<div align="center">

Experiment No. (1)
Introduction to MATLAB

</div>

1-1   Introduction

      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.

      The name **MATLAB** stands for *matrix laboratory*. MATLAB was originally written to provide easy access to matrix.

1-1-1 Starting and Quitting MATLAB

- To start MATLAB, double-click the MATLAB shortcut icon ![icon] on your Windows desktop. You will know MALTAB is running when you see the special " >> " prompt in the MATLAB Command Window.

- To end your MATLAB session, select **Exit MATLAB** from the **File** menu in the desktop, or type **quit** (or **exit**) in the Command Window, or with easy way by click on close button ![close] in control box.

1-1-2 Desktop Tools

    1- Command Window: Use the Command Window to enter variables and run functions and M-files.

    2- Command History: Statements you enter in the Command Window are logged in the Command History. In the Command History, you can view previously run statements, and copy and execute selected statements.

    3- Current Directory Browser: MATLAB file operations use the current directory reference point. Any file you want to run must be in the current directory or on the search path.

    4- Workspace: The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory.

<div align="center">

1

</div>

Current Directory Browser   Command Window

Workspace

Command History
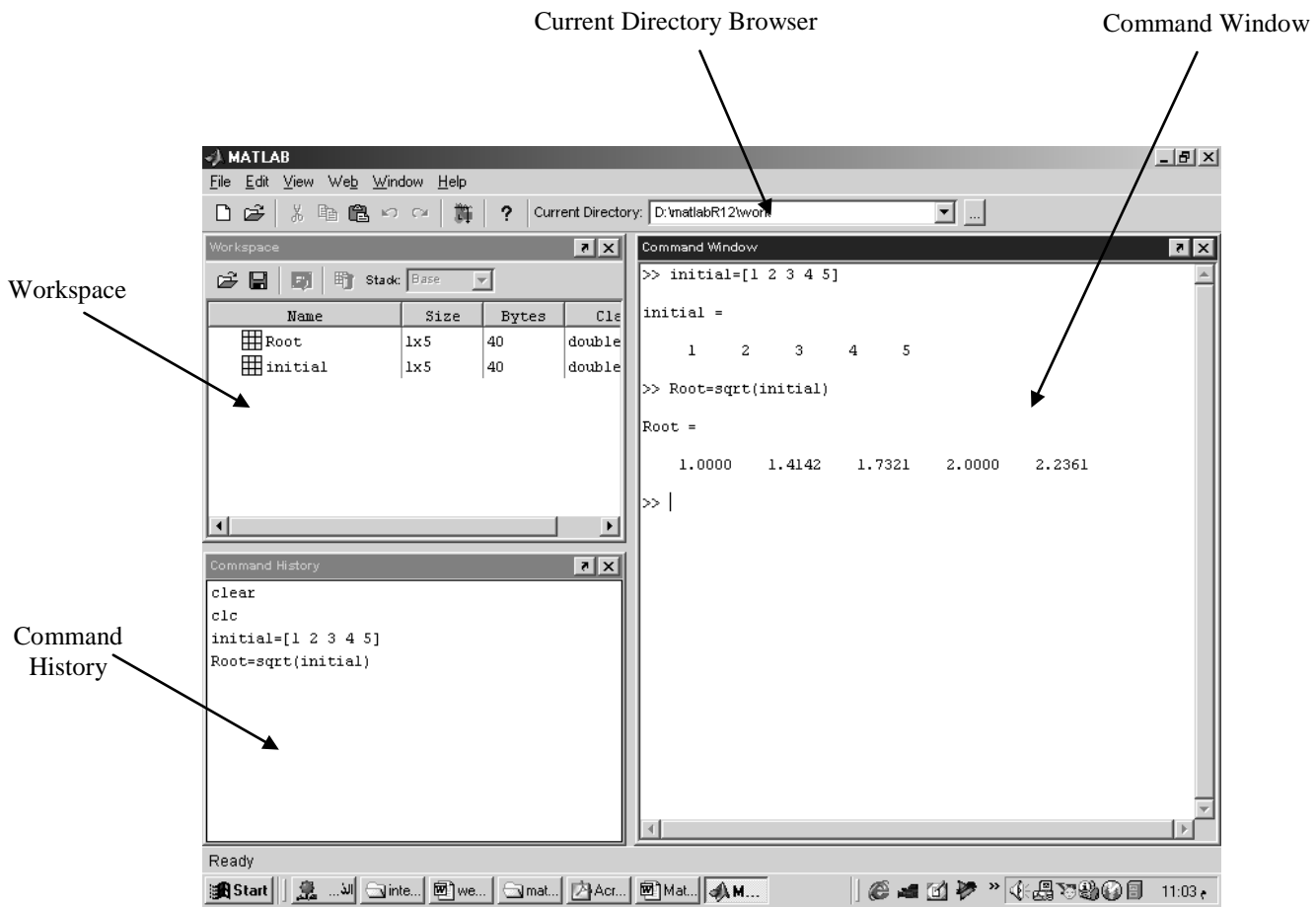
5- Editor/Debugger Window: Use the Editor/Debugger to create and debug M-files.
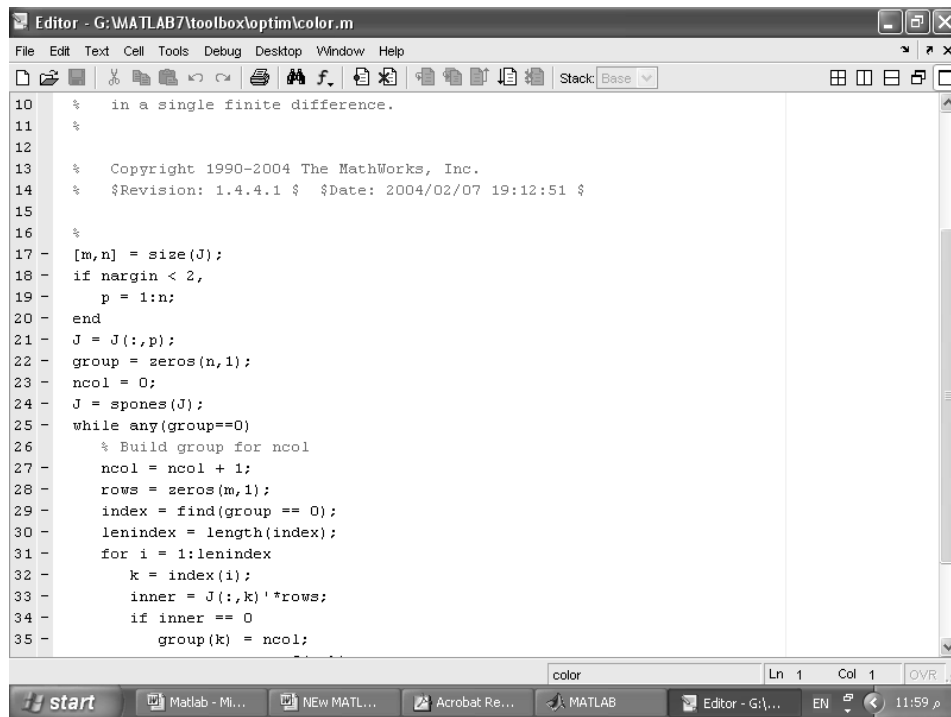
1-2   <u>Basic Commands</u>

- **clear Command:** Removes all variables from workspace.

- **clc Command:** Clears the Command window and homes the cursor.

- **help Command: help <Topic>** displays help about that **Topic** if it exist.

- **lookfor Command:** Provides help by searching through all the first lines of MATLAB help topics and returning those that contains a key word you specify.

- **edit Command:** enable you to edit (open) any M-file in Editor Window. This command doesn't open built-in function like, sqrt. See also **type Command**.

- **more command: more on** enables paging of the output in the MATLAB command window, and **more off** disables paging of the output in the MATLAB command window.

<u>Notes:</u>

- A semicolon " **;** " at the end of a MATLAB statement suppresses printing of results.

- If a statement does not fit on one line, use " **...** ", followed by **Enter** to indicate that the statement continues on the next line. For example:

```
>> S= sqrt (225)*30 /...
   (20*sqrt (100)
```

- If we don't specify an output variable, MATLAB uses the variable **ans** (short for *answer*), to store the last results of a calculation.
- Use **Up arrow** and **Down arrow** to edit previous commands you entered in Command Window.
- Insert " **%** " before the statement that you want to use it as comment; the statement will appear in green color.

Now Try to do the following:

```
>> a=3
```

```
>> a=3;                                    can you see the effect of semicolon " ; "
```

```
>> a+5                                     assign the sum of **a** and **5** to **ans**
```

```
>> b=a+5                                   assign the sum of **a** and **5** to **b**
```

```
>> clear a
```

```
>> a                                       can you see the effect of **clear** command
```

```
>> clc                                     clean the screen
```

```
>> b
```

Exercises

1- Use **edit** command to edit the **dct** function, then try to edit **sin** function. State the difference.

2- Use **help** command to get help about **rand** function.

3- Enter **a**=3; **b**=5; **c**=7, then clear the variable **b** only

Experiment No. (2)
Working with Matrices

2-1   Entering Matrix

The best way for you to get started with MATLAB is to learn how to handle matrices. You only have to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon (;) to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

For Example

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered.

```
A =

    16    3     2    13
     5   10    11     8
     9    6     7    12
     4   15    14     1
```

Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as **A**. Also you can enter and change the values of matrix elements by using **workspace** window.

2-2   Subscripts

The element in row **i** and column **j** of **A** is denoted by **A(i,j)**. For example, **A(4,2)** is the number in the fourth row and second column. For the above matrix, **A(4,2)** is **15**. So to compute the sum of the elements in the fourth column of **A**, type

```
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)
ans =
34
```

You can do the above summation, in simple way by using **sum** command.

If you try to use the value of an element outside of the matrix, it is an error.

```
>> t = A(4,5)
??? Index exceeds matrix dimensions.
```

On the other hand, if you store a value in an element outside of the matrix, the size increases to accommodate the newcomer. The initial values of other new elements are zeros.

```
>> X = A;

>> X(4,5) = 17

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

2-3 Colon Operator

The colon " **:** " is one of the most important MATLAB operators. It occurs in several different forms. The expression

```
>> 1:10
```

is a row vector containing the integers from 1 to 10

```
1 2 3 4 5 6 7 8 9 10
```

To obtain nonunit spacing, specify an increment. For example,

```
>> 100:-7:50
```

```
100 93 86 79 72 65 58 51
```

Subscript expressions involving colons refer to portions of a matrix.

```
>>A(1:k,j)
```

is the first k elements of the jth column of A.

       The colon by itself refers to *all* the elements in a row or column of a matrix and the keyword **end** refers to the *last* row or column. So

```
>> A(4,:)           or      >> A(4,1:end)      give the same action
ans =
4    15   14   1


>> A(2,end)
ans =
8
```

2-4   <u>Basic Matrix Functions</u>

Command	Description

`sum(x)`
 	
`>> x=[1 2 3`
`      4 5 6];`
 	
`>> sum(x)`
 	
`ans =`
 	
`      5      7      9`
	The sum of the elements of x. For matrices, sum(x) is a row vector with the sum over each column.

`>> sum(x,2)`
 	
`ans=`
 	
`      6`
 	
`      15`
	sum (x,dim) sums along the dimension dim.

`>>sum(sum(x))`
 	
`ans =`
 	
`    21`
	In order to find the sum of elements that are stored in matrix with n dimensions, you must use **sum** command n times in cascade form, this is also applicable for **max, min, prod, mean, median** commands.

Command	Description

`mean(x)`
 	
`x=[1 2 3; 4 5 6];`
 	
`>> mean(x)`
 	
`ans =`
 	
`    2.5    3.5     4.5`
 	
`>> mean(x,2)`
 	
`ans =`
 	
`         2`
 	
`         5`
 	
`>>mean(mean(x))`
 	
`ans =`
 	
`        3.5000`
	The average of the elements of x. For matrices, mean(x) is a row vector with the average over each column.
	
	mean (x,dim) averages along the dimension dim.

`zeros(N)`
 	
`zeros(N,M)`
 	
`>> zeros(2,3)`
 	
`ans =`
 	
`        0        0        0`
 	
`        0        0        0`
	Produce N by N matrix of zeros.
	
	Produce N by M matrix of zeros.

`ones(N)`
 	
`ones(N,M)`
 	
`>> ones(2,3)`
 	
`ans =`
 	
`        1        1        1`
 	
`        1        1        1`
	Produce N by N matrix of ones.
	
	Produce N by M matrix of ones.

Command	Description

size(x)
 	
>> x=[1 2 3
 	
    4 5 6];
 	
>> size(x)
 	
ans =
 	
    2   3
	return the size (dimensions) of matrix x.

length(v) >> v=[1 2 3]; >> length(v) ans = 3	return the length (number of elements) of vector v.

numel(x)
 	
>> v =[55 63 34];
 	
>> numel(v)
 	
ans =
 	
    3
 	
>> x=[1 2
 	
    4 5
 	
    7 8];
 	
>> numel(x)
 	
ans =
 	
    6
	returns the number of elements in array x.

Command	Description

single quote (**'**)
 	
`>> x=[1 2 3`
`      4 5 6`
`      7 8 9];`
 	
`>> x'`
`ans =`
 	
`     1      4      7`
`     2      5      8`
`     3      6      9`
`>> v=[1 2 3];`
`>> v'`
`ans =`
 	
`     1`
`     2`
`     3`
	Matrix transpose. It flips a matrix about its main diagonal and it turns a row vector into a column vector.

max (x)
`>> x=[1 2 3`
`      4 5 6];`
`>> max (x)`
`ans =`
 	
`       4      5      6`
`>> max(max(x))`
`ans =`
 	
`     6`
	Find the largest element in a matrix or a vector.

Command	Description

min (x)
 	
>> x=[1 2 3
 	
    4 5 6];
 	
>> min (x)
 	
ans =
 	
    1   2   3
 	
>> min(min(x))
 	
ans =
 	
    1
	Find the smallest element in a matrix or a vector.

magic(N)
 	
>> magic(3)
 	
ans =
 	
    8   1   6
 	
    3   5   7
 	
    4   9   2
	produce N Magic square. This command produces valid magic squares for all N>0 except N=2.

inv(x)
 	
>> x=[1 4;
 	
    5 8];
 	
>> inv(x)
 	
ans =
 	
  -0.6667   0.3333
 	
   0.4167   -0.0833
	produce the inverse of matrix x.

Command	Description

diag(x)
 	
`>> x=[1    2    3`
 	
`        4    5    6`
 	
`        7    8    9];`
 	
`>> diag(x)`
 	
`ans =`
 	
`     1`
 	
`     5`
 	
`     9`
 	
`>> v=[1 2 3];`
 	
`>> diag(v)`
 	
`ans =`
 	
`     1        0        0`
 	
`     0        2        0`
 	
`     0        0        3`
	Return the diagonal of matrix x. if x is a vector then this command produce a diagonal matrix with diagonal x.

prod(x)
 	
`>> x=[1 2 3`
 	
`         4 5 6];`
 	
`>> prod(x)`
 	
`ans =   4     10     18`
 	
`>> prod(prod(x))`
 	
`ans =`
 	
`   720`
	Product of the elements of x. For matrices, Prod(x) is a row vector with the product over each column.

Command	Description

median(x)
 	
x=[4 6 8
 	
  10 9 1
 	
   8 2 5];
 	
>> median(x)
 	
ans =
 	
    8   6   5
 	
>> median(x,2)
 	
ans =
 	
    6
 	
    9
 	
    5
 	
>> median(median(x))
 	
ans =
 	
    6
	The median value of the elements of x. For matrices, median (x) is a row vector with the median value for each column.
	
	median(x,dim) takes the median along the dimension dim of x.

sort(x,DIM,MODE)
 	
>> x = [3 7 5
        0 4 2];
>> sort(x,1)
ans =
    0   4   2
    3   7   5
>> sort(x,2)
ans =
    3   5   7
    0   2   4
>> sort(x,2,'descend')
ans =
    7   5   3
    4   2   0
	Sort in ascending or descending order.
	- For vectors, sort(x) sorts the elements of x in ascending order.
	  For matrices, sort(x) sorts each column of x in ascending order.
	
	DIM= 1 by default
	MODE= 'ascend' by default

Command	Description

det(x)
 	
`>> x=[5 1 8`
`      4 7 3`
`      2 5 6];`
`>> det(x)`
`ans =`
`   165`
	Det is the determinant of the square matrix x.

tril(x)
 	
`>> x=[5 1 8`
`      4 7 3`
`      2 5 6];`
`>> tril(x)`
`ans =`
`     5     0     0`
`     4     7     0`
`     2     5     6`
	Extract lower triangular part of matrix x.

triu(x)
 	
`>> x=[5 1 8`
`      4 7 3`
`      2 5 6];`
`>> triu(x)`
`ans =`
`     5     1     8`
`     0     7     3`
`     0     0     6`
	Extract upper triangular part of matrix x.

Note

When we are taken away from the world of linear algebra, matrices become two-dimensional numeric arrays. Arithmetic operations on arrays are done element-by-element. This means that addition and subtraction are the same for arrays and matrices, but **that multiplicative operations are different**. MATLAB uses a **dot (.), or decimal point, as part of the notation for multiplicative array operations**.

Example: Find the factorial of 5

```
>> x=2:5;
>> prod(x)
```

Example: if x = [1,5,7,9,13,20,6,7,8], then

    a) replace the first five elements of vector x with its maximum value.
    b) reshape this vector into a 3 x 3 matrix.

solution

```
a)
 >> x(1:5)=max(x)

b)
 >> y(1,:)=x(1:3);
 >> y(2,:)=x(4:6);
 >> y(3,:)=x(7:9);
 >> y
```

Example: Generate the following row vector   b=[1, 2, 3, 4, 5, ... 9,10], then   transpose it to column vector.

solution

```
>> b=1:10
b =
    1    2    3    4    5    6    7    8    9    10
>> b=b';
```

Exercises

1- If `x=[1 4; 8 3]`, find :
   a) the inverse matrix of x .
   b) the diagonal of x.
   c) the sum of each column and the sum of whole matrix x.
   d) the transpose of x.

2- If `x= [2 8 5; 9 7 1],   b=[2 4 5]` find:
   a) find the maximum and minimum of x.
   b) find median value over each row of x.
   c) add the vector **b** as a third row to x.

3- If `x=[ 2 6 12; 15 6 3; 10 11 1]`, then

   a) replace the first row elements of matrix x with its average value.
   b) reshape this matrix into row vector.

4- Generate a 4 x 4 Identity matrix.

5- Generate the following row vector b=[5, 10, 15, 20 . . . . 95, 100], then find the number of elements in this vector.

Experiment No. (3)
Expressions

Like most other programming languages, MATLAB provides mathematical *expressions*, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

1- Variable   2- Numbers   3- Operators   4- Functions

3-1   Variable

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

creates a 1-by-1 matrix named num_students and stores the value 25 in its

single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. **A** and **a** are not the same variable.

3-2   Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. *Scientific notation* uses the letter (**e**) to specify a power-of-ten scale factor. *Imaginary numbers* use either **i** or **j** as a suffix. Some examples of legal numbers are

3	-99	0.0001	9.6397238	1.60210e-20

6.02252e23   1i   3+5j

3-3   <u>Arithmetic Operators</u>

Operator	Description
+	Plus
-	Minus
*	Matrix multiply
.*	Array multiply
^	Matrix power
.^	Array power
\	Backslash or left matrix divide
/	Slash or right matrix divide
.\	Left array divide
./	Right array divide
( )	Specify evaluation order

3-4   <u>Function</u>

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. <u>Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically</u>. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

```
>> help elfun
```

For a list of more advanced mathematical and matrix functions, type

```
>> help specfun
>> help elmat
```

Some of the functions, like sqrt and sin, are *built in*. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other

functions, like gamma and sinh, are implemented in M-files. You can see the code and even modify it if you want.

Command	Description
abs(x)	Absolute value (magnitude of complex number).
acos(x)	Inverse cosine.
angle(x)	Phase angle (angle of complex number).
asin(x)	Inverse sine.
atan(x)	Inverse tangent.
atan2(x,y)	Four quadrant inverse tangent: $\tan^{-1}(x/y)$.
ceil(x)	Round towards plus infinity.
conj(x)	Complex conjugate.
cos(x)	Cosine of x, assumes radians.
exp(x)	Exponential: e^x.
fix(x)	Round towards zero.
floor(x)	Round towards minus infinity.
imag(x)	Complex imaginary part.
log(x)	Natural logarithm: ln(x).
log10(x)	Common (base 10) logarithm: $\log_{10}(x)$.
log2(x)	Base 2 logarithm: $\log_2(x)$.
real(x)	Complex real part.
rem(x)	Remainder after division.
mod(x)	Modulus after division.
round(x)	Round towards nearest integer.
sign(x)	Signum: return sign of argument.
sin(x)	Sine of x, assumes radians.
sqrt(x)	Square root.
tan(x)	Tangent of x, assumes radians.

rand(n)	returns an N-by-N matrix containing pseudorandom values drawn from the standard uniform distribution on the open interval(0,1). **rand(M,N)** returns an M-by-N matrix. **rand** returns a scalar.
randn(n)	returns an N-by-N matrix containing pseudorandom values drawn from the standard normal distribution on the open interval(0,1). **randn(M,N)** returns an M-by-N matrix. **randn** returns a scalar.

Several special functions provide values of useful constants.

Constant	Description
eps	Floating point relative accuracy ≈ 2.2204e-016.
realmax	Largest positive floating point number.
realmin	Smallest positive floating point number.
pi	3.1415926535897....
i and j	Imaginary unit $\sqrt{-1}$.
inf	Infinity, e.g. 1/0
NaN	Not A Number, e.g. 0/0

3-5   <u>Examples of Expressions</u>

```
1)   >> x = (1+sqrt(5))/2
     x =
     1.6180
     >> a = abs(3+4i)
     a =
     5
     >> y=sin(pi/3)+cos(pi/4)-2*sqrt(3)
     y =
     -1.8910
```

2)   Solve the following system

x+y=1
x-y+z=0
x+y+z=2

Solution

```
>> a=[1 1 0; 1 -1 1; 1 1 1]; b=[1;0;2];
>> x=inv(a)*b
or
>> x=a\b
```

Exercises

1- Write a MATLAB program to calculate the following expression and round the answers to the nearest integer.

a) $z=\sqrt{5x^2+y^2}$ where   x=2, y=4

b) $z=4\cos(x)+j6\sin(x)$ where   x=π/4

c) $z=3\sin(x)+4\cos(x)+3e^y$ where   x=π/3 , y=2

d) $y=\sin(x)/x$ where $0 \le x \le 2\pi$

2- Solve the following system

x + y - 2z = 3
2x + y = 7
x + y - z = 4

3- Use [**round, fix, ceil, floor**] commands to round the following numbers towards integer numbers:

Before	After
1.3	1
1.5	1
1.9	2
11.9	11
-2.9	-2
-3.9	-4
3.4	3

4- Write a Program to calculate the electromagnetic force between two electrons placed (in vacuum) at a distance ($r = 2*10^{-13}$ m) from each other. Charge of electron (Q) is $1.6*10^{-19}$ C.

<u>Hint</u>

$$\text{Electromagnetic Force} = K\frac{Q_1Q_2}{r^2}$$

$K=9*10^9$

5- Generate ten values from the uniform distribution on the interval [2, 3.5].

Experiment No. (4)
Relational and Logical Operations

     These operations and functions provide answers to **True-False** questions. One important use of this capability is to control the flow or order of execution of a series of MATLAB commands (usually in an M-file) based on the results of true/false questions.

     As inputs to all relational and logical expressions, MATLAB considers any nonzero number to be true, and zero to be False. The output of all relational and logical expressions produces *one for True* and *zero for False*, and the array is flagged as *logical*. That is, the result contains numerical values 1 and 0 that can be used in mathematical statement, but also allow logical array addressing.

4-1   Relational Operations

Operation	Description
= =	Equal
~ =	Not equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

Example:

```
>> a=1:9; b=9-a;
>> t=a>4       %finds elements of (a) that are greater than 4.

t =

    0    0    0    0    1    1    1    1    1
```

Zeros appear where a ≤ 4, and ones where a > 4.

```
>> t= (a==b)      %finds elements of (a) that are equal to those in (b).


t =
     0      0      0      0      0      0      0      0      0
```

4-2 Logical Operation

Operation	Description
&	Logical AND **and(a,b)**
|	Logical OR **or(a,b)**
~	Logical NOT
xor (a,b)	Logical EXCLUSIVE OR

Example:

```
>> a = [0   4   0   -3   -5   2];
>> b = ~a
b =
     1      0      1      0      0      0


>> c=a&b
c =
     0      0      0      0      0      0
```

Example: let x=[2 -3 5 ;0 11 0], then

      a) find elements in x that are greater than 2
      b) find the number of nonzero elements in x

solution

```
     a)
        >> x>2
        ans =

                0      0      1
                0      1      0
```

```
b)
    >> t=~(~x);
    >> sum(sum(t))

    ans =

          4
```

4-3 Bitwise Operation

MATLAB also has a number of functions that perform bitwise logical operations**.**

If A, B unsigned integers then**:**

Operation	Description
bitand (A, B)	Bitwise AND
bitor (A, B)	Bitwise OR
bitset (A, BIT)	sets bit position **BIT** in **A** to 1
bitget (A, BIT)	returns the value of the bit at position **BIT** in A
xor (A, B)	Bitwise EXCLUSIVE OR

Example: **if A=5, B=6 then:**   **5**

```
>> bitget(A,3)
ans = 1                        where   A= 1 0 1 0 0 0 0 0

>> bitget(A,(1:8))
ans =
        1   0   1   0   0   0   0   0
>> bitand(A,B)
ans =
        4
>> and(A,B)
ans =
        1
```

4-4   Logical Functions

MATLAB has a number of useful logical functions that operate on scalars, vectors, and matrices. Examples are given in the following list:-

Function	Description
`any(x)`	True if any element of a vector is a nonzero number or is logical 1 (TRUE)
`all(x)`	True if all elements of a vector are nonzero.
`find(x)`	Find indices of nonzero elements
`isnan(x)`	True for Not-a-Number
`isinf(x)`	True for infinite elements.
`isempty(x)`	True for empty array.

Example: Let A=[4 9 7 0 5],

>> any(A)
ans =   1

>> all(A)
ans =   0

>> find(A)
ans =   1   2   3   5

To remove zero elements from matrix
>> B=A(find(A));
>> B
B =   4   9   7   5

To find the **location** of maximum number of **B**
>> find(B==max(B))
ans =   2

Exercises

1- write a program to read three bits x, y, z, then compute:

```
a)    v = (x and y) or z
b)    w = not (x or y) and z
c)    u = (x and not (y)) or (not (x) and y)
```

2- Write a program for three bits parity generator using even-parity bit.

3- Write a program to convert a three bits binary number into its equivalent gray code.

4- if q=[1 5 6 8 3 2 4 5 9 10 1],x=[3 5 7 8 3 1 2 4 11 5 9],
   then:

   a) find elements of (q) that are greater than 4.
   b) find elements of (q) that are equal to those in (x).
   c) find elements of (x) that are less than or equal to 7.

5- If x=[10 3 ; 9 15],   y=[10 0; 9 3], z=[-1 0; -3 2], what is the
   output of the following statements:

```
a) v = x > y
b) w = z >= y
c) u = ~z & y
d) t = x & y < z
```

Experiment No. (5)
Plotting Function

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs.

5-1 Creating a Plot Using Plot Function

The plot function has 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 you specify two vectors as arguments, plot(x,y) produces a graph of y versus x. For example, these statements use the colon operator to create a vector of x values ranging from zero to 2*pi, compute the sine of these values, and plot the result.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(y)
```
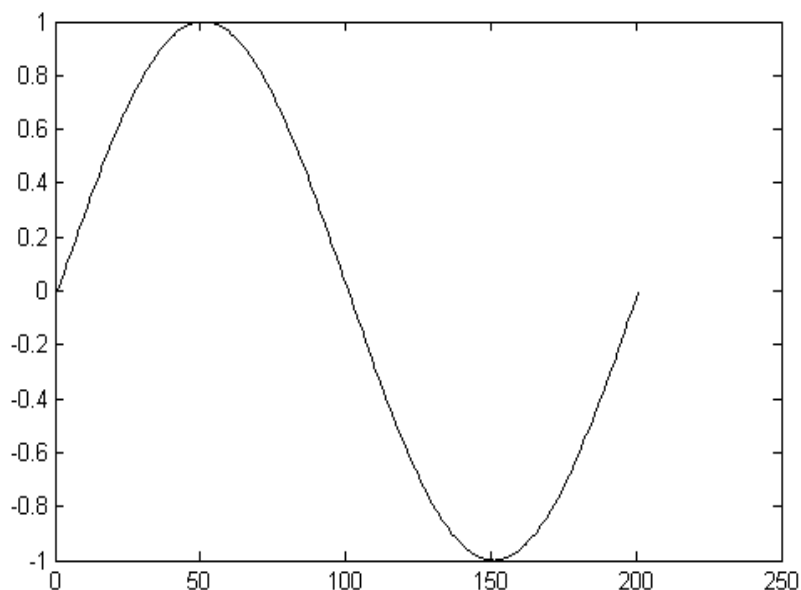
Now plot **y** variable by using:

    plot(x,y)

 can you see the difference at x-axis
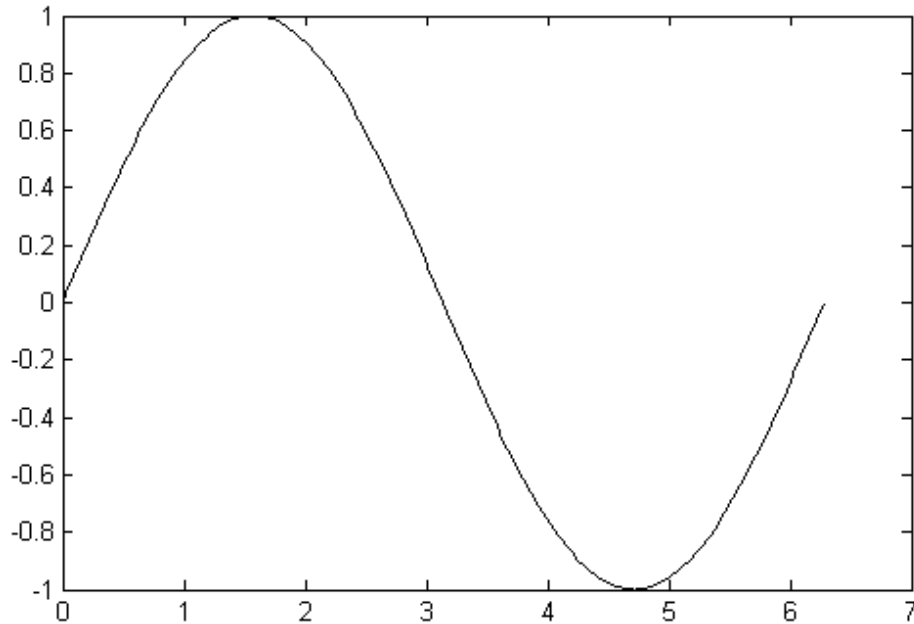
5-2   Specifying Line Styles and Colors

Various line types, plot symbols and colors may be obtained with **plot(x,y,s)** where s is a character string made from one element from any or all the following 3 columns:

Color		Marker		Line Style	
b	blue	**.**	point	**-**	solid
g	green	**o**	circle	**:**	dotted
r	red	**x**	x-mark	**-.**	dashdot
c	cyan	**+**	plus	**--**	dashed
m	magenta	*****	star	(none)	no line
y	yellow	**s**	square		
k	black	**d**	diamond		
		v	triangle (down)		
		>	triangle (left)		
		p	pentagram		
		h	hexagram		

Example:

```
x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
plot(x1,sin(x1),'r:',x2,cos(x2),'r+')
```
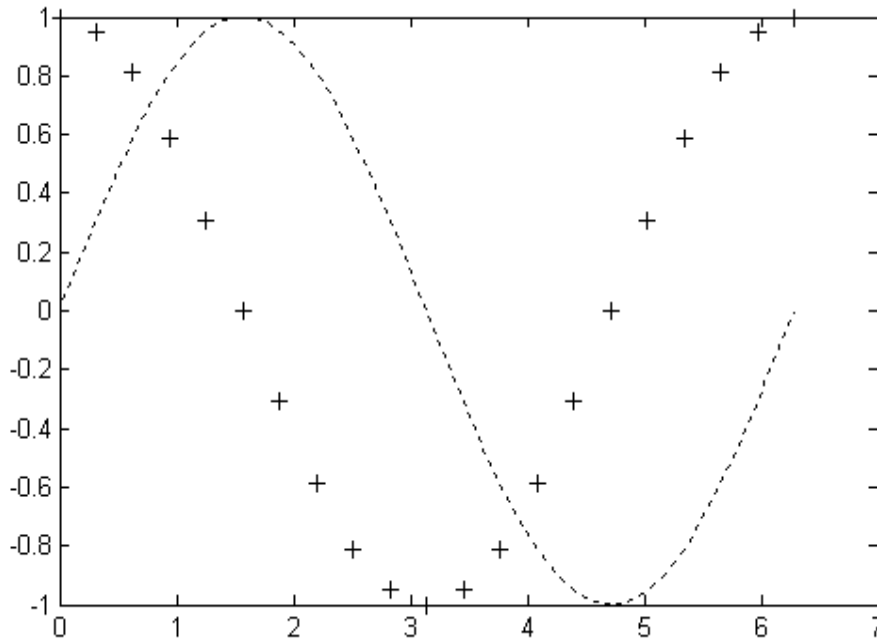

5-3   Adding Plots to an Existing Graph

The hold command enables you to add plots to an existing graph. When you type

hold on MATLAB does not replace the existing graph when you issue another plotting command; it adds the new data to the current graph, rescaling the axes if necessary.

Example:

```
x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
plot(x1,sin(x1),'r:')

hold on

plot(x2,cos(x2),'r+')
```
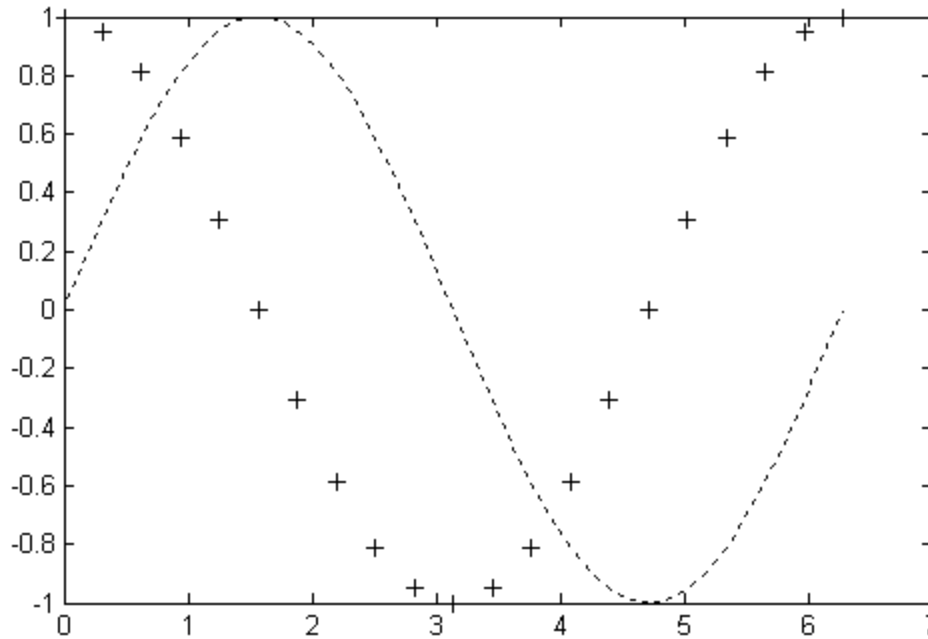
5-4   Multiple Plots in One Figure

The subplot command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing

```
subplot(m,n,p)
```

partitions the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot. The plots are numbered along first the top row of the figure window, then the second row, and so on. For example, these statements plot data in four different sub regions of the figure window.

```
t = 0:pi/10:2*pi;

x=sin(t); y=cos(t); z= 2*y-3*x; v=5-z;

subplot(2,2,1); plot(x)

subplot(2,2,2); plot(y)

subplot(2,2,3); plot(z)

subplot(2,2,4); plot(v)
```
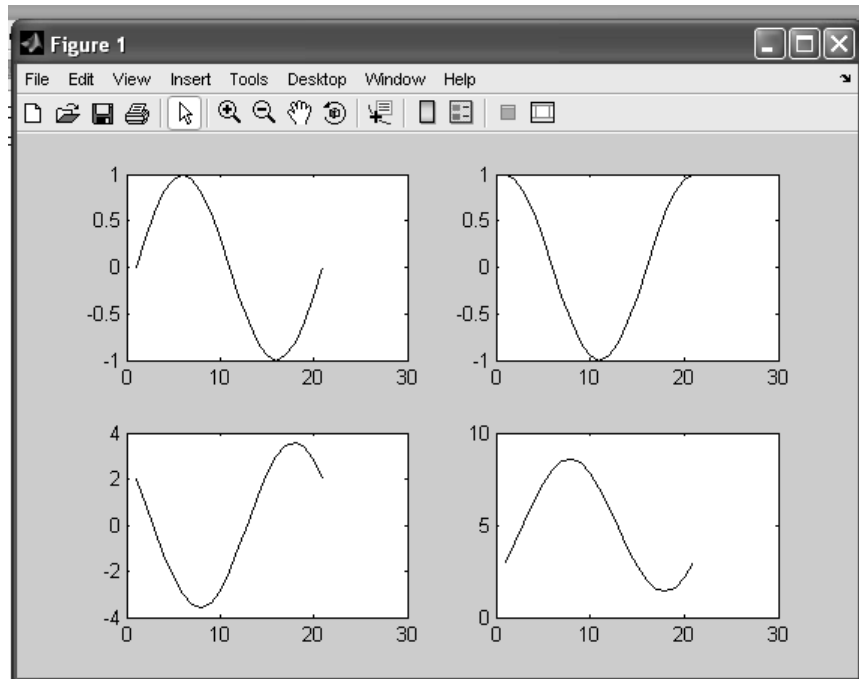
5-5   Setting Axis Limits

By default, MATLAB finds the maxima and minima of the data to choose the axis limits to span this range. The axis command enables you to specify your own limits

```
axis([xmin xmax ymin ymax])
```

5-6   Axis Labels and Titles

The **xlabel, ylabel,** and **zlabel** commands add *x*-, *y*-, and *z*-axis labels. The **title** command adds a title at the top of the figure and the **text** function inserts text anywhere in the figure.

Example:

```
t = -pi:pi/100:pi;

y = sin(t);

plot(t,y)

axis([-pi pi -1 1])
```

```
xlabel('-\pi to \pi')

ylabel('sin(t)')

title('Graph of the sine function')

text(1,-1/3,'Note the odd symmetry')
```

Graph of the sine function

Note the odd symmetry

-π to π
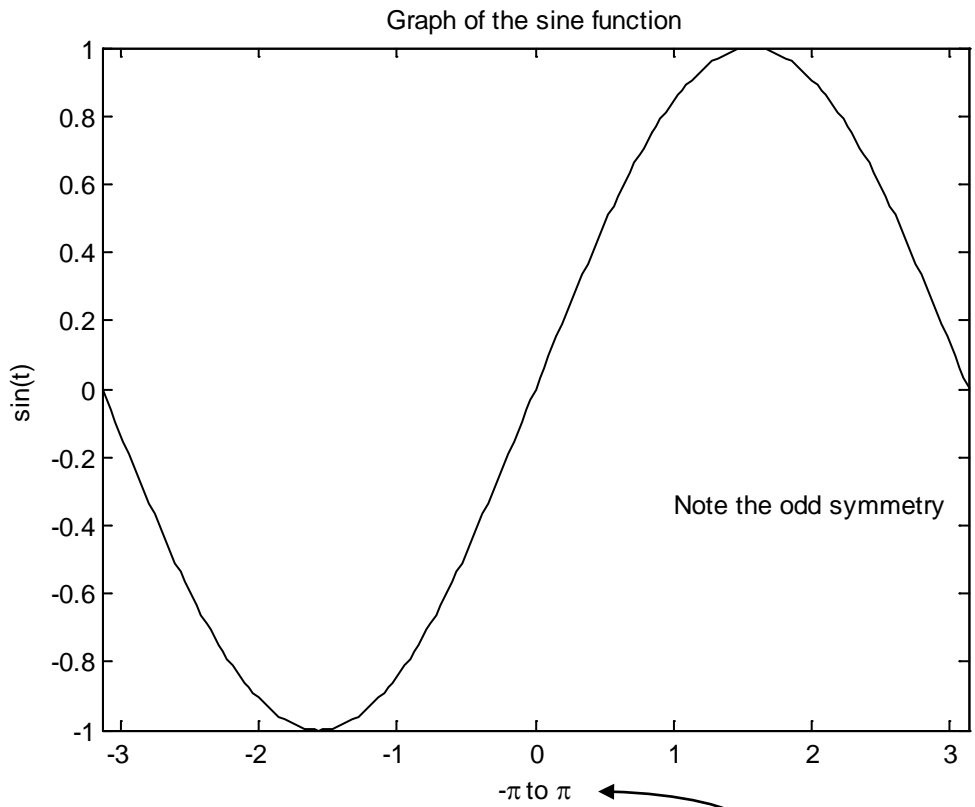
Notice the symbol

Exercises

1- Plot sawtooth waveform as shown below
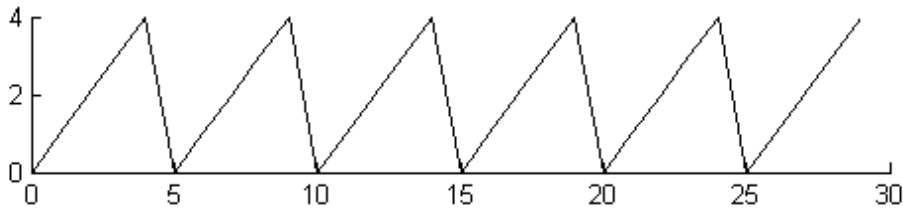
2- Plot Sinc function, where Sinc $(x) = \sin(x) / x$,   and $-2\pi \leq x \leq 2\pi$

3- Plot sin(x) and cos(x) on the same figure, then on the same axis using different colors.

4- if `y=sin(x)`, `z=cos(x)`, `v=exp(x)`,   where $-\pi \leq x \leq \pi$

could you plot **y, z, v** as shown below!
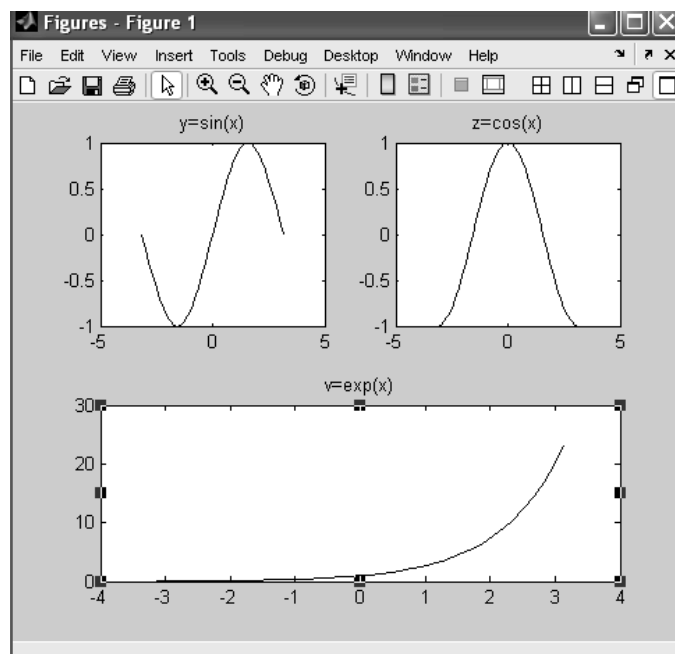
Experiment No. (6)
Complex and Statistical Functions

6-1   Complex Numbers

It is very easy to handle complex numbers in MATLAB. The special values i and j stand for $\sqrt{-1}$. Try sqrt(–1) to see how MATLAB represents complex numbers.

The symbol i may be used to assign complex values, for example,

>> z = 2 + 3*i

represents the complex number 2 + 3i (real part 2, imaginary part 3).

You can also input a complex value like this:

>> z=2 + 3i

The imaginary part of a complex number may also be entered without an asterisk (*), 3i.

You can also input a complex value like this:

>> z=complex(2,3)
z =
  2.0000 + 3.0000i

Example: Produce ten elements vector of random complex numbers and find the summation of this vector

```
>> x=rand(1,10);
>> y=rand(1,10);
>> z=x+i*y
>> sum(z)
```
Or

```
>> z=complex(x,y)
>> sum(z)
```

All of the arithmetic operators (and most functions) work with complex numbers, such as sqrt(2 + 3*i) and exp(i*pi). Some functions are specific to complex numbers, like:

Command	Description

>> A=3+4i
A =
   3.0000 + 4.0000i
 	
>> A = complex(3,4)
A =
   3.0000 + 4.0000i
 	
>> B = complex(-1,-3)
B =
  -1.0000 - 3.0000i
	Construct complex data from real and imaginary components

>> abs(A)
ans =
    5
 	
>> abs(B)
ans =
   3.1623
	Absolute value and complex magnitude

>> angle(A)
ans =
   0.9273
 	
>> angle(A)*180/pi
ans =
   53.1301
 	
>> angle(B)*180/pi
ans =
-108.4349
	Phase angle

>> conj(A)
ans =
   3.0000 - 4.0000i
>> conj(B)
ans =
  -1.0000 + 3.0000i
	Complex conjugate

Command	Description

>> real(A)
ans =
    3
>> real(B)
ans =
    -1
	Real part of complex number

>> imag(A)
ans =
    4
>> imag(B)
ans =
    -3
	Imaginary part of complex number

Example: Exchange the real and imaginary parts of the following matrix

$$A = \quad 0.8147 + 0.1576i \quad 0.9058 + 0.9706i \quad 0.1270 + 0.9572i$$
$$0.9134 + 0.4854i \quad 0.6324 + 0.8003i \quad 0.0975 + 0.1419i$$

>> A=[0.8147 + 0.1576i,   0.9058 + 0.9706i,   0.1270 + 0.9572i
    0.9134 + 0.4854i,   0.6324 + 0.8003i,   0.0975 + 0.1419i];

>> x=real(A);
>> y=imag(A);
>> a=x;
>> x=y;
>> y=a;
>> A=x+i*y

A =

  0.1576 + 0.8147i   0.9706 + 0.9058i   0.9572 + 0.1270i
  0.4854 + 0.9134i   0.8003 + 0.6324i   0.1419 + 0.0975i

6-2   <u>Statistical Functions</u>

Function	Description
mean(x)	Average or mean value of array (x)
median(x)	Median value of array (x)
mode(x)	Most frequent values in array (x) When there are multiple values occurring equally frequently, mode returns the smallest of those values. For complex inputs, this is taken to be the first value in a sorted list of values.

std(x)	returns the standard deviation of array (x)
	
	 s = std(x) = std(x,0) and it is equal to
	
	 $$s = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right)^{\frac{1}{2}}$$
	
	 $$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$
	
	 s = std(x,1) , it is equal to
	
	 $$s = \left(\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right)^{\frac{1}{2}}$$
	
	 >> x =[1 5 9
	          7 15 22];
	
	 >> s = std(x,0)
	
	 s =
	
	    4.2426 7.0711 9.1924
	
	 >> s = std(x,1)
	
	 s =
	
	    3.0000 5.0000 6.5000

Function	Description

var(x)	Returns the variance of array (x), The variance is the square of the standard deviation (STD).
	
	 s = var(x,0) when the summation normalized by N-1
	
	 s = var(x,1) when the summation normalized by N
	
	 >> x = [1 5 9
	
	        7 15 22];
	
	 >> s = var(x,0)
	
	 s =
	
	   18.0000 50.0000 84.5000
	
	 >> s = var(x,1)
	
	 s =
	
	    9.0000 25.0000 42.2500

Example: If X = 3 3 1 4
                    0 0 1 1
                    0 1 2 4

  Then
 >>mode(x)

    ans=   [0 0 1 4]

and

>> mode(X,2)

    ans=   [3 ; 0 ; 0]

Example: If A = 1 2 4 4
                    3 4 6 6
                    5 6 8 8
                    5 6 8 8

  >> median(A)
      ans= [4 5 7 7]
 >> median(A,2)
      ans= [3 ; 5 ; 7 ; 7]

Exercises

   1- Represent the following complex numbers in polar coordinate

      Z= 2 + 5j
      Y= -3 - 3j
      D= -2 + 6j

   2- Find the conjugate of the numbers above
   3- Represent the following numbers in rectangular coordinate

      W= 5∟ 30o
      A= 2.5∟ -20o
      Q= 3e1.5∟ -73o

4- Compute the standard deviation by using the following equations then compare the result with that one obtained by **std** command

$$s = \left(\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right)^{\frac{1}{2}}$$

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

5- Write a program to compute the most frequent numbers in vectors (x), and (y)
   if
   x= a*b
   y=a* c
   a = [1 3]
   b = [2 3 5 ; 4 7 8]
   b = [2 3 3 ; 4 7 7]

Experiment No. (7)
Input / Output of Variables
(Numbers and Strings)

7-1   Characters and Text

Enter text into MATLAB using single quotes. For example,

```
>> S = 'Hello'
```

The result is not the same kind of numeric matrix or array we have been dealing with up to now. The string is actually a vector whose components are the numeric codes for the characters (the first 127 codes in ASCII). The length of S is the number of characters. It is a 1-by-5 character array. A quote within the string is indicated by two quotes.

Concatenation with square brackets joins text variables together into larger strings. For example,
```
>> h = ['MAT', 'LAB']
```
joins the strings horizontally and produces
```
h =
     MATLAB
```
and the statement
```
>> v = ['MAT'; 'LAB']
```
joins the strings vertically and produces
```
v =
     MAT
     LAB
```

Note that both words in v have to have the same length. The resulting arrays are both character arrays; h is 1-by-6 and v is 2-by-3.

7-1-1 Some String Function

Function	Description

`char (x)`
 	
`>> char(100)`
`    ans =`
`        d`
`>> char([73 82 65 81])`
`    ans =`
`        IRAQ`
	converts the array x that contains positive integers representing character codes into a MATLAB character array (the first 127 codes in ASCII).

Function	Description

double(s)
 	
`>> double('z')`
`    ans =`
`        122`
`>> double('ali')`
`    ans =`
`        97    108    105`
	converts the character array to a numeric matrix containing floating point representations of the ASCII codes for each character.

strcat(S1,S2,...)
 	
`>>strcat('Hello',' Ali')`
`    ans =`
`        Hello Ali`
	joins S1,S2,...variables horizontally together into larger string.

strvcat(S1,S2,...)
 	
`>> strvcat ('Hello', 'Hi', 'Bye')`
`    ans =`
`        Hello`
`        Hi`
`        Bye`
	joins S1,S2,... variables vertically together into larger string.

s = num2str(x)
 	
`>> num2str(20)`
`ans =`
`20       % as a string,`
`        not a number`
	converts the variable x into a string representation s.

x = str2num(s)
 	
`>> str2num('20')`
`ans =`
`20`
	converts character array representation of a matrix of numbers to a numeric matrix.

error (*Msg*)	displays the error message in the string (*Msg*), and causes an error exit from the currently executing M-file to the keyboard.
lower(A)	Converts any uppercase characters in **A** to the corresponding lowercase character and leaves all other characters unchanged.

Function	Description
`upper(x)`	Converts any lower case characters in **A** to the corresponding upper case character and leaves all other characters unchanged

Note

The printable characters in the basic ASCII character set are represented by the integers 32:127. (The integers less than 32 represent nonprintable control characters). Try

```
>> char(33:127)
```

7-2 Input of Variable

- To enter matrix or vector or single element:

```
>> x=input('parameter=   ')
parameter=   2
x =
     2
>> x=input('parameter=   ')
parameter=   [2 4 6]
x = 2    4    6
>> x=input('parameter=   ')
parameter=   [1 2 3;4 5 6]
x =  1     2     3
     4     5     6
```

- To enter text:

```
>> x=input('parameter=   ')
parameter=   'faaz'
x =
       faaz
>> x=input('parameter=   ' , 's' )
parameter=   faaz
x =        faaz
```

Notice the difference between the two statements

7-3   Output of Variable

- **disp (x)**

displays the array (x), without printing the array name.   In all other ways it's the same as leaving the semicolon off an expression except that empty arrays don't display.

If (x) is a string, the text is displayed.

```
>> x=[1 2 3];

>> x

x =

     1     2     3

>> disp(x)

     1     2     3
```

Example:

```
>> a=6;
>> b=a;
>> s='Ahmed has ';
>> w='Ali has  ';
>> t=' Dinars';
>>disp([ s  num2str(a)  t]);
>>disp([ w  num2str(b)  t]);
```

the execution result is:

```
Ahmed has 6 Dinars
Ali has  6 Dinars
```

7-4   M-File:

An M-File is an external file that contains a sequence of MATLAB statements. By typing the filename in the **Command Window**, subsequent MATLAB input is

obtained from the file. M-Files have a filename extension of " **.m** " and can be created or modified by using **Editor/Debugger Window**.

7-4-1   Script Files

You need to save the program if you want to use it again later. To save the contents of the Editor, select **File → Save** from the **Editor** menu bar. Under **Save** file as, select a directory and enter a *filename*, which **must have** the extension **.m**, in the File name: box (e.g., faez.m). Click **Save**. The Editor window now has the title faez.m If you make subsequent changes to faez.m an **asterisk** appears next to its name at the top of the Editor until you save the changes.

A MATLAB program saved from the Editor with the extension .m is called a script file, or simply a script. (MATLAB function files also have the extension .m. We therefore refer to both script and function files generally as M-files.).

The special significances of a script file are that:-
1- if you enter its name at the command-line prompt, MATLAB carries out each statement in it as if it were entered at the prompt.
2- Scripts M-file does not accept input arguments or return output arguments. They operate on data in the workspace.
3- The rules for script file names are the same as those for MATLAB variable names.

7-4-2   Function Files

MATLAB enables you to create your own function M-files. A function M-file is similar to a script file in that it also has an .m extension. However, it differs from a script file in that it communicates with the MATLAB workspace only through specially designated input and output arguments.

 * Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

General form of a function: A function M-file *filename*.m has the following general form:

function [*outarg1*, *outarg2*,...] = *filename* (*inarg1*, *inarg2*,...)

% comments to be displayed with **help**

...
outarg1 = ... ;
outarg2 = ... ;

Note:

inarg1, inarg2,...   are the input variables to the function ***filename.m***

outarg1, outarg2,... are the output variables from the function ***filename.m***

function   The function file must start with the keyword function (in the function definition line).

Example:

sphecart.m   *((function))*

function [x,y,z] = **sphecart**(r,theta,rho)

%conversion from spherical to Cartesian coordinates
x = r*cos(rho)*cos(theta);
y = r*cos(rho)*sin(theta);
z = r*sin(rho);

sphecart.m   *((script file))*

%conversion from spherical to Cartesian coordinates

x = r*cos(rho)*cos(theta);
y = r*cos(rho)*sin(theta);
z = r*sin(rho);
%the values of r, rho, theta are be obtained from workspace of command window

Exercises:

1- If x=[1 5 9; 2 7 4], then
   a) display the last two elements by using **disp** command.
   b) display the sum of each row as show below
```
        The sum of 1st row  =
        The sum of 2nd row  =
```

2- Write a program in M-File to read 3 x 3 Matrix, then display the diagonal of matrix as shown below:

```
 The Diagonal of This Matrix = [            ]
```

3- Write a program to read a string, then replace each character in the string with its following character in ASCII code*.

4- The Table shown below lists the degrees of three students, Write a program in M-file to read these degrees and calculate the average degree for each student.

Name	Mathematics	Electric Circuits	Communication
Ahmed	80	80	80
Waleed	75	80	70
Hasan	80	90	85

Then display results as shown below

```
Name                        Degree
----------------------------
Ahmed                       80
Waleed                      75
Hasan                       85
```

5- Write a group of statements that carry out the same action of **upper** and **lower** functions.

---

* This operation is called *Caesar cipher*.

Experiment No. (8)
Flow Control

Computer programming languages offer features that allow you to control the flow of command execution based on decision making structures. MATLAB has several flow control constructions:

- if statement.
- switch and case statement.
- for statement.
- while statement.
- break statement.

8-1   if statement

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 general form of **if** statement is:

```
if expression 1
     group of statements 1

elseif expression 2
     group of statements 2

else expression 3
     group of statements 3

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 legal 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. The proper way to check for equality between two matrix is to use the **isequal** function,

```
if isequal(A,B)
```

Example:

```
A=input('A=');
B=input('B=');
if A > B
'greater'
elseif A < B
'less'
elseif A == B
'equal'
else
error ('Unexpected situation')
end
```

8-2   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. If the first case statement is true, the other case statements do not execute.

The general form of **switch** statement is:

```
switch expression
case 0
    statements 0
case 1
    statements 1

otherwise
    statements 3
end
```

Example

```
method = 'Bilinear';

switch lower(method)
case 'bilinear'
  disp('Method is bilinear')
 case 'cubic'
   disp('Method is cubic')
 case 'nearest'
   disp('Method is nearest')
 otherwise
   disp('Unknown method.')
 end
```

Execution result is:

```
            Method is bilinear
```

8-3   for statement

The **for** loop repeats a group of statements a fixed, predetermined number of times.

The general form of **for** statement is:

```
for variable = initial value: step size: final value
    statement
    . . .

    statement

end
```

Example:

```
for i=1:5
for k=5:-1:1
m(i,k)=i*k;
end
end

>> m
```

```
m =

    1     2     3     4     5
    2     4     6     8    10
    3     6     9    12    15
    4     8    12    16    20
    5    10    15    20    25
```

A for loop cannot be terminated by reassigning the loop variable within the for loop:

```
for i=1:10
x(i)=sin (pi/i);
i=10;                    % this step do not effect on the for loop
end
x
i
```

Execution results are,
```
x=
    0.0000    1.0000    0.8660    0.7071    0.5878    0.5000
    0.4339    0.3827    0.3420    0.3090
i=
    10
```

8-4   <u>while statement</u>

repeat statements an indefinite number of times under control of a logical condition. A matching **end** delineates the statements.

The general form of **while** statement is:

```
while expression
    statement
    ...
    statement
end
```

Example: Here is a complete program, illustrating `while`, `if`, `else`, and `end`, that uses interval bisection method to find a zero of a polynomial.

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
    x = (a+b)/2;
    fx = x^3-2*x-5;
    if sign(fx) == sign(fa)
        a = x; fa = fx;
    else
        b = x; fb = fx;
    end
end
```

8-5   <u>break</u>

Terminate execution of while or for loop. In nested loops, break exits from the innermost loop only. If **break** is executed in an IF, SWITCH-CASE statement, it terminates the statement at that point.

8-6   <u>Continue</u>

 passes control to the next iteration of FOR or WHILE loop in which it appears, skipping any remaining statements in the body of the FOR or WHILE loop.

Example:

we can modify the previous example by using **break** command.

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
    x = (a+b)/2;
    fx = x^3-2*x-5;
    if fx == 0
        break
    elseif sign(fx) == sign(fa)
        a = x; fa = fx;
    else
        b = x; fb = fx;
    end
end
```

Example: Without using the **max** command, find the maximum value of matrix (a) where

    a =[11 3 14;8 6 2;10 13 1]

Solution

```
a=[11 3 14;8 6 2;10 13 1]
temp=a(1);
[n,m]=size(a);
for i=1:n
    for j=1:m
        if a(i,j)>temp
            temp=a(i,j);
        end
    end
end
temp
```

the execution result is   **14**

Example: Let x=[2 6; 1 8], y=[.8 -0.3 ; -0.1 0.2], prove that y is not the inverse matrix of x.

Solution

```
z=inv(x);
if ~isequal(z,y)
disp(' y is not the inverse matrix of x ')
end
```

the execution result is

    y is not the inverse matrix of x

Exercises

1- The value of s could be calculated from the equation below:

$$s = \begin{cases} \sqrt{y^2 - 4xz} & \text{if } y \geq 4xz \\ \inf & \text{if } y < 4xz \end{cases}$$

write a MATLAB program in M-File to do the following steps:-

a)   input the value of x, y, z
b)   caluclate s
c)   print the output as shown below

x = . . .
y = . . .
z = . . .
s = . . .

2- Write a program to find the current I in the circuit shown below
   a)   By using conditional statements.
   b)   Without using any conditional statements.
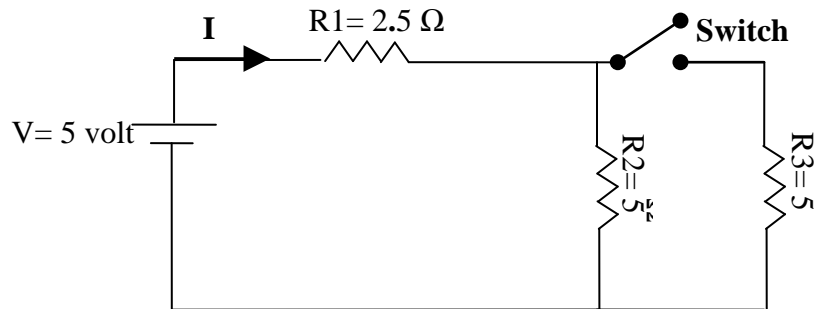
Experiment No. (9)
MATLAB Simulink Basic

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.

Simulink is supported on Unix, Macintosh, and Windows environments, and it is included in the student version of MATLAB for personal computers. For more information on Simulink, contact the MathWorks.

The idea behind these tutorials is that you can view them in one window while running Simulink in another window. Do not confuse the windows, icons, and menus in the tutorials for your actual Simulink windows. Most images in these tutorials are not live - they simply display what you should see in your own Simulink windows. All Simulink operations should be done in your Simulink windows.

9 - 1   Starting Simulink

Simulink is started from the MATLAB command prompt by entering the following command:simulink
Alternatively, you can click on the "Simulink Library Browser" button at the top of the MATLAB command window as shown below:
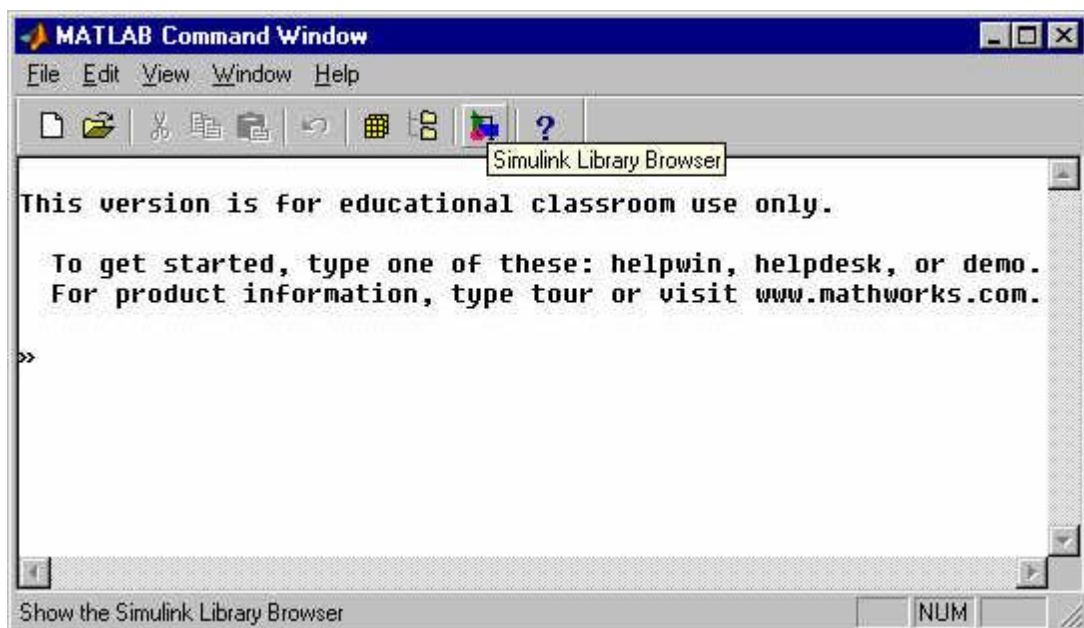
The Simulink Library Browser window should now appear on the screen. Most of the blocks needed for modeling basic systems can be found in the subfolders of the main "Simulink" folder (opened by clicking on the "+" in front of "Simulink"). Once the "Simulink" folder has been opened, the Library Browser window should look like:
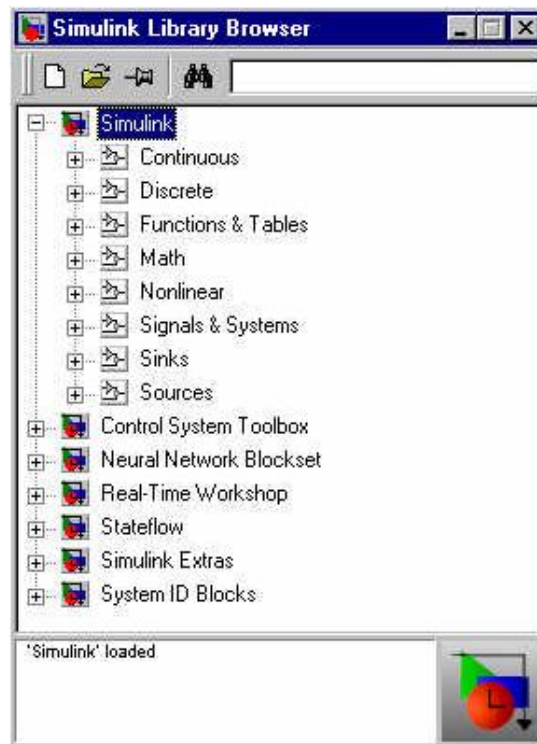
9 - 2   Basic Elements

There are two major classes of elements in Simulink: **blocks** and **lines**. Blocks are used to generate, modify, combine, output, and display signals. Lines are used to transfer signals from one block to another.

Blocks

The subfolders underneath the "Simulink" folder indicate the general classes of blocks available for us to use:

- Continuous: Linear, continuous-time system elements (integrators, transfer   functions, state - space   models, etc.)
- Discrete: Linear, discrete-time system elements (integrators, transfer functions, state - space models, etc.)
- Functions & Tables: User-defined functions and tables for interpolating function values
- Math: Mathematical operators (sum, gain, dot product, etc.)
- Nonlinear: Nonlinear operators (coulomb/viscous friction, switches, relays, etc.)
- Signals & Systems: Blocks for controlling/monitoring signal(s) and for creating subsystems
- Sinks: Used to output or display signals (displays, scopes, graphs, etc.)
- Sources: Used to generate various signals (step, ramp, sinusoidal, etc.)

Blocks have zero to several input terminals and zero to several output terminals. Unused input terminals are indicated by a small open triangle. Unused output terminals are indicated by a small triangular point. The block shown below has an unused input terminal on the left and an unused output terminal on the right.
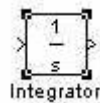
Lines

Lines transmit signals in the direction indicated by the arrow. Lines must always transmit signals from the output terminal of one block to the input terminal of another block. One exception to this is that a line can tap off of another line. This sends the original signal to each of two (or more) destination blocks, as shown below:
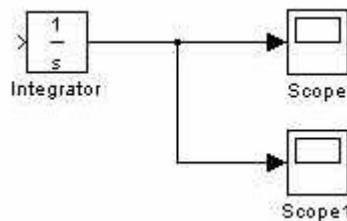
Lines can never inject a signal *into* another line; lines must be combined through the use of a block such as a summing junction.

A signal can be either a scalar signal or a vector signal. For Single-Input, Single-Output systems, scalar signals are generally used. For Multi-Input, Multi-Output systems, vector signals are often used, consisting of two or more scalar signals. The lines used to transmit scalar and vector signals are identical. The type of signal carried by a line is determined by the blocks on either end of the line.

9 - 3 Building a System

To demonstrate how a system is represented using Simulink, we will build the block diagram for a simple model consisting of a sinusoidal input multiplied by a constant gain, which is shown below:
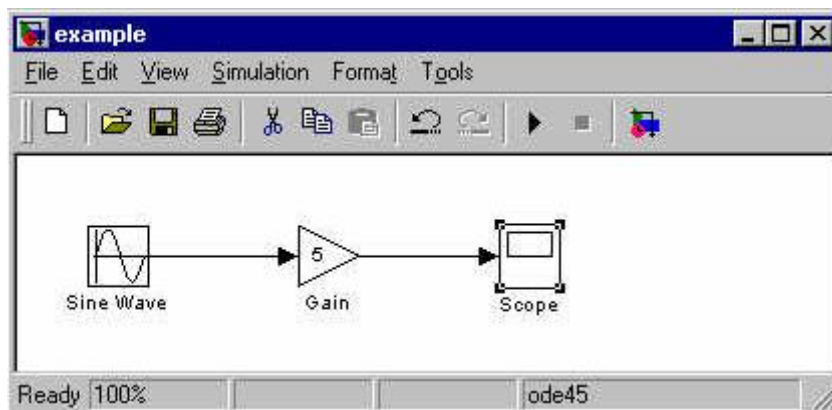
This model will consist of three blocks: Sine Wave, Gain, and Scope. The Sine Wave is a **Source Block** from which a sinusoidal input signal originates. This signal is transferred through a **line** in the direction indicated by the arrow to the Gain **Math Block**. The Gain block modifies its input signal (multiplies it by a constant value) and outputs a new signal through a **line** to the Scope block. The Scope is a **Sink Block** used to display a signal (much like an oscilloscope).

We begin building our system by bringing up a new model window in which to create the block diagram. This is done by clicking on the "New Model" button in the toolbar of the Simulink Library Browser (looks like a blank page).

Building the system model is then accomplished through a series of steps:

1. The necessary blocks are gathered from the Library Browser and placed in the model window.
2. The parameters of the blocks are then modified to correspond with the system we are modelling.
3. Finally, the blocks are connected with lines to complete the model.

Each of these steps will be explained in detail using our example system. Once a system is built, simulations are run to analyze its behavior.

9 - 4 Gathering Blocks

Each of the blocks we will use in our example model will be taken from the Simulink Library Browser. To place the Sine Wave block into the model window, follow these steps:

1. Click on the "+" in front of "Sources" (this is a subfolder beneath the "Simulink" folder) to display the various source blocks available for us to use.
2. Scroll down until you see the "Sine Wave" block. Clicking on this will display a short explanation of what that block does in the space below the folder list:
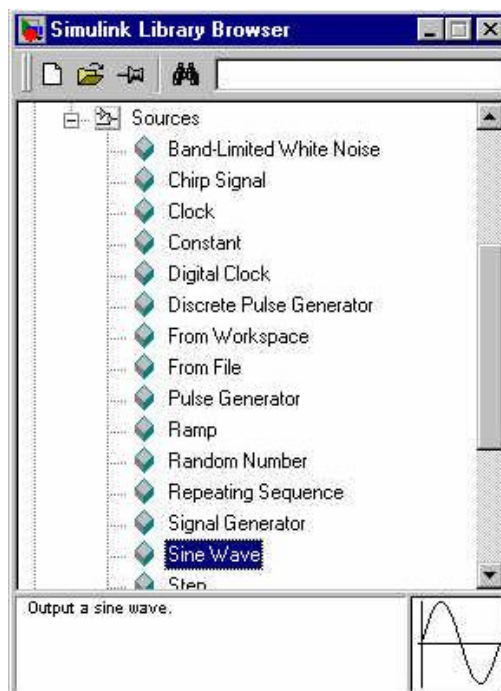

 3. To insert a Sine Wave block into your model window, click on it in the Library Browser and drag the block into your workspace.

The same method can be used to place the Gain and Scope blocks in the model window. The "Gain" block can be found in the "Math" subfolder and the "Scope" block is located in the "Sink" subfolder. Arrange the three blocks in the workspace (done by selecting and dragging an individual block to a new location) so that they look similar to the following:
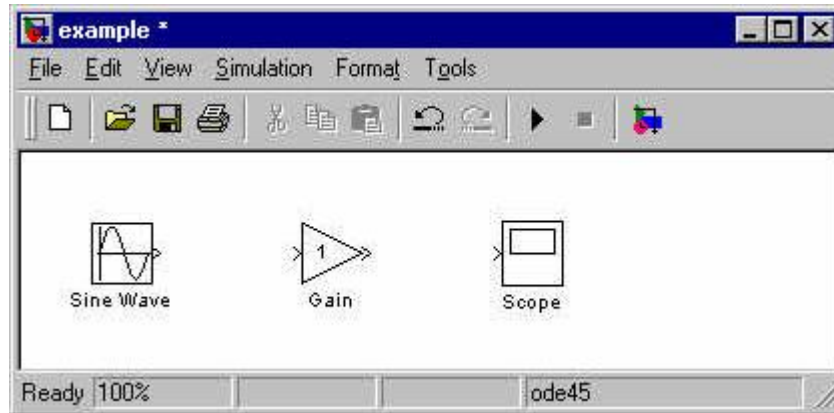
9 - 5 Modifying the Blocks

Simulink allows us to modify the blocks in our model so that they accurately reflect the characteristics of the system we are analyzing. For example, we can modify the Sine Wave block by double-clicking on it. Doing so will cause the following window to appear:
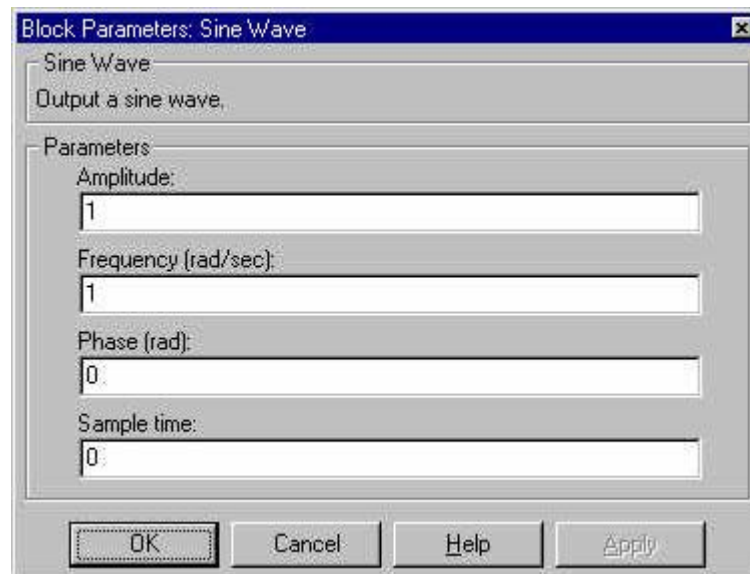
This window allows us to adjust the amplitude, frequency, and phase shift of the sinusoidal input. The "Sample time" value indicates the time interval between successive readings of the signal. Setting this value to 0 indicates the signal is sampled continuously.

Let us assume that our system's sinusoidal input has:

- Amplitude = 2

- Frequency = pi

- Phase = pi/2

Enter these values into the appropriate fields (leave the "Sample time" set to 0) and click "OK" to accept them and exit the window. Note that the frequency and phase for our system contain 'pi' (3.1415...). These values can be entered into Simulink just as they have been shown.

Next, we modify the Gain block by double-clicking on it in the model window. The following window will then appear:
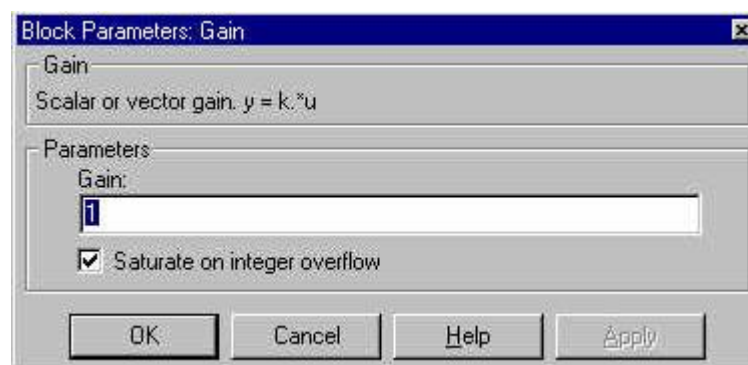
Note that Simulink gives a brief explanation of the block's function in the top portion of this window. In the case of the Gain block, the signal input to the block (u) is multiplied by a constant (k) to create the block's output signal (y). Changing the "Gain" parameter in this window changes the value of k.

For our system, we will let k = 5. Enter this value in the "Gain" field, and click "OK" to close the window.

The Scope block simply plots its input signal as a function of time, and thus there are no system parameters that we can change for it. We will look at the Scope block in more detail after we have run our simulation.

9 - 6 Connecting the Blocks

For a block diagram to accurately reflect the system we are modeling, the Simulink blocks must be properly connected. In our example system, the signal output by the Sine Wave block is transmitted to the Gain block. The Gain block amplifies this signal and outputs its new value to the Scope block, which graphs the signal as a function of time. Thus, we need to draw lines from the output of the Sine Wave block to the input of the Gain block, and from the output of the Gain block to the input of the Scope block.

Lines are drawn by dragging the mouse from where a signal starts (output terminal of a block) to where it ends (input terminal of another block). When drawing lines, it is important to make sure that the signal reaches each of its intended terminals.   Simulink will turn the mouse pointer into a crosshair when it is close enough to an output terminal to begin drawing a line, and the pointer will change into a double crosshair when it is close enough to snap to an input terminal. A signal is
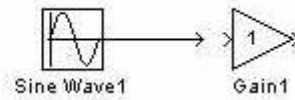
properly connected if its arrowhead is filled in. If the arrowhead is open, it means the signal is not connected to both blocks. To fix an open signal, you can treat the open arrowhead as an output terminal and continue drawing the line to an input terminal in the same manner as explained before.
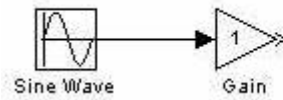

**Properly Connected Signal**   **Open Signal**

When drawing lines, you do not need to worry about the path you follow. The lines will route themselves automatically. Once blocks are connected, they can be repositioned for a neater appearance. This is done by clicking on and dragging each block to its desired location (signals will stay properly connected and will re-route themselves).

After drawing in the lines and repositioning the blocks, the example system model should look like:
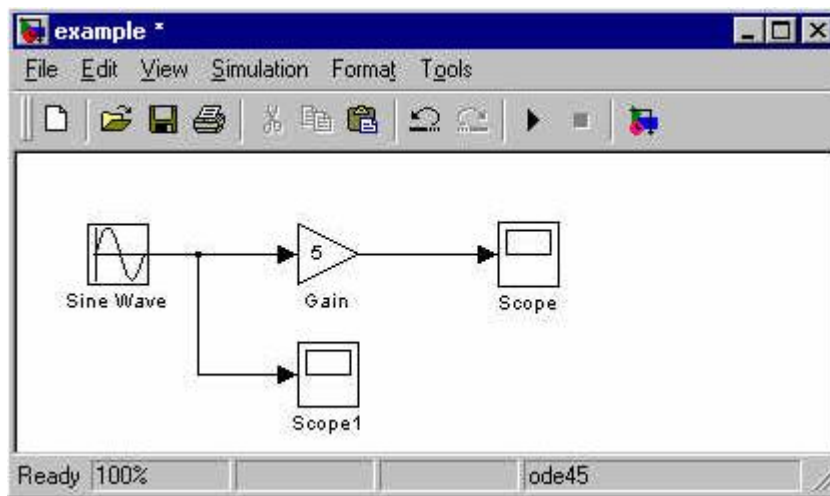
In some models, it will be necessary to branch a signal so that it is transmitted to two or more different input terminals. This is done by first placing the mouse cursor at the location where the signal is to branch. Then, using either the CTRL key in conjunction with the left mouse button or just the right mouse button, drag the new line to its intended destination. This method was used to construct the branch in the Sine Wave output signal shown below:
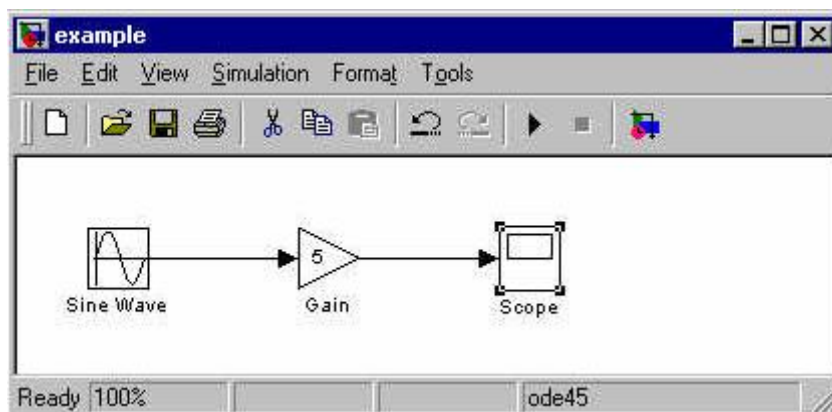
The routing of lines and the location of branches can be changed by dragging them to their desired new position. To delete an incorrectly drawn line, simply click on it to select it, and hit the DELETE key.

9 – 7 Running Simulations

Now that our model has been constructed, we are ready to simulate the system. To do this, go to the **Simulation** menu and click on **Start** , or just click on the "Start/Pause Simulation" button in the model window toolbar (looks like the "Play" button on a VCR). Because our example is a relatively simple model, its simulation runs almost instantaneously. With more complicated systems, however, you will be able to see the progress of the simulation by observing its running time in the the lower box of the model window. Double-click the Scope block to view the output of the Gain block for the simulation as a function of time. Once the Scope window appears, click the "Autoscale" button in its toolbar (looks like a pair of binoculars) to scale the graph to better fit the window. Having done this, you should see the following:
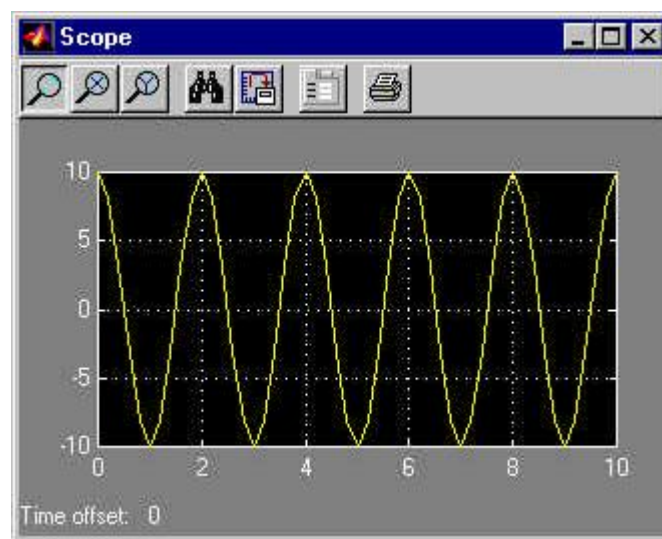
Note that the output of our system appears as a cosine curve with a period of 2 seconds and amplitude equal to 10. Does this result agree with the system parameters we set?   Its amplitude makes sense when we consider that the amplitude of the input signal was 2 and the constant gain of the system was 5 ($2 \times 5 = 10$). The output's period should be the same as that of the input signal, and this value is a function of the frequency we entered for the Sine Wave block (which was set equal to pi). Finally, the output's shape as a cosine curve is due to the phase value of pi/2 we set for the input (sine and cosine graphs differ by a phase shift of pi/2).

What if we were to modify the gain of the system to be 0.5? How would this affect the output of the Gain block as observed by the Scope? Make this change by double-clicking on the Gain block and changing the gain value to 0.5. Then, re-run the simulation and view the Scope (the Scope graph will not change unless the simulation is re-run, even though the gain value has been modified). The Scope graph should now look like the following:

Note that the only difference between this output and the one from our original system is the amplitude of the cosine curve. In the second case, the amplitude is equal to 1, or 1/10th of 10, which is a result of the gain value being 1/10th as large as it originally was.