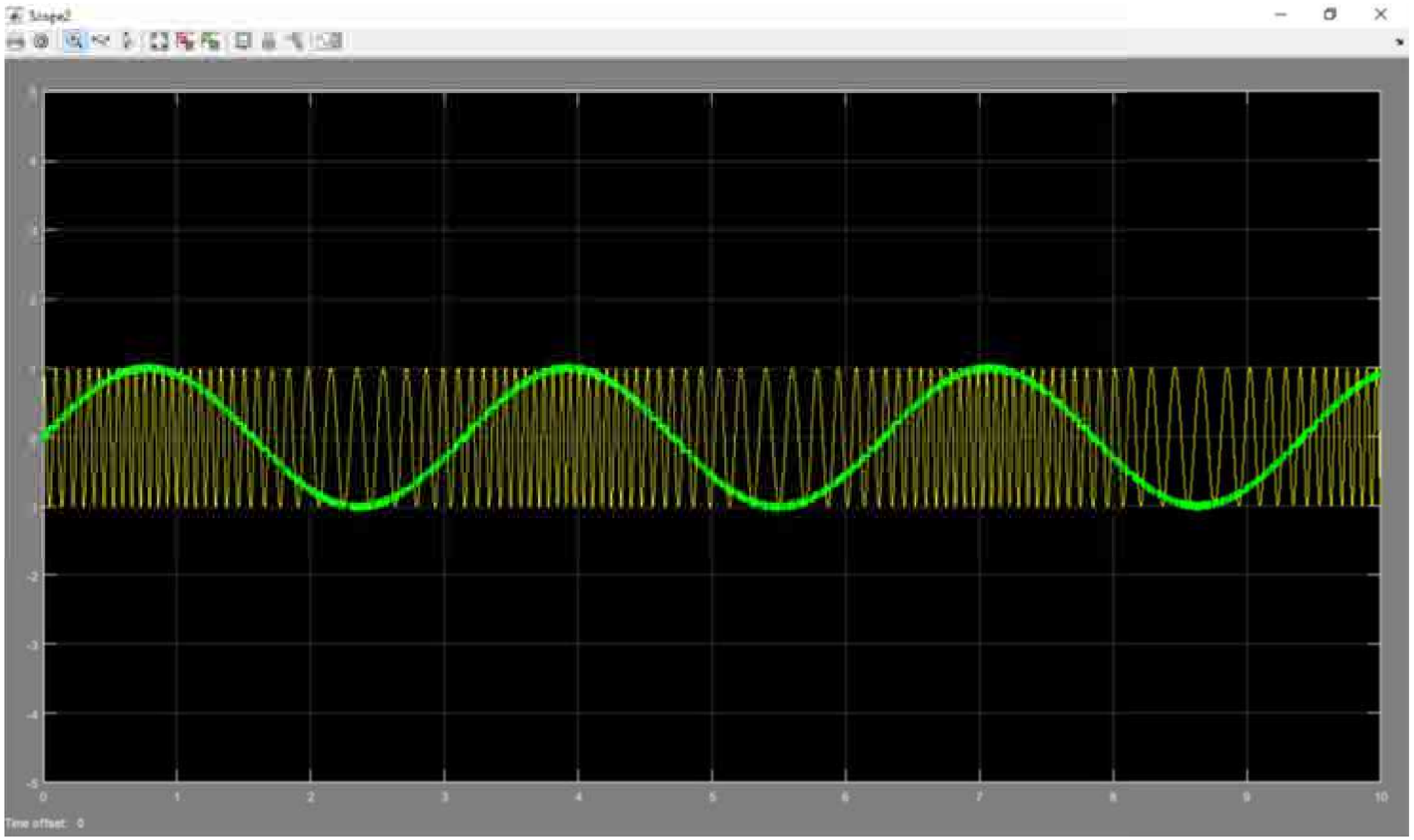
Phase (rad): 0

Sample time: 0.001

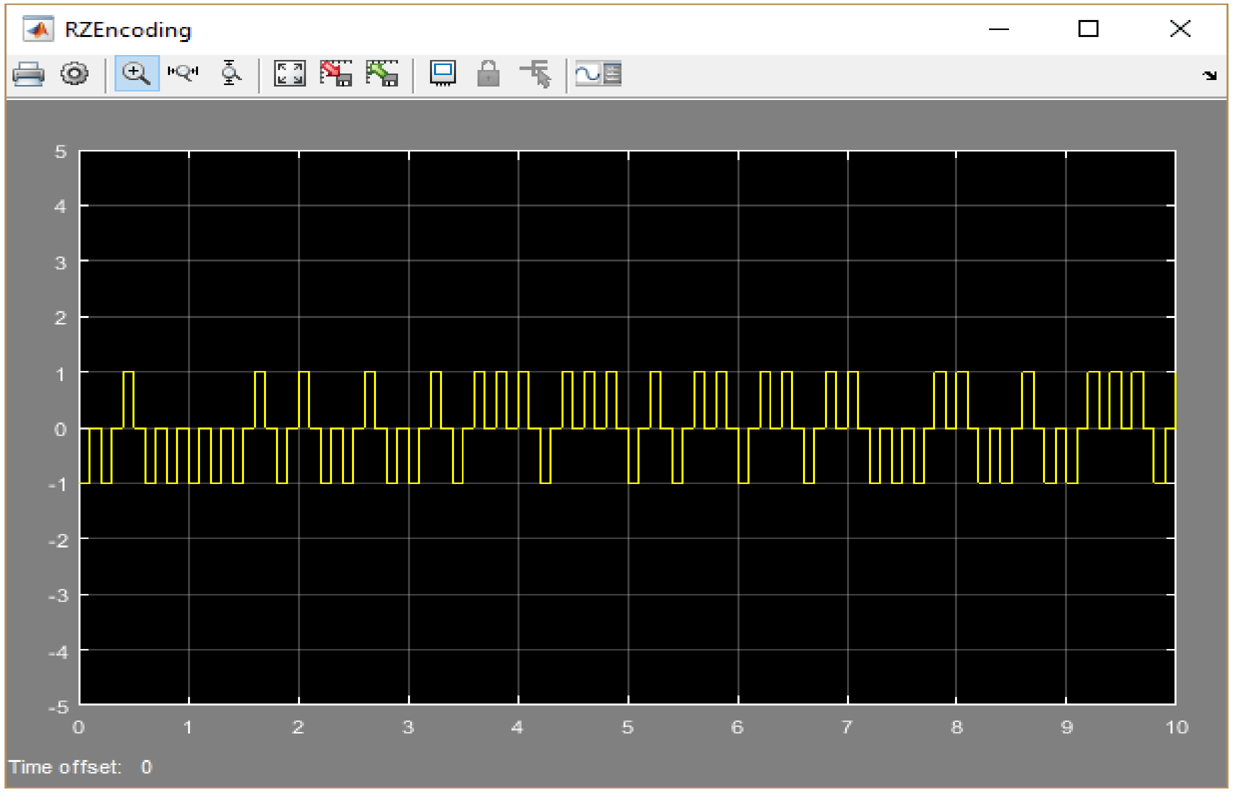
☑ Interpret vector parameters as 1-D

Results:

By setting the scope and simulate the model we get:



Below is the Scope output of the encoded message using RZ technique



We have used MUX to show the signal and the modulated FM signal on the same figure of the scope.

We can notice that the higher frequency represents the highest peak of the signal and the lowest peak represented by the lower frequencies.

FSK

Theory:

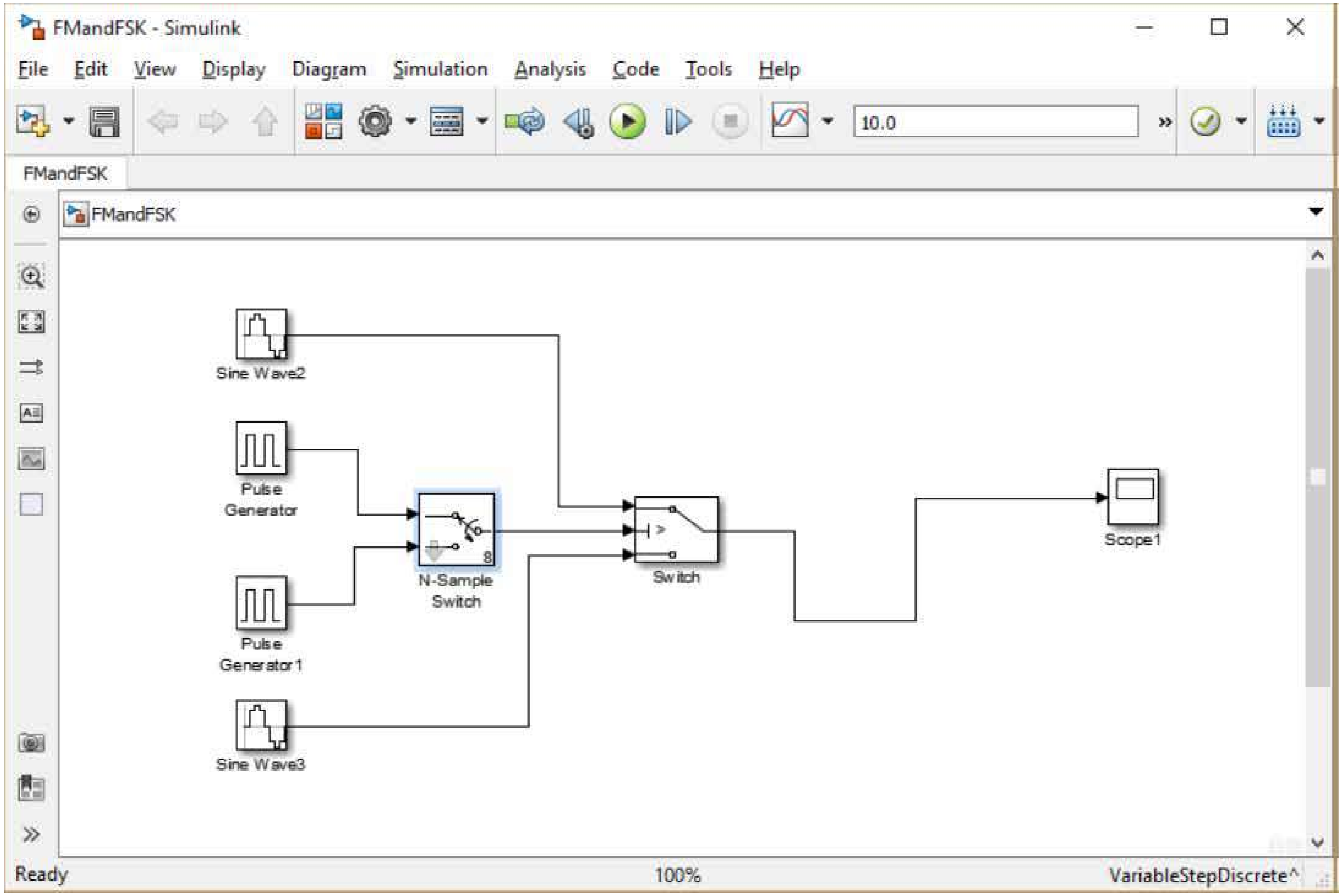
Frequency-shift keying (FSK) is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier signal

Objective:

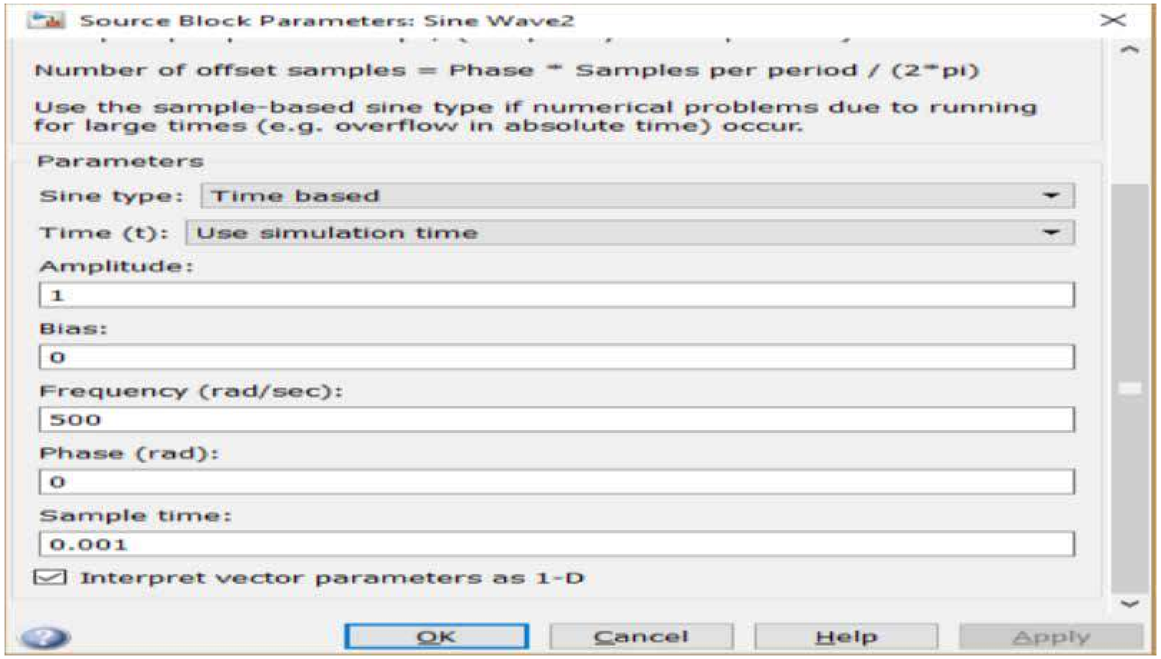
To simulate the frequency modulation for digital information.

Methodology:

Build a model in Simulink from basic block to simulate FSK modulation. As shown in figure below:


FMandFSK - Simulink
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
FMandFSK
FMandFSK
Sine Wave2
Pulse Generator
N-Sample Switch
Pulse Generator1
Sine Wave3
Switch
Scope1
Ready
100%
VariableStepDiscrete

By using different parameters setting for the sine wave input signal as below:



Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

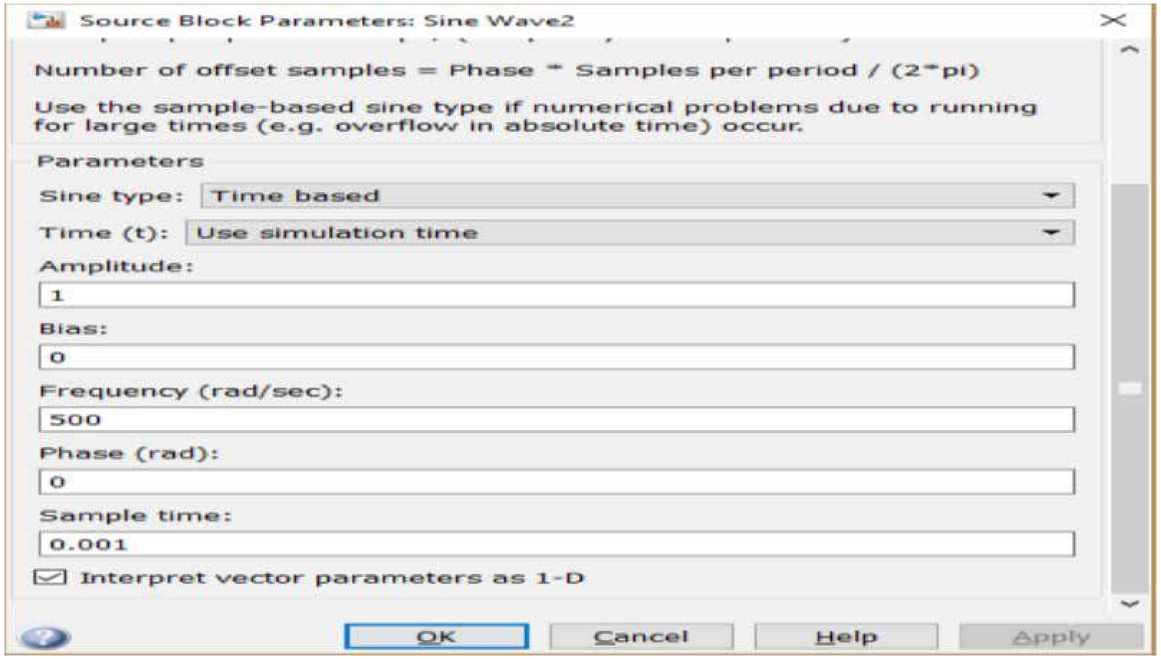
Frequency (rad/sec): 500

Phase (rad): 0

Sample time: 0.001

☑ Interpret vector parameters as 1-D

By using different parameters setting for the sine wave input signal as below:


Source Block Parameters: Sine Wave2

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude:

1

Bias:

0

Frequency (rad/sec):

500

Phase (rad):

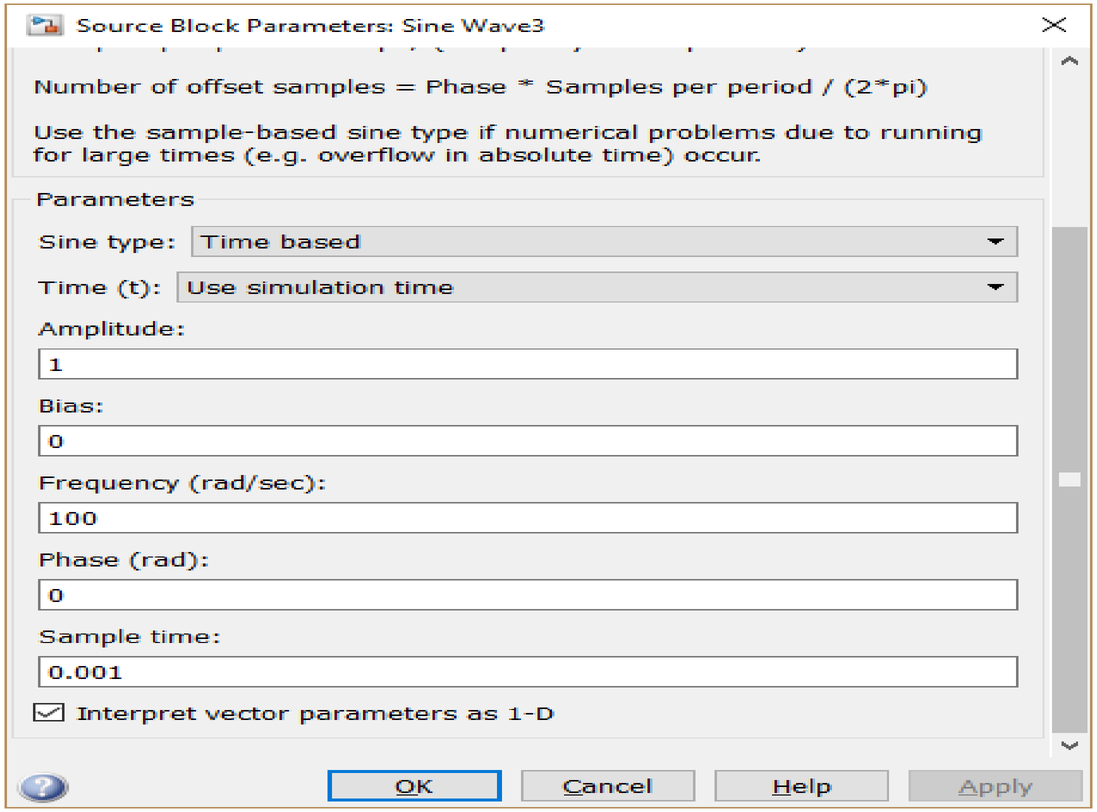
0

Sample time:

0.001

☑ Interpret vector parameters as 1-D

By using different parameters setting for the sine wave input signal as below:



Source Block Parameters: Sine Wave3

Number of offset samples = Phase * Samples per period / (2*pi)

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use simulation time

Amplitude: 1

Bias: 0

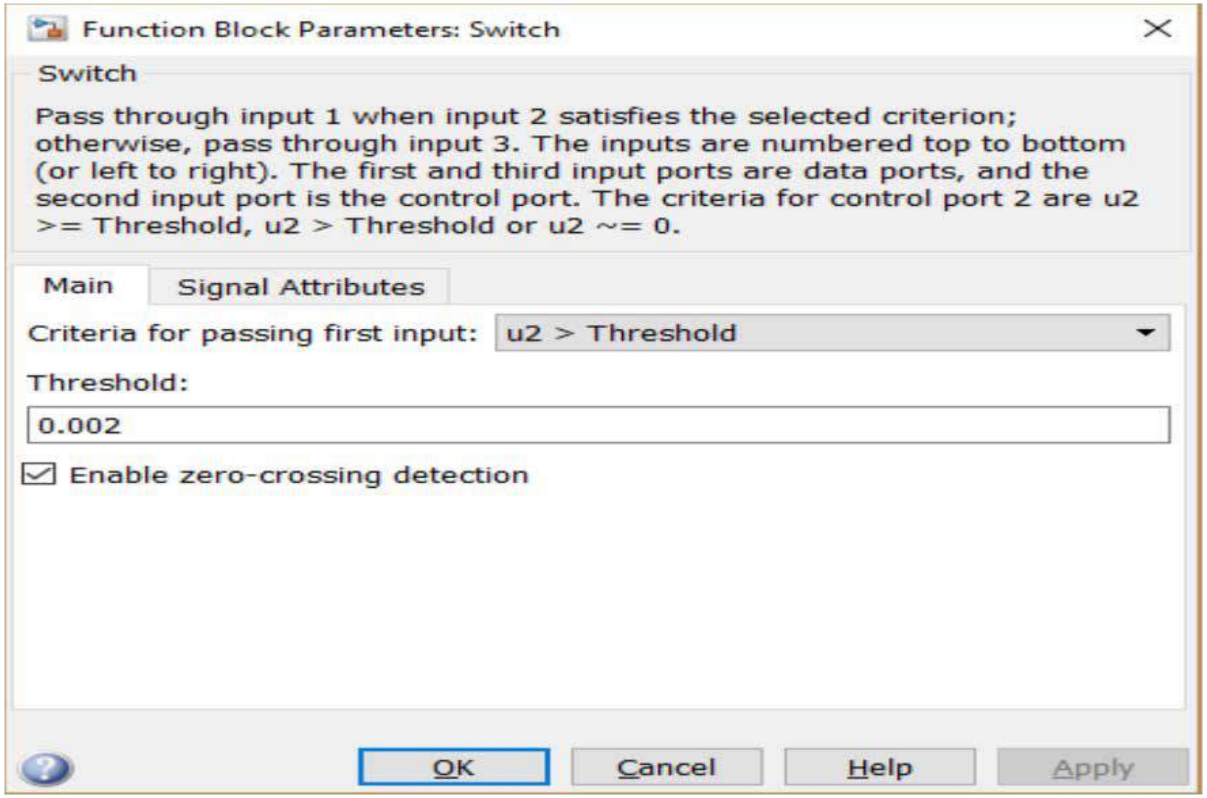
Frequency (rad/sec): 100

Phase (rad): 0

Sample time: 0.001

☑ Interpret vector parameters as 1-D

Using pulse generator and switch to combine different frequencies together representing the digital values and display it on the scope. For the switch we have set the threshold as below:



Function Block Parameters: Switch

Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The first and third input ports are data ports, and the second input port is the control port. The criteria for control port 2 are u2 >= Threshold, u2 > Threshold or u2 ~= 0.

Main | Signal Attributes

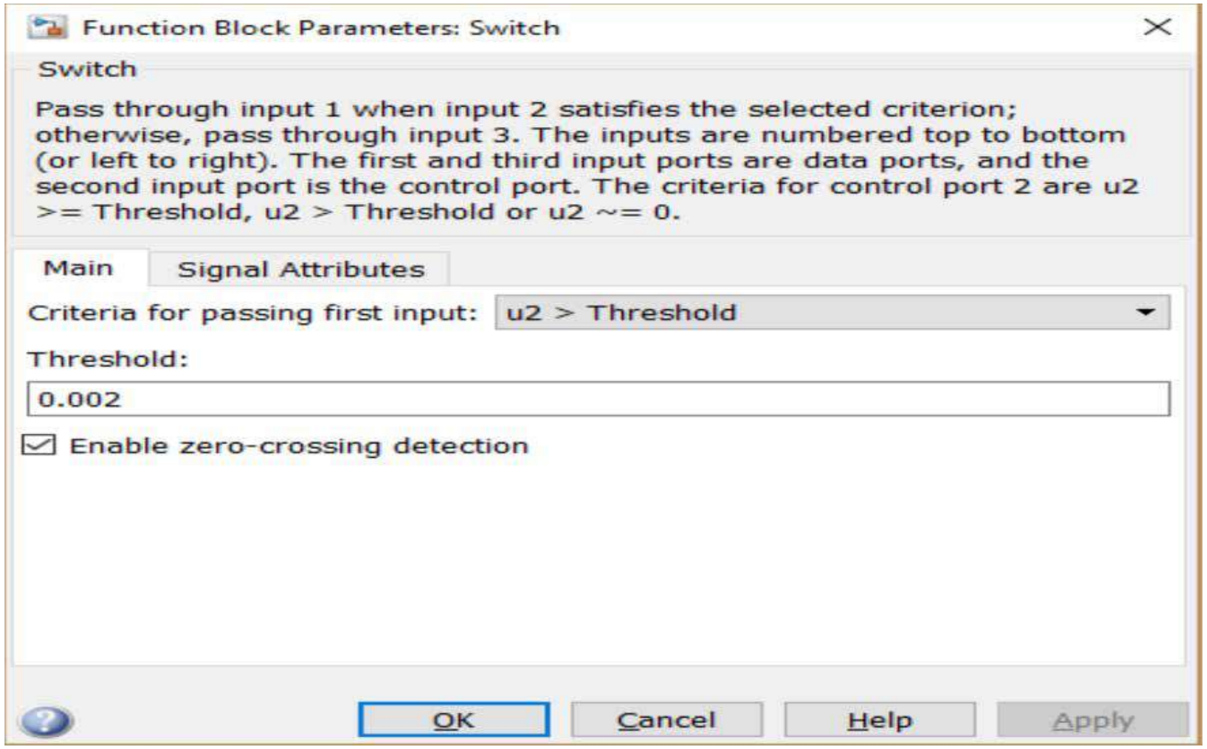
Criteria for passing first input: u2 > Threshold

Threshold:

0.002

☑ Enable zero-crossing detection

Using pulse generator and switch to combine different frequencies together representing the digital values and display it on the scope. For the switch we have set the threshold as below:



Function Block Parameters: Switch

Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The first and third input ports are data ports, and the second input port is the control port. The criteria for control port 2 are u2 >= Threshold, u2 > Threshold or u2 ~= 0.

Main | Signal Attributes

Criteria for passing first input: u2 > Threshold

Threshold:

0.002

☑ Enable zero-crossing detection

We can use MUX as well or edit the scope with multiple axes to clearly show the FSK modulation over time.