

University of Technology  
الجامعة التكنولوجية



Computer Science Department  
قسم علوم الحاسوب

Mathematics Lab  
الرياضيات - عملي

Dr. Khitam A. Salman  
د. ختام عبد النبي سلمان



[cs.uotechnology.edu.iq](http://cs.uotechnology.edu.iq)

# Introduction to MATLAB

**MATLAB** (**MAT**rix **LAB**oratory) is an interactive software system for numerical computations and graphics. As the name suggests, MATLAB is especially designed for matrix computations, solving systems of linear equations, computing eigenvalues and eigenvectors, factorization of matrices, and so much.

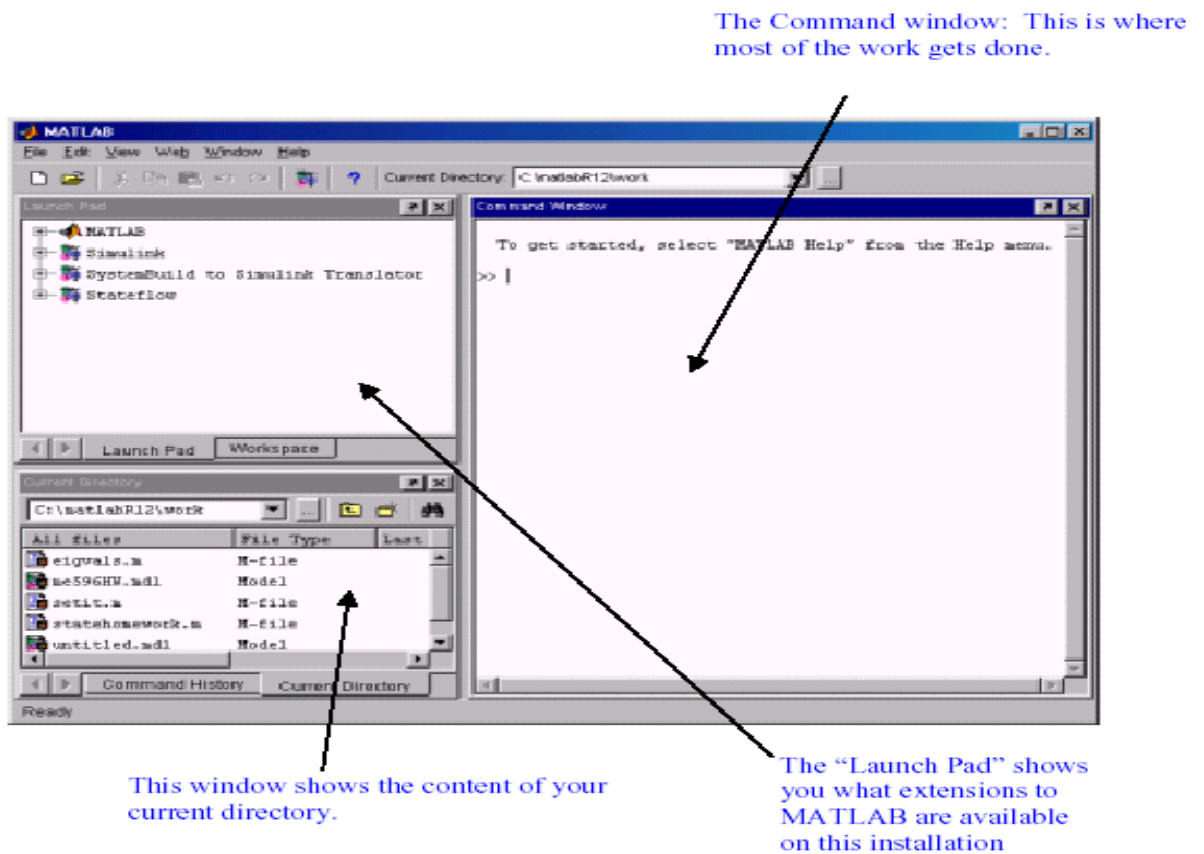
In addition, it has a variety of graphical capabilities, and can be extended through programs written in its own programming language. Many such programs come with the system; these extend MATLAB's capabilities to a wide range of applications, like solution of nonlinear systems of equations, integration of ordinary and partial differential equations, and many others.

MATLAB widely used in the engineering. It has many features and it could take years to learn all of its capabilities. However, it's basic functionality is quite easy to learn and in the course of the next few weeks we will be learning how to manipulate and graph data in MATLAB as well as writing simple programs to manipulate data. It offers a powerful programming language, excellent graphics, and a wide range of expert knowledge.

## **1. Getting Started**

Start Matlab by double clicking the icon on the desktop, or from the start menu. To use Matlab you can simply enter commands after the prompt (the >> is the Matlab prompt).

Figure 1 below shows the default frame with the three standard Matlab windows.

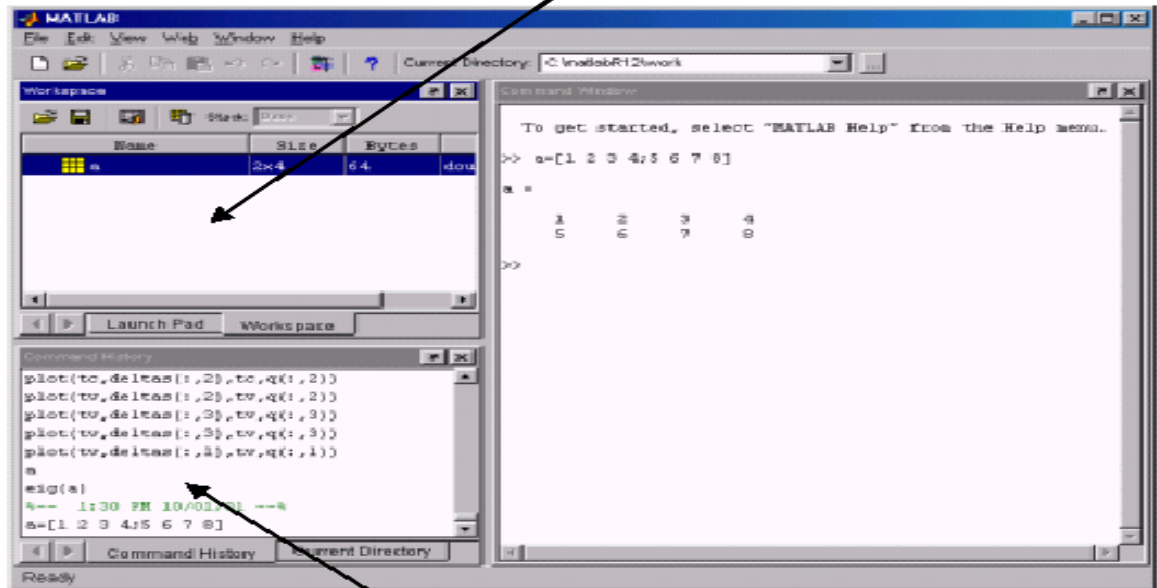


**Figure 1: The MATLAB Window**

### **1.1 Alternate windows:**

The smaller of the two windows is alternate windows that can be accessed by clicking on the tabs. Figure 2 shows the alternate windows and describes their functions.

The Workspace Browser lets you examine what you've stored in your workspace



The "Command History" lets you recall previously entered commands.

Figure 2: Alternate windows in the default frame

## 1.2 The command window:

The command window is the active window immediately appears after launching Matlab. One enters Matlab commands after the ">>" prompt and presses enter to execute the command. To recall the last commands entered, simply press the up or down arrows; one can edit the commands before executing them. Multiple commands may be entered on one line separated by commas. Separating commands by a semi-colon suppresses output to the command window.

This window allows a user to enter simple commands. To perform a simple computations type a command and next press the Enter or Return key. For instance

```
>> s = 1 + 2  
s =  
3
```

Note that the results of these computations are saved in variables whose names are chosen by the user. If you need to obtain their values again, type their names and pressing Enter key. If you type again:

```
>> s
s =
    3
```

Only for short computations it is useful to execute Matlab straightaway from the command line. The Editor Window is a word processor specifically designed for MATLAB commands. Files that written in this window are called the m-files. Another way to do calculations in MATLAB is to create an m-file with a series of commands and then to run some or all of the commands in that file. To create an m-file, click file, new and then m-files. The same statements that are entered in the command window can also be used in an m file. You can also copy a command you try out in the command window into an m file by using the copy and paste functions on the computer. Save the file under a name that ends in .m by clicking file and using “save as” icon. (See Figure 3)

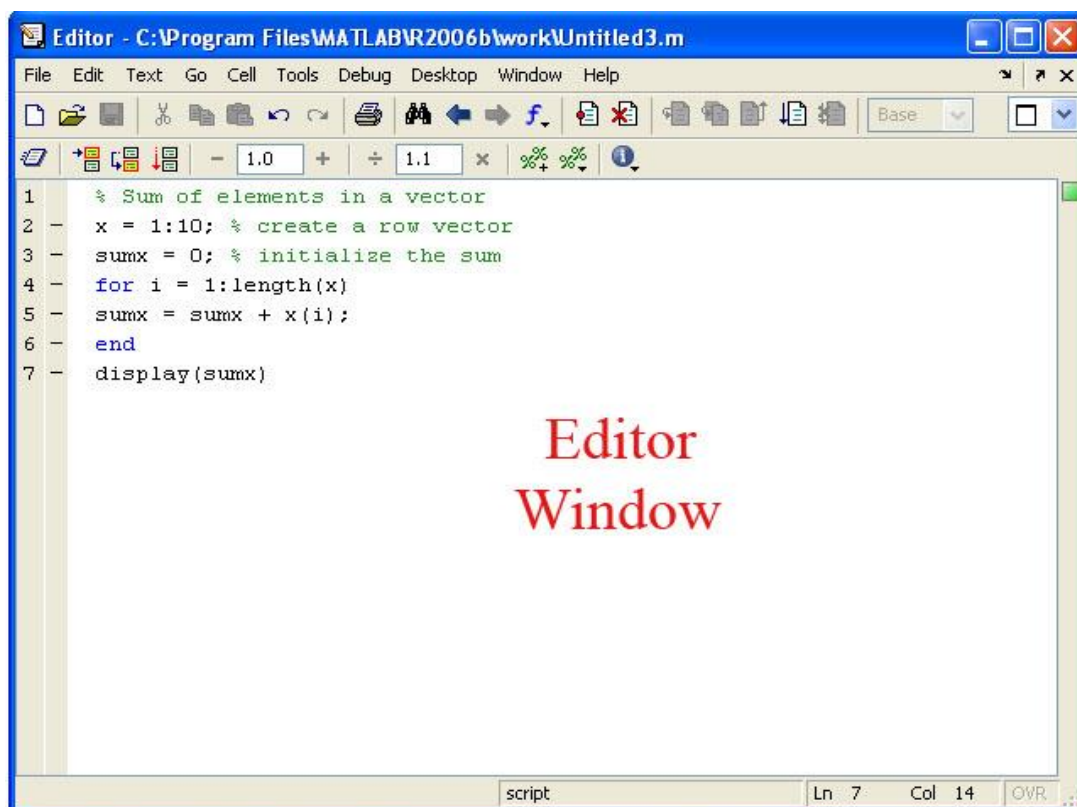


Figure 3: MATLAB Editor Window

### **1.3 Function Files**

Function files are a special kind of script file (M-file) that allow you to define your own functions for use during a Matlab session. You might think of them as subroutines that can be called from within a script, or even called directly from the command line. Many of the "built-in" functions in Matlab are actually stored as M-files as part of the Matlab package. Function files are created and edited in identically the same manner as the script files.

### **2. Numbers, Arithmetic Operations and Special Characters**

There are three kinds of numbers used in MATLAB:

- integers
- real numbers
- complex numbers

In addition to these, MATLAB has three variables representing non-numbers:

(-Inf, Inf, NaN)

The -Inf and Inf are the negative and positive infinity respectively. Infinity is generated by overflow or by the operation of dividing by zero. The NaN stands for Not-A-Number and it is obtained as a result of the mathematically undefined operations such as 0.0/0.0.

The list of basic arithmetic operations in MATLAB includes six operations:

+ : addition

- : subtraction

\* : multiplication

/ : right division

\ : left division

^ : power

One can use Matlab like a calculator without specifying variables. Matlab has all the standard mathematical operations. Try type:

```
>> 2+3
```

Matlab returns the answer:

```
ans=
```

```
5
```

```
>> 5^3
```

```
ans=  
  125  
>>3.89*4.1  
ans =  
  15.9490  
>>99.3-25  
ans =  
  74.3000  
>> 3*(23+14.7-4/6)/3.5  
ans=  
  31.7429
```

The result of these operations is assigned to a default variable called `ans` and displayed. Adding a semicolon to the end of the operation suppresses the output; try it out!

```
>>25*3;
```

Also type:

```
>> 1/0
```

**Warning: Divide by zero.**

```
ans =
```

```
  Inf
```

```
>> Inf/Inf
```

```
ans =
```

```
  NaN
```

### **3. Relational operations**

Relational operators perform element-by-element comparisons between two numbers as following meaning;

$A < B$  Less than

$A > B$  Greater than

$A \leq B$  Less than or equal

$A \geq B$  Greater than or equal

$A == B$  Equal

$A \neq B$  Not equal

Further, there is a menu of special characters that have specific uses.

Logical AND &

Logical OR |

Logical NOT ~

Colon :

Subscripting ( )

Brackets [ ]

Decimal point .

Continuation ...

Separator ,

Semicolon ; (suppresses the output of a calculation)

Assignment =

Quote ' *statement* '

Transpose '

Comment %

**Note:** anything after % is a comment and will be ignored by Matlab.

## **4. Variables**

Variable names may be up to 19 characters long. Names must begin with a letter but may be followed by any combination of letters, digits or underscores. Variables are storage locations in the computer that are associated with an alphanumeric name. To assign a value to a variable in MATLAB simply type the name of the variable, followed by the assignment operator, =, followed by the value.

As an example, this is one way to compute 2+2:

```
>> a = 2
```

```
a =
```

```
2
```

```
>> b = 2
```

```
b =
```

```
2
```

```
>> c = a + b
```

```
c =
```

```
4
```



It is often annoying to have Matlab always print out the value of each variable. To avoid this, put a semicolon after the commands:

```
>> a = 2;  
>> b = 2;  
>> c = a + b;  
>> c  
c =  
    4
```

Only the final line produces output. Semicolons can also be used to string together more than one command on the same line:

```
>> a = 2; b = 2; c = a + b; c  
c =  
    4
```

Of course Matlab will also allow more complicated operations:

```
>> a = 2;  
>> b = -12;  
>> c = 16;  
>> qu1 = (-b + sqrt(b^2 - 4*a*c)) / (2*a)  
qu1 =  
    4
```

Understand that 'matlab' is "case sensitive", that is, it treats the name 'C' and 'c' as two different variables. Similarly, 'MID' and 'Mid' are treated as two different variables. Assign two different values to the variables and print them out by entering their names separated by a comma.

```
>>var=1.2  
var =  
    1.2000  
>>Var=-5.1  
Var =  
   -5.1000  
>>var, Var  
var =  
    1.2000
```

```
Var =  
-5.1000
```

#### **4.1 Predefined variables**

There are several predefined variables which can be used at any time, in the same manner as user defined variables (*ans*, *pi*, *j*, *eps*):

```
l: sqrt(-1)
```

```
j: sqrt(-1)
```

```
pi: 3.1416...
```

For example,

```
>>pi
```

```
ans =
```

```
3.1416
```

```
>>eps
```

```
eps =
```

```
2.2204e-016
```

```
>>j
```

```
ans =
```

```
0 + 1.0000i
```

```
>>y= 2*(1+4*j)
```

yields:

```
y=
```

```
2.0000 + 8.0000i
```

#### **5. Reporting format**

By default MATLAB returns numerical expressions as decimals with 4 digits. The `format` function is used to change the format of the output. Type `format rat` to have MATLAB return rational expressions.

```
>> format rat
```

```
>> 5.1-3.3
```

```
ans =
```

```
9/5
```

To eliminate the extra spacing type `format compact`.

```
>> format compact
```

```
>> 5*7
```

```
ans =
```

```
35
```

Now type

```
>> format long
```

```
>> 3*(23+14.7-4/6)/3.5
```

```
ans=
```

```
31.74285714285715
```

```
>> format short e
```

```
>> 3*(23+14.7-4/6)/3.5
```

```
ans=
```

```
3.1743e+01
```

Note that the answer is accurate to four decimal places. Now type

```
>> format long e
```

```
ans=
```

```
3.174285714285715e+01
```

```
>> format short
```

```
ans=
```

```
31.7429
```

Note: format short will return the numerical expression to default. Also, the format of reporting does not change the accuracy of the calculations only the appearance of the answer on screen.

## **6. Mathematical functions**

The following functions are defined in MATLAB

### **6.1. Trigonometric functions**

Those known to Matlab are sin, cos, tan and their arguments should be in radians.

**sin( ) - Sine.**

**sinh( ) - Hyperbolic sine.**

**asin( ) - Inverse sine.**

**asinh( ) - Inverse hyperbolic sine.**

**cos( ) - Cosine.**

**cosh( ) - Hyperbolic cosine.**

**acos( )** - Inverse cosine.  
**acosh( )** - Inverse hyperbolic cosine.  
**tan( )** - Tangent.  
**tanh( )** - Hyperbolic tangent.  
**atan( )** - Inverse tangent.  
**atanh( )** - Inverse hyperbolic tangent.  
**sec( )** - Secant.  
**sech( )** - Hyperbolic secant.  
**asec( )** - Inverse secant.  
**asech( )** - Inverse hyperbolic secant.  
**csc( )** - Cosecant.  
**csch( )** - Hyperbolic cosecant.  
**acsc( )** - Inverse cosecant.  
**acsch( )** - Inverse hyperbolic cosecant.  
**cot( )** - Cotangent.  
**coth( )** - Hyperbolic cotangent.  
**acot( )** - Inverse cotangent.  
**acoth( )** - Inverse hyperbolic cotangent.  
**>> x =5\*cos(pi/6), y = 5\*sin(pi/6)**  
**x =**  
**4.3301**  
**y =**  
**2.5000**

The inverse of trigonometric functions are called asin, acos, atan (as opposed to the usual arcsin or sin 1 etc.).

The result is in radians.

**>> acos(x/5), asin(y/5)**  
**ans =**  
**0.5236**  
**ans =**  
**0.5236**

**Note:** Matlab uses radian scale to calculate trigonometric functions. In other words, sin(90) is not equal to 1, but sin(pi/2) is.

## **6.2. Exponential**

These include sqrt, exp, log, log10

**exp( ) - Exponential.**

**log( ) - Natural logarithm.**

**log10( ) - Common (base 10) logarithm.**

**sqrt( ) - Square root.**

**abs( ) - Absolute value.**

**Note:** log( ) is ln in Matlab. To get logarithm in base 10, you must write log10( ).

```
>> exp(log(9)), log(exp(9))
```

```
ans =
```

```
9
```

```
ans =
```

```
9
```

Most common functions are available to Matlab

```
>>A=abs(-5), B=cos(3), C=exp(2), D=sqrt(4), E=log(40)
```

```
A =
```

```
5
```

```
B =
```

```
-0.9900
```

```
C =
```

```
7.3891
```

```
D =
```

```
2
```

```
E =
```

```
3.6889
```

## **6.3. Complex Number Functions**

**conj( ) - Complex conjugate.**

**Imag( ) - Complex imaginary part.**

**Real( ) - Complex real part.**

```
>>A=2+4*i, B=conj(A), C=Imag(A), D=real(A)
```

```
A =
```

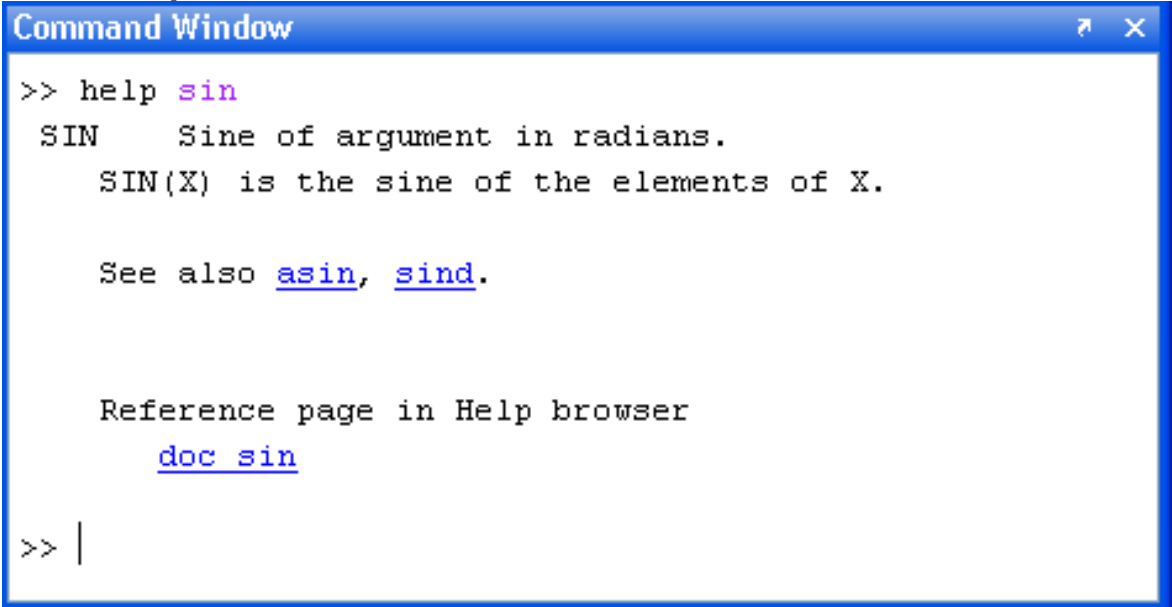
**2.0000 + 4.0000i**  
**B =**  
**2.0000 - 4.0000i**  
**C =**  
**4**  
**D =**  
**2**

## **7. Help**

MATLAB is a huge package. You can't learn everything about it at once, or always remember how you have done things before. It is essential that you learn how to teach yourself more using the online help.

If you need quick help on the syntax of a command, use help. For example, **help plot** tells you all the ways in which you can use the plot command. (Of course, you have to know already the name of the command you want.)

**>> help sin**



```
Command Window
>> help sin
SIN    Sine of argument in radians.
       SIN(X) is the sine of the elements of X.

       See also asin, sind.

       Reference page in Help browser
       doc sin

>> |
```

## **8. Closing Matlab**

To close MATLAB type exit in the command window and next press Enter or Return key. A second way to close your current MATLAB session is to select File in the MATLAB's toolbar and next click on Exit MATLAB option. All unsaved information residing in the MATLAB.

Workspace will be lost. You can also exit by typing:

```
>> quit
```

or

```
>> exit
```

To terminate a running Matlab command you may use **[Ctrl]+[c]** (Press both the Ctrl button and the c button simultaneously).

## **9. Common commands**

**whos** : gives a list of Matlab variables stored in the memory

**clear** : clears the memory

**clear A** : clears variable named A from the memory

**clc** : clears the command window

**clf** : clears the graphical window

The following example show how to assign values to variables, x and y.

```
>> x=sin(pi/4), y=log(2)
```

```
x =
```

```
0.7071
```

```
y =
```

```
0.6931
```

You can use who command to list the currently active variables. For the preceding session this results in

```
>> who
```

```
Your variables are:
```

```
ans    x      y
```

Use clear command to delete variables from computer memory

```
>> clear x
```

```
>> x
```

```
??? Undefined function or variable 'x'.
```

# ALGEBRA

## 1. Symbolic Toolbox

One of the most powerful tools that can be used in Matlab is the “Symbolic Toolbox”. To use Matlab’s facility for doing symbolic mathematics, it is necessary to declare the variables to be “symbolic”. The best way to do that is to use the **syms** declaration statement:

```
>>syms a b x y;  
>>c = 5;  
>>E = a*x^2 + b*x + c;
```

The **syms** statement makes all the variables listed with it into symbolic variables and gives each of the variables a value that is equal to its own name. Thus, the value of **a** is **a**, the value of **b** is **b**, etc. The variable **E** is now symbolic because it was assigned to be equal to an expression that contained one or more symbolic variables. Its value is **a\*x^2 + b\*x + 5**.

For example, suppose that you need factor  $x^2-3x+2$ . Note that you must type  $3*x$  for  $3x$ . Then you type:

```
>> syms x
```

By syms you are declaring that  $x$  is a variable. Then type

```
>> factor(x^2-3*x+2)
```

```
ans =
```

```
(x-1)*(x-2)
```

To factor  $x^2+2x+1$ , you write:

```
>> syms x
```

```
>> factor(x^2+2*x+1)
```

```
ans =
```

```
(x+1)^2
```

To factor the equation  $x^2-y^2$ ;

```
>>syms x y
```

```
>> factor(x^2-y^2)
```

```
ans =
```

```
(x-y)*(x+y)
```

Expand command can be used to expand the terms of any power equation.

Let's use expand command to expand the following equation  $(x^2-y^2)^3$ .

```
>> expand((x^2-y^2)^3)
```



**ans =**

**$x^6-3x^4y^2+3x^2y^4-y^6$**

The simplify command is useful to simplify some equations like.

**>> simplify((x^3-4\*x)/(x^2+2\*x))**

**ans =**

**x-2**

## **2. Solving Equations**

### **2.1 Algebraic Equations**

By using Symbolic Toolbox, you can find solutions of algebraic equations with or without using numerical values. If you need to solve equations, you can use the command solve. For example, to find the solution of  $x^3+x^2+x+1=0$  you write:

**>> solve('x^3+x^2+x+1=0')**

And Matlab give you the answer in the form

**ans =**

**[-1]**

**[ i]**

**[-i]**

That means the three solutions for the equation are 1, j, and -j.

**>>x=solve('sin(x)+x=0.1')**

**x =**

**5.001042187833512e-2**

In expressions with more than one variable, we can solve for one or more of the variables in terms of the others. Here we find the roots of the quadratic  $ax^2+bx+c$  in x in terms of a, b and c. By default solve sets the given expression equal to zero if an equation is not given.

**>> x=solve('a\*x^2+b\*x+c','x')**

**x =**

**[ 1/2/a\*(-b+(b^2-4\*a\*c)^(1/2))]**

**[ 1/2/a\*(-b-(b^2-4\*a\*c)^(1/2))]**

You can solve an equation in two variables for one of them. For example:

**>> y=solve('y^2+2\*x\*y+2\*x^2+2\*x+1=0', 'y')**

**y =**

```
[-x+i*(x+1)]  
[-x-i*(x+1)]
```

You can solve more than one equation simultaneously. For example to find the value of x and y from the equations:  $5x+10y=46$  and  $28x+32y=32$ , you write:

```
>> [x,y]=solve('5*x+10*y=46', '28*x+32*y=32')
```

And you get the following result:

```
x =  
-48/5
```

```
y =  
47/5
```

```
>> [x,y]=solve('log(x)+x*y=0', 'x*y+5*y=1')
```

```
x =  
.8631121967939437
```

```
y =  
.1705578823046945
```

To solve the system  $x^2+x+y^2=2$  and  $2x-y=2$ . We can type:

```
>> [x,y] = solve('x^2+ x+ y^2 = 2', '2*x-y = 2')
```

And get the solutions

```
x =  
[ 2/5]
```

```
[ 1]
```

```
y =  
[ -6/5]
```

```
[ 0]
```

This means that there are two points which are  $(2/5, -6/5)$  and  $(1, 0)$ .

Now let's find the points of intersection of the circles  $x^2+y^2=4$  and  $(x-1)^2+(y-1)^2=1$ .

```
>>[x,y]=solve('x^2+y^2=4','(x-1)^2+(y-1)^2=1')
```

```
x=  
[ 5/4-1/4*7^(1/2)]
```

```
[5/4+1/4*7^(1/2)]
```

```
y=  
[ 5/4+1/4*7^(1/2)]
```

**[5/4-1/4\*7^(1/2)]**

In same way if you have more then two equations you can use the same command to solve them for example:

```
[x,y,z]=solve('x+y+z=1','x+2*y-z=3','2*x-2*z=2')
```

```
x =
```

```
1/2
```

```
y =
```

```
1
```

```
z =
```

```
-1/2
```

## **2.2 DIFFERENTIAL EQUATIONS**

### **2.2.1 First Order Differential Equations**

Matlab can solve linear ordinary differential equations with or without initial/boundary conditions. Do not expect Matlab can solve nonlinear ordinary differential equations which typically have no analytical solutions. Higher derivatives can be handled as well. The command for finding the symbolic solution of differential equations is `dsolve`. For that command, the derivative of the function  $y$  is represented by  $Dy$ . For example, suppose that we want to find the solution of the equation  $x y' - y = 1$ . We will have:

```
>> dsolve('x*Dy-y=1', 'x')
```

```
ans =
```

```
-1+x*C1
```

This means that the solution is any function of the form  $y = -1 + cx$ , where  $c$  is any constant. The letter “D” has a special meaning and cannot be used otherwise inside `dsolve`. It means “first derivative of”. The  $C1$  is a constant of integration.

If we have the initial condition  $y(1) = 5$ , we can get the particular solution on the following way:

```
>>dsolve('Dy+y=cos(t)')
```

```
ans =
```

```
1/2*cos(t)+1/2*sin(t)+exp(-t)*C1
```

```
>> dsolve('x*Dy-y=1', 'y(1)=5', 'x')
```

```
ans =
```

$-1+6*x$

### **2.2.2 Second Order Equations**

The second order linear equations can be solved similarly as the first order differential equations by using dsolve. For the command dsolve, the second derivative of  $y$  is represented with  $D^2y$ . The letters “ $D^2$ ” mean second derivative.

For example, the command for solving  $y''-3y'+2y = \sin x$ .

```
>> dsolve('D2y-3*Dy+2*y=sin(x)', 'x')
```

```
ans =
```

```
3/10*cos(x)+1/10*sin(x)+C1*exp(x)+C2*exp(2*x)
```

If we have the initial conditions  $y(0) = 1$ ,  $y'(0)=-1$ , we would have:

```
>> dsolve('D2y-3*Dy+2*y=sin(x)', 'y(0)=1', 'Dy(0)=-1', 'x')
```

```
ans =
```

```
3/10*cos(x)+1/10*sin(x)+5/2*exp(x)-9/5*exp(2*x)
```

Example:  $d^2y/dx^2 - 2dy/dx - 3y = x^2$

```
>> dsolve('D2y - 2*Dy - 3*y=x^2', 'x')
```

```
ans =
```

```
-14/27+4/9*x-1/3*x^2+C1*exp(3*x)+C2*exp(-x)
```

Example:  $d^2y/dx^2 - 2dy/dx - 3y = x^2$ , with  $y(0)=0$ , and  $dy/dx = 1$  at  $x=1$

```
>> dsolve('D2y - 2*Dy - 3*y=x^2','y(0)=0, Dy(1)=1','x')
```

```
ans =
```

```
-1/3*x^2+4/9*x-14/27+1/9*(-11+14*exp(3))/(3*exp(3)+exp(-1))*exp(-x)  
+1/27*(33+14*exp(-1))/(3*exp(3)+exp(-1))*exp(3*x)
```

### **2.2.3 Higher Order Differential Equations**

Similarly you can use the same way to solve the higher order differential equations.

## **3. Representing Functions**

There is a way to define functions in MATLAB that behave in the usual

manner. To represent a function in Matlab, we use “inline” command. For example to declare  $f(x)=x^2+3x+1$  you write:

```
>> f=inline('x^2+3*x+1')
```

```
f=
```

**Inline function:**

**f(x) = x<sup>2</sup>+3\*x+1**

Therefore to find f(2), to get the answer you write:

```
>> f(2)
```

```
ans =
```

```
11
```

The function  $g(x,y)=x^2-3xy+2$  is defined as follows.

```
>> g=inline('x^2-3*x*y+2')
```

```
g =
```

**Inline function:**

**g(x,y) = x<sup>2</sup>-3\*x\*y+2**

Now we can evaluate  $g(2,3)$  in the usual way.

```
>>g(2,3)
```

```
ans =
```

```
-12
```

In some cases, if we need to define function f as a vector. Then we use:

```
>> f = inline(vectorize('x^2+3*x-2'))
```

```
f =
```

**Inline function:**

**f(x) = x.<sup>2</sup>+3.\*x-2**

In this case, we can evaluate a function at more than one point at the same time. For example, to evaluate the above function at 1, 3 and 5 we have:

```
>> f([1 3 5])
```

```
ans =
```

```
2 16 38
```

#### **4. Differentiation**

The Matlab function that performs differentiation is **diff**. These operations show how it works:

```
>> syms x
```

```
>>diff(x^2)
```

```
ans =
```

```
2*x
```

```
>>diff(sin(x)^2)
```

```
ans =
```

```
2*sin(x)*cos(x)
```

For example, let's find the derivative of  $f(x)=\sin(e^x)$ .

```
>> syms x
```

```
>> diff(sin(exp(x)))
```

and get the answer as:

```
ans =
```

```
cos(exp(x))*exp(x)
```

Note: Instead of using syms to declare of variables you can use two Quotes ' ' to declare that the variable x is the interested variable in equation; you can use the same example in otherwise

```
>>diff('sin(exp(x))')
```

```
ans =
```

```
cos(exp(x))*exp(x)
```

The  $n^{th}$  derivative of  $f$  is in the written in the form  $diff(f,n)$ . then to find the second derivative we write;

```
>> diff(sin(exp(x)),2)
```

```
ans =
```

```
-sin(exp(x))*exp(x)^2+cos(exp(x))*exp(x)
```

For example to find the first derivative of  $x^3+3x^2+8x$  you simply write:

```
>> syms x
```

```
>> diff(x^3+3*x^2+8*x)
```

```
ans =
```

```
3*x^2+6*x+8
```

Moreover to get the 3rd derivative, write:

```
>> diff(x^3+3*x^2+8*x ,3)
```

```
ans =
```

```
6
```

Note: To get higher derivatives, you can write the degree in place of 3.

To compute the partial derivative of an expression with respect to some

variable we specify that variable as an additional argument in diff. For example to find the derivative for x in equation  $f(x,y)=x^3y^4+ysin(x)$ .

```
>>syms x y
>> diff(x^3*y^4+y*sin(x),x)
ans =
3*x^2*y^4+y*cos(x)
Next we compute diff for y
>> diff(x^3*y^4+y*sin(x),y)
ans =
4*x^3*y^3+sin(x)
Finally we compute  $d^3 f_x^3$ .
>> diff(x^3*y^4+y*sin(x),x,3)
ans =
6*y^4-y*cos(x)
```

## **5. Integration**

By using the Symbolic Toolbox, you can find both definitive and in-definitive integrals of functions. We can use MATLAB for computing both definite and indefinite integrals using the command int. If  $f$  is a symbolic expression in  $x$ , then:

$$\text{int}(f) \rightarrow \int f(x)dx$$

For the indefinite integrals, consider the following example:

```
>> int('x^2')
ans =
1/3*x^3
```

Similarly as for diff command, we do not need the quotes if we declare  $x$  to be a symbolic variable. Therefore the above command can be re-written in otherwise such as:

```
>> syms x
>> int(x^2)
ans =
1/3*x^3
```

For example to find the in-definitive integral of  $x^3+sin(x)$ , you write:

```
>> syms x
>> int(x^3+sin(x))
```

```
ans =  
1/4*x^4-cos(x)
```

A definite integral can be taken by giving three arguments. The second and third arguments in that case are the first and second limits of integration. For the definitive integrals:

```
>> int(x^2, 0, 1)
```

```
ans =  
1/3
```

Try these examples,

```
int(x,1,2)
```

```
int(x*sin(x),-2,7)
```

Moreover to get definitive integral to  $\ln(x)+1/(x+1)$  from  $x=1$  to  $x=2$  write, you simply write:

```
>> int('ln(x) + 1/(x+1)', 1, 2)
```

```
ans =  
log(6)-1
```

## 6. Limits

You can use limit to compute limits. For example, to evaluate the limit when  $x$  goes to 2 of the function  $(x^2-4)/(x-2)$ , we have:

```
>> syms x
```

```
>> limit((x^2-4)/(x-2), x, 2)
```

```
ans =  
4
```

Limits at infinity:

```
>> limit(exp(-x^2-5)+3, x, Inf)
```

```
ans =  
3
```

## Practice Problems

1) Write required code to solve each of the following:-

a) Factor  $x^3+3x^2y+3xy^2+y^3$ .



- b) Simplify  $(x^3-8)/(x-2)$ .
- c) Expand  $(x^2+1)(x-5)(2x+3)$ .
- d) Solve  $\sin x = 2-x$  for  $x$ .
- e) Solve  $5x+2y+4z = 8$ ,  $-3x+y+2z = -7$ ,  $2x+y+z = 3$  for  $x$ ,  $y$  and  $z$ .
- f) Solve  $y^2-5xy-y+6x^2+x = 2$  for  $x$ .
- g) Find the first derivative of the function  $(\sin x / (\ln(x^2+1))) - e^x$  and evaluate it at  $x=3$ .
- h) Find the 12<sup>th</sup> derivative of the function  $(x/2+1)^{65}$
- i) Find the first and second partial derivatives for  $x$  of the function  $e^{x^2} \sin xy$

2) Obtain the first and second derivatives of the following functions using MATLAB's symbolic mathematics.

- a)  $F(x) = x^5 - 8x^4 + 5x^3 - 7x^2 + 11x - 9$
- b)  $F(x) = (x^3 + 3x - 8)(x^2 + 21)$
- c)  $F(x) = (3x^3 - 8x^2 + 5x + 9)/(x + 2)$
- d)  $F(x) = (x^5 - 3x^4 + 5x^3 + 8x^2 - 13)^2$
- e)  $F(x) = (x^2 + 8x - 11)/(x^7 - 7x^6 + 5x^3 + 9x - 17)$

# Vectors

An **array** is a collection of numbers, called **elements**, referenced by one or more indices running over different index sets. In MATLAB, the index sets are always sequential integers starting with 1. The **dimension** of the array is the number of indices needed to specify an element. The **size** of an array is a list of the sizes of the index sets.

A **matrix** is a two-dimensional array with special rules for addition, multiplication, and other operations. The two dimensions are called the **rows** and the **columns**.

A **vector** is a matrix in which one dimension has only the index 1. A **row vector** has only one row and a **column vector** has only one column.

## 1. Entering Vectors and Matrices

There are several ways to enter vectors and matrices in Matlab. These include:

- 1- Entering an explicit list of elements.
- 2- Loading matrices from external data files.
- 3- Generating matrices using functions.

To enter a vector or matrix explicitly, there are a few basic rules to follow:

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

For example, to input a 3×1 vector:

```
>> u=[1 2 3]
```

```
u =
```

```
1 2 3
```

The entries must be enclosed in square brackets.

```
>> v=[ 1 3 sqrt(5)]
```

```
v =
```

```
1.0000 3.0000 2.2361
```

Spaces is very important:

```
>> v2=[3+ 4 5]
```

```
v2 =
```

```
7 5
```

```
>> v3=[3 +4 5]
```

```
v3 =
```

```
3 4 5
```

We can do certain arithmetic operations with vectors of the same length, such as v and v3 in the previous section.

```
>> v + v3
```

```
ans =
```

```
4.0000 7.0000 7.2361
```

```
>> v4 = 3*v
```

```
v4 =
```

```
3.0000 9.0000 6.7082
```

```
>> v5 = 2*v - 3*v3
```

```
v5 =
```

```
-7.0000 -6.0000 -10.5279
```

We can build row vectors from existing ones:

```
>> w = [1 2 3], z = [8 9]
```

```
w =
```

```
1 2 3
```

```
z =
```

```
8 9
```

```
>> v6 = [w z]
```

```
v6 =
```

```
1 2 3 8 9
```

A vector can be defined by previously defined vectors.

```
>> x = [2 4 -1]
```

```
x =
```

```
2 4 -1
```

```
>> x1 = [x 5 8]
```

```
x1 =
```

```
2 4 -1 5 8
```

An special array is the **empty matrix**, which is entered as [].and can be used to delete a part of any matrix or vector.

```
>> w = [4 5 6 7]
```

```
w =
```

```
4 5 6 7
```

```
>>w(4)=[];w
```

```
w =
```

```
4 5 6
```

The elements of a matrix can be defined with algebraic expressions placed at the appropriate location of the element. Thus

```
>> a = [ sin(pi/2) sqrt(2) 3+4 6/3 exp(2) ]
```

```
a =
```

```
1.0000 1.4142 7.0000 2.0000 7.3891
```

## **2. Column Vectors**

The column vectors have similar constructs to row vectors. When defining them, entries are separated by ; or “newlines”.

Note how the column vector is defined, using brackets and semicolons to separate the different rows. To define a column vector x:

```
x=[1; -2; 4]
```

```
x =
```

```
1
```

```
-2
```

```
4
```

or write

```
>>x=[1
```

```
2
```

```
3]
```

```
x =
```

```
1
```

```
-2
```

```
4
```

```
>> c =[ 1; 3; sqrt(5)]
```

```
c =
```

```
1.0000
```

```
3.0000
```

```
2.2361
```

## **3. Transposing**

We can convert a row vector into a column vector (and vice versa) by a process called *transposing*, denoted by  $'$

```
> A=[ 1 2 3]
A =
    1    2    3
>>B= A'
B =
    1
    2
    3
```

#### **4. Vectors Addition and subtraction**

Addition and subtraction of a number to or from a vector can be made. In this case, the number is added to or subtracted from all the elements of the vector. For example

```
>> x=[-1; 0; 2];
>> y=x-1
y =
   -2
   -1
    1
```

If we look to make a simple addition and subtraction of vectors. The notation is the same as found in most linear algebra. We will define two vectors then add or subtract them:

```
>> v = [1; 2; 3]
v =
    1
    2
    3
>> b = [2; 4; 6]
b =
    2
    4
```

```

6
>> v+b
ans =
    3
    6
    9
>> v-b
ans =
   -1
   -2
   -3
>> sin(v)
ans =
    0.8415
    0.9093
    0.1411
>> log(v)
ans =
    0
    0.6931
    1.0986
>> pi*v
ans =
    3.1416
    6.2832
    9.4248

```

## **5. Vectors multiplication**

Multiplication of vectors and matrices must follow special rules. In the example above, the vectors are both column vectors with three entries. You cannot add a row vector to a column vector. In the case of multiplication vectors, the number of columns of the vector on the left must be equal to the number of rows of the vector on the right.

```
>> b = [2; 4; 6];
```

```

>> v = [1; 2; 3];
>> v*b
??? Error using ==> *
Inner matrix dimensions must agree.
>> v*b'
ans =
    2    4    6
    4    8   12
    6   12   18
>> v'*b
ans =
    28

```

## **6. element-wise operation**

There are many times where we want to do an operation to every entry in a vector or matrix. Matlab will allow you to do this with "element-wise" operations. For example, suppose you want to multiply each entry in vector  $v$  with its corresponding entry in vector  $b$ . In other words, suppose you want to find  $v(1)*b(1)$ ,  $v(2)*b(2)$ , and  $v(3)*b(3)$ . It would be nice to use the "\*" symbol since you are doing some sort of multiplication, but since it already has a definition, we have to come up with something else. The programmers who came up with Matlab decided to use the symbols ".\*" to do this.

```

>> v.*b
ans =
    2
    8
   18

```

Also for division we must use "./"

```

>> v./b
ans =
    0.5000
    0.5000
    0.5000

```

### **Note that**

$v .* b$  multiplies each element of  $v$  by the respective element of  $b$ .

$v ./ b$  divides each element of  $v$  by the respective element of  $b$ .

$v .\ b$  divides each element of  $b$  by the respective element of  $v$ .

$v .^ b$  raise each element of  $v$  by the respective  $b$  element.

## **7. The Colon Operator**

The colon operator ' : ' is understood by 'matlab' to perform special and useful operations. If two integer numbers are separated by a colon, 'matlab' will generate all of the integers between these two integers.

In particular, you will be able to use it to extract or manipulate elements of matrices. The following command creates a row vector whose components increase arithmetically:

```
>> 1:5
```

```
ans =
```

```
1 2 3 4 5
```

And, if three numbers, integer or non-integer, are separated by two colons, the middle number is interpreted to be a "step" and the first and third are interpreted to be "limits". Thus

```
>> b = 0.0 : 0.2 : 1.0
```

```
b =
```

```
0 0.2000 0.4000 0.6000 0.8000 1.0000
```

Suppose you want to create a vector with elements between 0 and 20 evenly spaced in increments of 2. Then you have to type:

```
>> t = 0:2:20
```

```
t =
```

```
0 2 4 6 8 10 12 14 16 18 20
```

```
>> m=0.32:0.1:0.6
```

```
m =
```

```
0.3200 0.4200 0.5200
```

```
>>w= -1.4:-0.3:-2
```

```
w =
```

```
-1.4000 -1.7000 -2.0000
```

The format is **first:step:last**. The result is always a row vector.



A negative step is also allowed. The command has similar results; it creates a vector with linearly spaced entries.

There is another way to create row arrays is to use `linspace` functions:

```
>> A=linspace(1,2,5)
```

```
A =
```

```
1.0000 1.2500 1.5000 1.7500 2.0000
```

```
>> A=linspace(0,20,11)
```

```
A =
```

```
0 2 4 6 8 10 12 14 16 18 20
```

## **8. Referencing elements**

It is frequently necessary to call one or more of the elements of a vector. Each dimension is given a single index. Some examples using the definitions above:

Evaluate a vector A:

```
>> A=0:10:100
```

```
A =
```

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

Now after definition of a vector A, try to type:

```
>> A(10)
```

```
ans =
```

```
90
```

```
>> B=A(1:5)
```

```
B =
```

```
0 10 20 30 40
```

```
>> C=A(1:2:10)
```

```
C =
```

```
0 20 40 60 80
```

```
>> A(6:2:10)
```

```
ans =
```

```
50 70 90
```

## Practice Problems

1. Create a vector of the even whole numbers between 31 and 75.
2. Create a vector  $x$  with the elements;
  - a) 2, 4, 6, 8
  - b) 8, 6, 4, 2, 0, -2, -4, -6
  - c) 1, 1/2, 1/3, 1/4, 1/5, 1/6
  - d) 0, 1/2, 2/3, 3/4, 4/5, 5/6
3. Let  $x = [2 \ 5 \ 1 \ 6]$ .
  - a) Add 16 to each element.
  - b) Add 3 to just the odd-index elements.
  - c) Compute the square root of each element.
  - d) Compute the square of each element.
4. Let  $x = [3 \ 2 \ 6 \ 8]$  and  $y = [4 \ 1 \ 3 \ 5]$ 
  - a) Add the elements in  $x$  to  $y$
  - b) Raise each element of  $x$  to the power specified by the corresponding element in  $y$ .
  - c) Divide each element of  $y$  by the corresponding element in  $x$
  - d) Multiply each element in  $x$  by the corresponding element in  $y$ , calling the result "z".
5. When  $A$  and  $B$  contain the values shown below, give the values of  $C$  vector after executing the following statements.  
 $A = [2 \ -1 \ 5 \ 0]$ ;  $B = [3 \ 2 \ -1 \ 4]$ 
  - a)  $C = A - B$
  - b)  $C = B + A - 3$
  - c)  $C = B ./ A$
  - d)  $C = A.^B$
  - e)  $C = 2.^B + A$
  - f)  $C = 2 * A + A.^B$

6. Use the **linspace** commands to create 4 vectors a, b, c, d with the elements:

a) 2, 4, 6, 8... to 10 elements

b) 10, 8, 6, 4, 2, 0, -2, -4...to 15 elements

c) 1, 1/2, 1/3, 1/4, 1/5... to 10 elements

d) 0, 1/2, 2/3, 3/4, 4/5... to 10 element

## Other Operations on Vectors

MATLAB has a large number of built-in functions. You will only become familiar with them by using them.

Try to make the vector **v** in command window and use the functions below.

**v=[23 0 3 16 -8 13]**

**length(v)** number of elements in v.

**6**

**size(v)** size of matrix v (row, column).

**1 6**

**find(v)** finds indices of non-zero elements.

**1 3 4 5 6**

**find(v==0)** finds indices of elements equal to zero.

**2**

**find(v==16)** finds indices of elements equal to 16.

**4**

**find(v>7)** finds indices of elements greater than 7.

**1 4 6**

**v(find(v>7))** finds the values of elements greater than 7.

**23 16 13**

**sum(v)** sum of elements

**47**

**max(v)** maximum element.

**23**

**min(v)** minimum element.

**-8**

**mean(v)** mean of elements.

**7.8333**

**sort(v)** sorts elements from minimum to maximum value.

**-8 0 3 13 16 23**

**all(v)** equal to 1 if all elements nonzero, 0 if any element nonzero.

**0**

**abs(v)** vector with absolute value of all element

**23 0 3 16 8 13**

## **Roots**

To calculate the roots of a polynomial, enter the coefficients in an array in descending order. Be sure to include zeroes where appropriate.

For example to find the roots of the polynomial  $y = x^4 + 6x^3 + 7x^2 - 6x - 8 = 0$  type the command:

```
p = [ 1 6 7 -6 -8 ];
```

```
r = roots(p)
```

yields

```
r =
```

```
-4.0000
```

```
-2.0000
```

```
-1.0000
```

```
1.0000
```

Note: The coefficients could be entered directly in the roots command. The same answer as above would be obtained using the following expression.

```
r = roots([ 1 6 7 -6 -8 ])
```

```
r =
```

```
-4.0000
```

```
1.0000
```

```
-2.0000
```

```
-1.0000
```

For example finding the roots of  $y = x^4 + 3x^3 - 15x^2 - 2x + 9 = 0$  would be as easy as entering the following command;

```
r=roots([1 3 -15 -2 9])
```

```
r =
```

```
-5.5745
```

```
2.5836
```

```
-0.7951
```

```
0.7860
```

The **roots** command can find imaginary roots.

```
p = [ 1 -6 18 -30 25 ];
```

```
r = roots(p)
```

```
r =
```

**1.0000 + 2.0000i**

**1.0000 - 2.0000i**

**2.0000 + 1.0000i**

**2.0000 - 1.0000i**

It can also find repeated roots. Note the imaginary portion of the repeated roots is displayed as zero.

**p = [ 1 7 12 -4 -16 ];**

**r = roots(p)**

**r =**

**-4.0000**

**-2.0000 + 0.0000i**

**-2.0000 - 0.0000i**

**1.0000**

### **13. PolyVal**

You can use polyval and the fitted polynomial p to predict the y value of the data you've fitted for some other x values. The syntax for determining the value of a polynomial  $y=x^4 + 6x^3 + 7x^2 - 6x - 8$  at any point is as follows.

**p = [ 1 6 7 -6 -8 ];**

**y= polyval(p, 3)**

**y=**

**280**

Where p is the vector containing the polynomial coefficients, (see above). Similarly, the coefficients can be entered directly in the polyval command.

**y = polyval([1 6 7 -6 -8], 3)**

**y =**

**280**

The polynomial value at multiple points (vector) can be found.

**z = [ 3 5 7];**

**y = polyval(p,z)**

**y =**

**280 1512 4752**

### **14. Polyfit**

To determining the coefficients of a polynomial that is the best fit of a given data you can use **polyfit** command. The command is **polyfit(x, y, n)**, where x, y are the data vectors and 'n' is the order of the polynomial for which the least-squares fit is desired.

**Exercise 4:**

Fit x, y vectors to 3 rd order polynomial

**x = [ 1.0 1.3 2.4 3.7 3.8 5.1 ];**

**y = [ -6.3 -8.7 -5.2 9.5 9.8 43.9 ];**

**coeff = polyfit(x,y,3)**

**coeff =**

**0.3124 1.5982 -7.3925 -1.4759**

After determining the polynomial coefficients, the **polyval** command can be used to predict the values of the dependent variable at each value of the independent variable.

**ypred = polyval(coeff,x)**

**ypred =**

**-6.9579 -7.6990 -5.6943 8.8733 10.6506 43.8273**

Its clear that there is a deviation between the actual y points and predicted y points because that the polynomial is best fit to this actual points.

# Matrices

## 1. Entering matrices

Entering matrices into Matlab is the same as entering a vector, except each row of elements is separated by a semicolon (;) or a return:

```
>>B = [1 2 3 4; 5 6 7 8; 9 10 11 12]
```

```
B =
```

```
 1  2  3  4  
 5  6  7  8  
 9 10 11 12
```

Alternatively, you can enter the same matrix as follows:

```
>>B = [ 1 2 3 4
```

```
5 6 7 8
```

```
9 10 11 12]
```

```
B =
```

```
 1  2  3  4  
 5  6  7  8  
 9 10 11 12
```

Note how the matrix is defined, using brackets and semicolons to separate the different rows.

## 2. Transpose

The special character prime ' denotes the transpose of a matrix e.g.

```
>> A=[1 2 3; 4 5 6; 7 8 9]
```

```
A =
```

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

```
>> B=A'
```

```
B =
```

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



### **3. Matrix operations**

#### **3.1 Addition and subtraction**

Addition and subtraction of matrices are denoted by + and -. This operations are defined whenever the matrices have the same dimensions.

For example: If A and B are matrices, then Matlab can compute A+B and A-B when these operations are defined.

```
>> A = [1 2 3;4 5 6;7 8 9]
```

```
A =
```

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

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

```
B =
```

```
1 1 1
2 2 2
3 3 3
```

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

```
C =
```

```
1 2
3 4
5 6
```

```
>> A+B
```

```
ans =
```

```
2 3 4
6 7 8
10 11 12
```

```
>> A+C
```

```
??? Error using ==>+
```

**Matrix dimensions must agree.**

Matrices can be joined together by treating them as elements of vectors:

```
>>D=[A B]
```

```
D =
```

```
1 2 3 1 1 1
4 5 6 2 2 2
```

```

    7  8  9  3  3  3
>>A - 2.3
ans =
   -1.3000  -0.3000   0.7000
    1.7000   2.7000   3.7000
    4.7000   5.7000   6.7000

```

### **3.2 Matrix multiplication**

Matrix operations simply act identically on each element of an array. We have already seen some vector operations, namely + and -, which are defined for vectors the same as for matrices. But the operators \*, / and ^ have different matrix interpretations.

```

>> A=[1,2,3;4,5,6;7,8 0]
A =
    1    2    3
    4    5    6
    7    8    0
>> B=[1,4,7;2,5,8;3,6,0]
B =
    1    4    7
    2    5    8
    3    6    0
>> A*B
ans =
    14    32    23
    32    77    68
    23    68   113

```

### **3.3 Matrix division**

To recognize how the two operator / and \ work ;

$X = A \setminus B$  is a solution to  $A * X = B$

$X = B / A$  is a solution to  $X * A = B$

```

>>A=[1,2,3;4,5,6;7,8 0];
>>B=[1,4,7;2,5,8;3,6,0];

```

```
>>X= A\B
ans =
  -0.3333  -3.3333  -5.3333
   0.6667   3.6667   4.6667
   0        -0.0000   1.0000
```

```
>> X = B/A
```

```
X =
   3.6667  -0.6667   0.0000
   3.3333  -0.3333   0.0000
   4.0000  -2.0000   1.0000
```

### **3.4 Element-wise operation**

You may also want to operate on a matrix element-by-element. To get element-wise behavior appropriate for an array, precede the operator with a dot. There are two important operators here .\* and ./

A.\*B is a matrix containing the elements of A multiplied by the corresponding elements of B. Obviously A and B must have the same size. The ./ operation is similar but does a division. There is a similar operator .^ which raises each element of a matrix to some power.

```
>> E = [1 2;3 4]
```

```
E =
   1   2
   3   4
```

```
>> F = [2 3;4 5]
```

```
F =
   2   3
   4   5
```

```
>> G = E .* F
```

```
G =
   2   6
  12  20
```

If you have a square matrix, like E, you can also multiply it by itself as many times as you like by raising it to a given power.

```
>>E^3
```

```
ans =
```

```
37 54
81 118
```

If wanted to cube each element in the matrix, just use the element-by-element cubing.

```
>> E.^3
```

```
ans =
```

```
1 8
27 64
```

```
>> A = [1 2 3;4 5 6;7 8 9];
```

```
1./A
```

```
ans =
```

```
1.0000 0.5000 0.3333
0.2500 0.2000 0.1667
0.1429 0.1250 0.1111
```

```
>> A./A
```

```
ans =
```

```
1 1 1
1 1 1
1 1 1
```

Most elementary functions, such as sin, exp, etc., act element-wise.

```
>> cos(A*pi)
```

```
ans =
```

```
-1 1 -1
1 -1 1
-1 1 -1
```

```
>> exp(A)
```

```
ans =
```

```
1.0e+003 *
0.0027 0.0074 0.0201
0.0546 0.1484 0.4034
1.0966 2.9810 8.1031
```

#### **4. The Colon Operator**

The colon operator can also be used to create a vector from a matrix.

Define:

```
>> A = [1 2 3;4 5 6;7 8 9];
```

```
>> B=A(:,1)
```

```
B =
```

```
1
```

```
4
```

```
7
```

Note that the expressions before the comma refer to the matrix rows and after the comma to the matrix columns.

```
>> B=A(:,2)
```

```
B =
```

```
2
```

```
5
```

```
8
```

```
>> B=A(1,:)
```

```
B =
```

```
1 2 3
```

The colon operator is also useful in extracting smaller matrices from larger matrices. If the 4 x 3 matrix C is defined by

```
>> C = [ -1 0 0;1 1 0;1 -1 0;0 0 2 ]
```

```
C =
```

```
-1 0 0
```

```
1 1 0
```

```
1 -1 0
```

```
0 0 2
```

```
>> D=C(:,2:3)
```

creates the following 4 x 2 matrix:

```
D =
```

```
0 0
```

```
1 0
```

```
-1 0
```

```
0 2
```

```
>> D= C(3:4,1:2)
```

Creates a 2 x 2 matrix in which the rows are defined by the 3rd and 4th row of C and the columns are defined by the 1st and 2nd columns of the matrix, C.

```
D =
```

```
1 -1
```

```
0 0
```

## 5. Referencing elements

The colon is often a useful way to construct these indices.

```
>> A = [1 2 3;4 5 6;7 8 9];
```

```
>> A(:,3)=0
```

Evaluated the third column to zero.

```
A =
```

```
1 2 0
```

```
4 5 0
```

```
7 8 0
```

```
>> A(:,3)=[]
```

Deleted the third column.

```
A =
```

```
1 2
```

```
4 5
```

```
7 8
```

```
>> A(3,:)=[]
```

Deleted the third row.

```
A =
```

```
1 2
```

```
4 5
```

```
>> A(:,3)=5
```

Expand the matrix into 2x 3 matrix, with a the values of the third column equal to 5.

```
A =
```

```
1 2 5
```

```
4 5 5
```

```
>> A(3,:)=7:9
```

Expand the matrix into  $3 \times 3$  matrix, with a values of the third column equal to 7, 8, 9:

```
A =  
 1  2  5  
 4  5  5  
 7  8  9
```

An array is resized automatically if you delete elements or make assignments outside the current size. (Any new undefined elements are made zero.)

```
>> A(:,5)=10
```

Expand the matrix into  $3 \times 5$  matrix, with a values of the fourth column equal to 0 and the last column equal to 10:

```
A =  
 1  2  5  0 10  
 4  5  5  0 10  
 7  8  9  0 10
```

## **6. Matrix Inverse**

The function `inv` is used to compute the inverse of a matrix. Let, for instance, the matrix A be defined as follows:

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

```
A =  
 1  2  3  
 4  5  6  
 7  8 10
```

Then,

```
>> B = inv(A)
```

```
B =  
-0.6667 -1.3333 1.0000  
-0.6667 3.6667 -2.0000  
1.0000 -2.0000 1.0000
```

The inverse of matrix A can be found by using either  $A^{-1}$  or `inv(A)`.

```
>> A=[2 1 1; 1 2 2; 2 1 2]
```

```
A =  
2 1 1  
1 2 2  
2 1 2  
>> Ainv=inv(A)
```

```
Ainv =  
2/3 -1/3 0  
2/3 2/3 -1  
-1 0 1
```

Let's verify the result of  $A \cdot \text{inv}(A)$ .

```
>> A*Ainv
```

```
ans =  
1 0 0  
0 1 0  
0 0 1
```

Also let's verify the result of  $\text{inv}(A) \cdot A$

```
>> Ainv*A
```

```
ans =  
1 0 0  
0 1 0  
0 0 1
```

Note: There are two matrix division symbols in Matlab, / and \ in which

$a/b = a \cdot \text{inv}(b)$

$a \backslash b = \text{inv}(a) \cdot b$ .

## **7. Predefined Matrix**

Sometimes, it is often useful to start with a predefined matrix providing only the dimension. A partial list of these functions is:

**zeros:** matrix filled with 0.

**ones:** matrix filled with 1.

**eye:** Identity matrix.

Finally, here are some examples on this special matrices

```
>>A=zeros(2,3)
```



```

A =
0 0 0
0 0 0
>>B=ones(2,4)
B =
1 1 1 1
1 1 1 1
>>C=eye(3)
C =
1 0 0
0 1 0
0 0 1

```

## **8. Other Operations on Matrix**

Define a matrix M and examine the effect of each command separately:

```
>>M=[23 0 3;16 8 5;13 2 4;1 10 7]
```

```

M =
    23     0     3
    16     8     5
    13     2     4
     1    10     7

```

```
>>length(M) number of rows in M
```

```
4
```

```
>>size(M) matrix size (rows, columns)
```

```
4 3
```

```
>>find(M>7) finds indices of elements greater than 7.
```

```

1
2
3
6
8

```

```
>>sum(M) sum of elements in each column
```

```
53 20 19
```

```
>>max(M) maximum element in each column.
```

**23 10 7**

>>**min(M)** minimum element in each column

**1 0 3**

>>**mean(M)** mean of elements in each column

**13.2500 5.0000 4.7500**

>>**sort(M)** sorts each column  
column

**1 0 3**

**13 2 4**

**16 8 5**

**23 10 7**

>>**all(M)** 1 if all elements nonzero, 0 if any element nonzero

**1 0 1**

>>**abs(M)** vector with absolute value of all elements

**23 0 3**

**16 8 5**

**13 2 4**

**1 10 7**

>>**rand** returns a random value from 0 to 1.

**0.9058**

>>**rand(2,3)** returns a random matrix with 2 rows and 3 columns

**0.1270 0.6324 0.2785**

**0.9134 0.0975 0.5469**

>>**rand(3)** returns a random matrix 3x3

**0.9575 0.9706 0.8003**

**0.9649 0.9572 0.1419**

**0.1576 0.4854 0.4218**

- Given the arrays  $x = [1 \ 3 \ 5]$ ,  $y = [2 \ 4 \ 6]$  and  $A = [3 \ 1 \ 5 ; 5 \ 9 \ 7]$ , Calculate;

a)  $x + y$

b)  $x' + y'$

- c)  $A - [x ; y]$
- d)  $[x ; y].*A$
- e)  $A - 3$

## Condition

### 1. If Statement

What are you doing if you want certain parts of your program to be executed **only** in limited circumstances? The way to do that is by putting the code within an "if" statement. The most basic structure for "if" statement is as following:

```
if (relation)  
    (matlab commands)  
end
```

More complicated structures are also possible including combinations like the following:

```
if (relation)  
    (matlab commands)  
elseif (relation)  
    (matlab commands)  
elseif (relation)  
    (matlab commands)  
.  
.  
else  
    (matlab commands)  
end
```

Standard comparisons can be made by using relations. The relational operators in MATLAB are;

- < less than
- > greater than
- <= less than or equal
- >= greater than or equal
- == equal
- ~= not equal.

For example, the following code will set the variable  $j$  to be  $-1$  if  $a$  is less than  $b$ :

```
a = 2;  
b = 3;  
if a < b  
    j = -1  
end
```

Additional statements can be added for more decision. The following code sets the variable  $j$  to be  $2$  when  $a$  is greater than  $b$ .

```
a = 4; b = 3;  
if a < b  
    j = -1  
elseif a > b  
    j = 2  
end
```

The **else** statement provides all that will be executed if no other condition is met. The following code sets the variable  $j$  to be  $3$  when  $a$  is not greater and less than  $b$ .

```
a = 4; b = 4;  
if a < b  
    j = -1  
elseif a > b  
    j = 2  
else  
    j = 3  
end
```

Matlab allows you to write together multiple condition expressions using the standard logic operators, "&" (*and*), "|" (*or*), and "~" (*not*).

For example to check if  $a$  is less than  $b$  and at the same time  $b$  is greater than or equal to  $c$  you would use the following commands:

```
if a < b & b >= c  
    Matlab commands  
end
```

For example

```
x=1; y=0;  
if x < 0 & y < 0  
z = -x * y  
elseif x == 0 | y == 0  
z = 0  
else  
z = x^2  
end
```

The output of the above code is

```
z=  
0
```

## **2. Loop**

Many programs require iteration, or repetitive execution of a block of statements. MATLAB has two loop statements: **for** and **while** statements. All of the loop structures in matlab are started with a keyword such as "for" or "while" and all of them end with the word "end".

### **2.1 For loop**

If you want to repeat a certain commands in a predetermined way, you can use the "for" loop. The "for" loop will loop around some statement, and you must tell Matlab where to start and where to end. Basically, you give a vector in the "for" statement, and Matlab will loop through for each value in the vector:

The for statements have the following structure:

```
For variable = expression  
statement  
end
```

For example, a simple loop will go around four times:

```
for j=1:4  
j  
end
```

The output of the above code is

```
j =  
1
```

```
j =  
    2  
j =  
    3  
j =  
    4
```

For another example, if we define a vector and later want to change its elements, we can step through and change each individual entry:

```
v = [1:3:10];  
for j=1:4  
    v(j) = j;  
end  
>>v  
v =  
    1    2    3    4
```

Matrices may also be constructed by loop. Here is an example using two "for" loops.

```
for i=1:10  
    for j=1:10  
        t(i,j) = i*j;  
    end  
end
```

Notice that there isn't any output, since the only line that would produce any ( $t(i,j) = i*j$ ;) ends in a semi-colon. Without the semi-colon, Matlab would print the matrix  $t$  100 times!

### **Example 1**

Find summations of even numbers between 0 and 100

```
sum = 0;  
for i = 0 : 2 : 100  
    sum = sum + i;  
end  
sum  
sum =
```

## 2550

Note: if the increment is one you may write the first and the last limit without writing a step. In this case, the for-line above would become **for i = 0:100**.

warning: If you are using complex numbers, you cant use i as the loop index.

### 2.2 While Loop

If you don't like the "for" loop, you can also use a "while" loop. It is sometimes necessary to repeat statements based on a condition rather than a fixed number of times. The "while" loop repeats a sequence of commands as long as some condition is met. The general form of a while loop is

```
while relation  
    statements  
end
```

#### Example 2

```
x=1;  
while x < 10  
x = x+1;  
end  
>> x  
x =  
    10
```

The statements will be repeatedly executed as long as the relation remains true.

#### Example 3

```
x=10;  
while x > 1  
x = x/2;  
end  
>>x  
x =  
    0.6250
```

The condition is evaluated before the body is executed, so it is possible to

get zero iteration when x equal to 1 or less.

#### **Example 4**

```
sum = 0;
x = 1;
while x < 4
sum = sum + 1/x;
x = x + 1;
end
>>sum
sum =
    1.8333
```

#### **2.3 Break statement**

It's often a good idea to limit the number of repetitions, to avoid infinite loops (as could happen in above example if  $x = \text{Inf}$ ). This can be done using **break**. A break can be used whenever you need to stop the loop (break statement can be used to stop both for and while loops in same way), for example.

```
n = 0;
x=100
while x > 1
x = x/2;
n = n+1;
if n > 50
break
end
end
>> x
x =
    0.7813
```

A break immediately jumps execution to the first statement after the loop.

**Note:** When using programs with loops, you should also know how to stop a program in the middle of its execution, such as in the case of your programs contains an infinite loop. During a program is running, the **Ctrl+C** command will



stop it.

### **Exercise 1:**

Use loop in 20 iteration to calculate x in equation  $x=2\sin(x)$ ?

Solution:

**format long**

**x=1**

**for i=1:20**

**x=2\*sin(x)**

**end**

The results will be:

**x = 1**

**x = 1.682941969615793**

**x = 1.987436530272152**

**x = 1.828907552623580**

**x = 1.933747642340163**

**x = 1.869706153630775**

**x = 1.911316179125262**

**x = 1.885162348212226**

**x = 1.901985209663949**

**x = 1.891312851651419**

**x = 1.898145620030117**

**x = 1.893795924017278**

**x = 1.896575152213355**

**x = 1.894803506466833**

**x = 1.895934551140671**

**x = 1.895213162228076**

**x = 1.895673549670181**

**x = 1.895379846173585**

**x = 1.895567260289186**

**x = 1.895447688999369**

**x = 1.895523983862163**

You can see the convergence through the digits after the dot. The best guess for x is

$x = 1.895523983862163$

## **Practice Problems**

- Provide the right answers and use MATLAB to check them.

**if  $0 < x < 10$**

**$y = 2*x$**

**elseif  $10 < x < 50$**

**$y = 3*x$**

**else**

**$y = 200$**

**end**

- a)  $x = -1$   $y = ?$
- b)  $x = 5$   $y = ?$
- c)  $x = 10$   $y = ?$
- d)  $x = 45$   $y = ?$
- e)  $x = 100$   $y = ?$

## 2D Graphics

One of Matlab's most powerful features is the ability to create graphic plots. There are so many methods of generating graphics in Matlab.

### 1. X-Y Plots

A Cartesian or x,y plot is based on plotting the x,y data pairs from the specified vectors. Clearly, the vectors x and y must have the same number of elements. Given a vector x of x-coordinates  $x_1$  through  $x_n$  and a vector y of y-coordinates  $y_1$  through  $y_n$ , `plot(x,y)` graphs the points  $(x_1,y_1)$  through  $(x_n,y_n)$ .

For example, to plot the two dimensions corresponding to (0,0), (1,1), (4,2), (5,1) and (0,0) points .

```
x=[0 1 4 5 0]; y=[0 1 2 -1 0];
```

```
plot(x,y)
```

The command `plot(x,y)` opens a graphics window and draws an x-y plot of the elements of x versus the elements of y.

To plot the graph of  $y=x^3$  on the interval [2,2], first define a row vector whose components range from -2 to 2 in increments of .05. Then define a vector y; of the same size as x whose components are the cubes of the components of x. Finally use the plot function to display the graph.

```
x=-2:.05:2; y=x.^3;
```

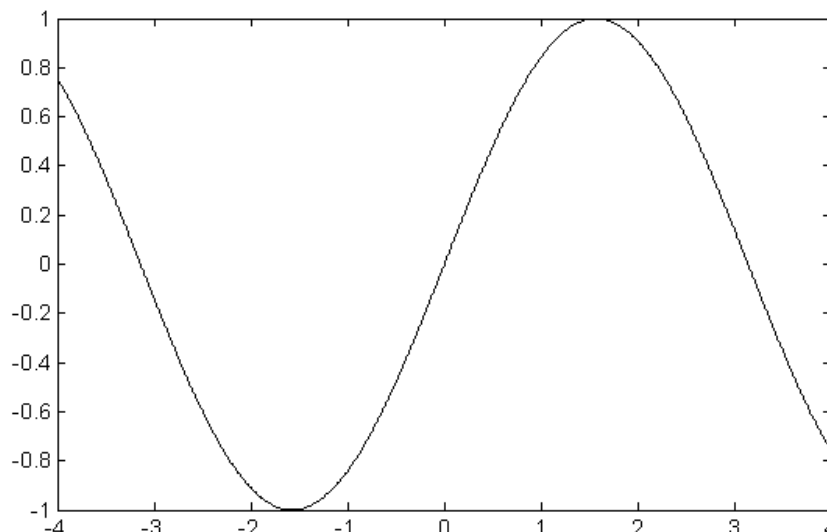
```
plot(x,y)
```

You can, also for example, draw the graph of the sine function over the interval -4 to 4 with the following commands:

```
x = -4:.01:4; y = sin(x);
```

```
plot(x,y)
```

The resulting plot is shown in figure 1.



### Figure 1. Plotting sine over the interval -4 to 4

Plots of parametrically defined curves can also be made. Try, for example,

```
t=0:.001:2*pi;
```

```
x=cos(3*t);
```

```
y=sin(2*t);
```

```
plot(x,y)
```

The resulting plot is shown in figure 2.

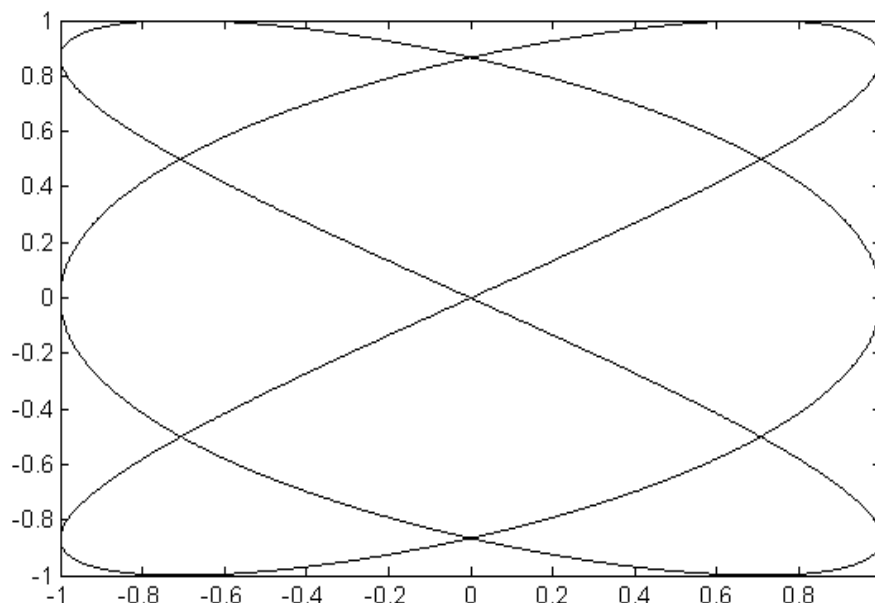


Figure 2. Plotting parametrically curves

### 2. Grid

A dotted grid may be added to the plot by:

```
grid on
```

This grid can be removed using grid off.

```
grid off
```

### 3. Controlling Axes

Once a plot has been created in the graphics window you may wish to change the range of x and y values, this can be done by using the command “axis” as follows:

```

x =0:.01:1;
y =sin(3*pi*x);
plot(x,y)
axis([-0.5 1.5 -1.2 1.2])

```

The axis command has four parameters, the first two are the minimum and maximum values of x to use on the axis and the last two are the minimum and maximum values of y.

#### **4. Plotting Line Style**

Matlab connects a line between the data pairs described by the vectors. You may wish to present data points and different type of connecting lines between these points. Data points can be described by a variety of characters such as ( . , + , \* , o and x .).

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:

<b>Character color</b>	<b>Character symbol</b>	<b>character line style</b>
b blue	. point	- solid
g green	o circle	: dotted
r red	x x-mark	-. dashdot
c cyan	+ plus	-- dashed
m magenta	* star	
y yellow	s square	
k black	d diamond	
	v triangle (down)	
	^ triangle (up)	
	< triangle (left)	
	> triangle (right)	
	p pentagram	
	h hexagram	

For example,

**plot(x,y,'k+:')** plots a black dotted line with a plus at each data point.

`plot(x,y,'bd')` plots blue diamond at each data point but does not draw any line.

`plot(x,y,'y-',x,y,'go')` plots the data twice, with a solid yellow line interpolating green circles at the data points.

### **5. Annotation**

The last step before printing or saving a plot is usually to put a title and label the axes. You can put labels, titles, and text on a plot by using the commands:

`xlabel('text')`

`ylabel('text')`

`zlabel('text')`

`title('text')`

`text(x,y,'text')` places text at position x,y

`gtext('text')` use mouse to place text

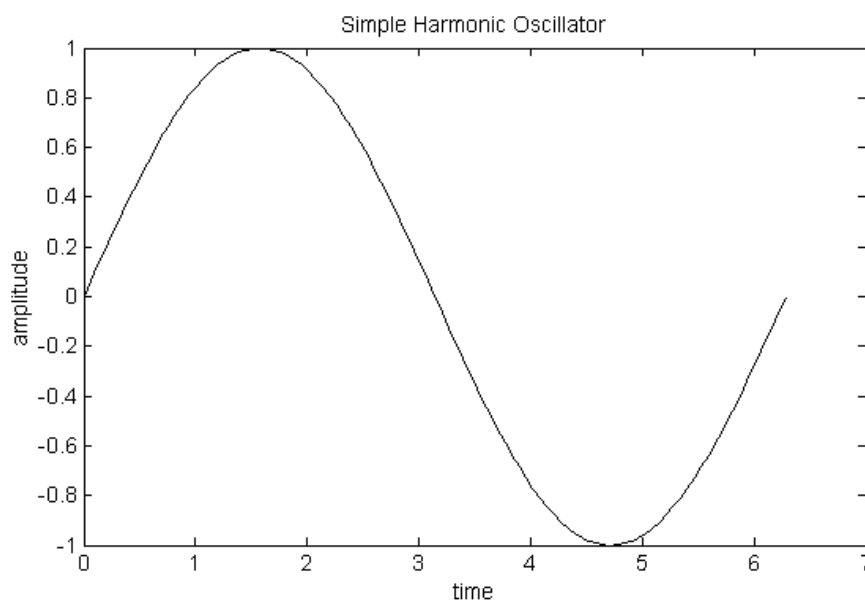
For example the graphs can be given titles, axes labeled, and text placed within the graph with the following commands;

`t = 0:0.02*pi:2*pi;`

`plot(t,sin(t))`

`xlabel('time');ylabel('amplitude');title('Simple Harmonic Oscillator')`

The resulting plot is shown in figure 3.



**Figure 3. Using annotation in plot**

## **6. Multi-plots**

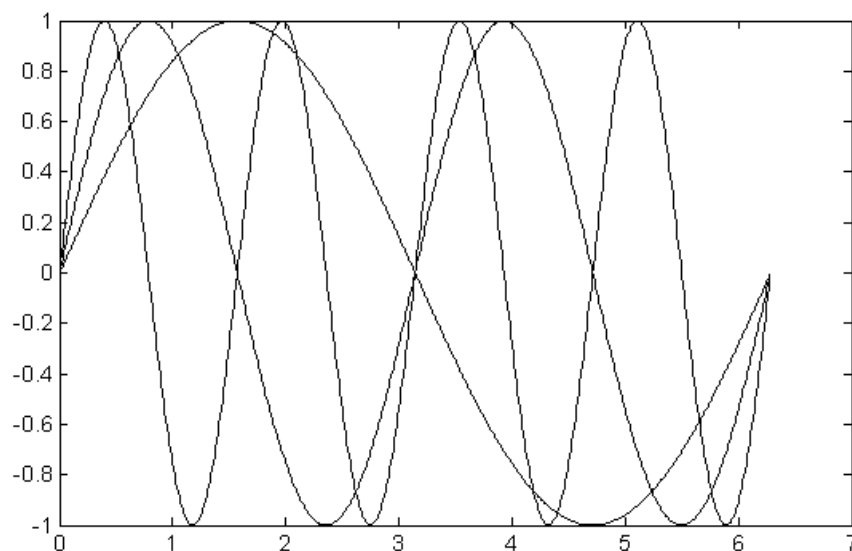
There are two ways to make multiple plots on a single graph are illustrated by.

### **6.1 Multiple Data Vectors Plots**

You can create multiple graphs by using multiple data vectors, for example:

```
x=0:.01:2*pi;  
y1=sin(x);y2=sin(2*x);y3=sin(4*x);  
plot(x,y1,x,y2,x,y3)
```

The resulting plot is shown in figure 4.



**Figure 4. Multiple plots of sine vectors**

### **6.2 Multiple Plots with Hold**

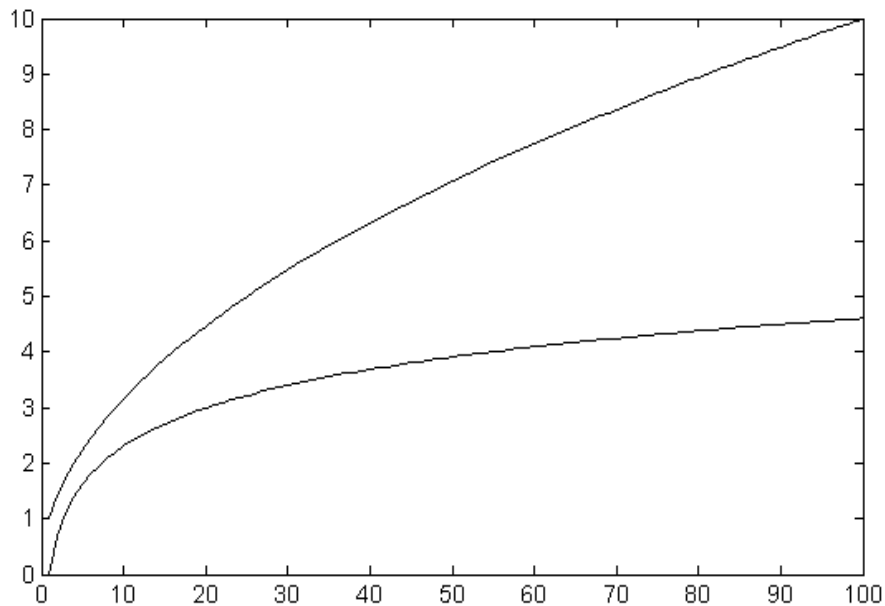
Another way is with hold on. The command "hold on" holds the old graph when the new one is plotted. The axes may, however, become rescaled. "hold on" holds the current picture; "hold off" releases it (but does not clear the window).

Here is an example. Suppose we want to compare the log function with the square root function graphically. We can put them on the same plot. By default, both plots will appear in blue, so we will not know which log. We could make them different colors using options. Here is the final answer,

```
x=1:100;  
y=log(x);  
z=sqrt(x);  
plot(x,y,'r'); % plot log in red
```

```
hold on;  
plot(x,z,'b'); % plot log in blue  
hold off;
```

The resulting plot is shown in figure 5.



**Figure 5. Multiple plots with hold**

The "hold" command will remain active until you turn it off with the command 'hold off'.

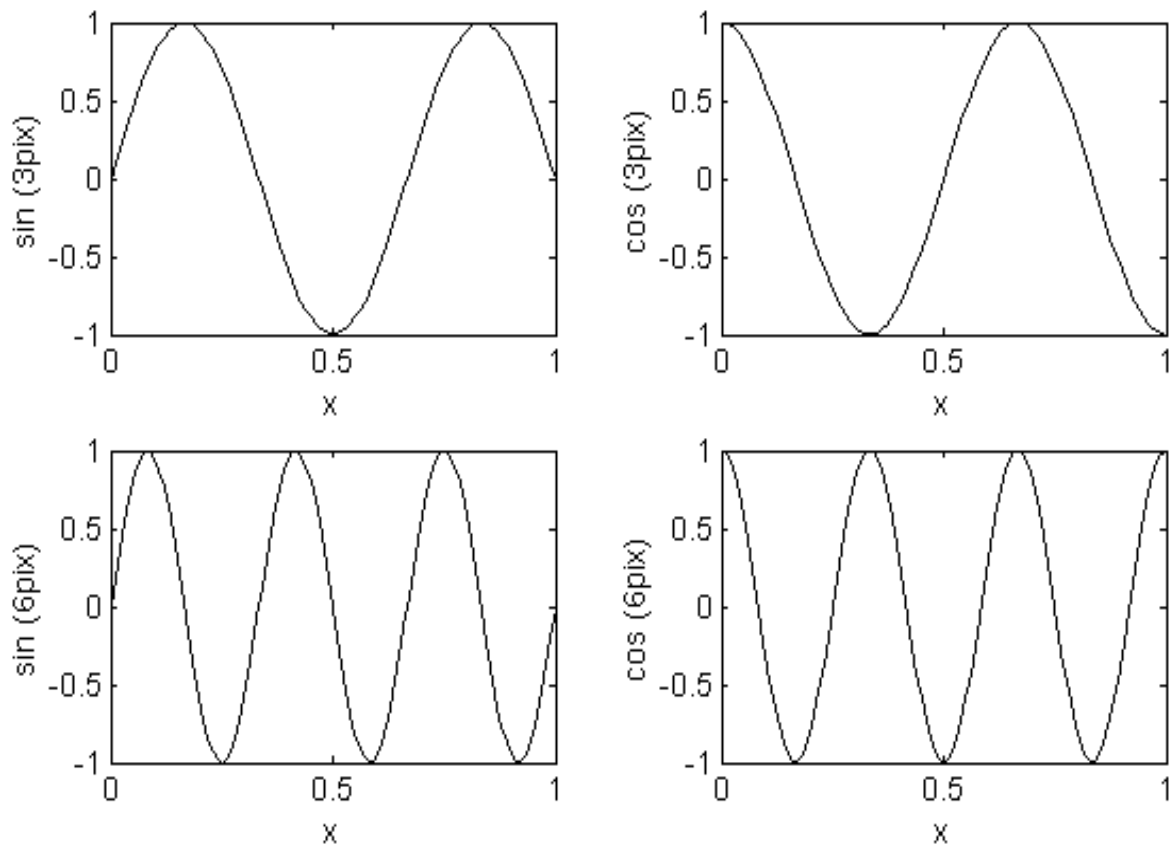
## **7. Subplot**

The graphics window may be split into an  $m$  by  $n$  array of smaller windows into which we may plot one or more graphs. The windows are numbered 1 to  $m$  by  $n$  row-wise, starting from the top left. All of plot properties such as hold and grid work on the current subplot individually.

```
x=0:.01:1;  
subplot(221), plot(x,sin(3*pi*x))  
xlabel('x'),ylabel('sin (3pix)')  
subplot(222), plot(x,cos(3*pi*x))  
xlabel('x'),ylabel('cos (3pix)')  
subplot(223), plot(x,sin(6*pi*x))  
xlabel('x'),ylabel('sin (6pix)')  
subplot(224), plot(x,cos(6*pi*x))  
xlabel('x'),ylabel('cos (6pix)')
```



The resulting plot is shown in figure 6.



**Figure 7. Using subplot**

Subplot(221) (or subplot(2,2,1)) specifies that the window should be split into a 2 by 2 array and we select the first sub-window.

### **Practice Problems**

- 1) Plot the log of the values from 1 to 100.
- 2) Plot the parametric curve  $x = t \cos(t)$ ,  $y = t \sin(t)$  for  $0 < t < 10$ .
- 3) Plot the cubic curve  $y = x^3$ . Label the axis and put a title "a cubic curve" on the top of the graph.
- 4) Plot  $y = (x^3 + x + 1)/x$ , for  $-4 < x < 4$  and  $-10 < y < 10$ .
- 5) Plot  $y = \ln(x+1)$  and  $y = 1 - x^2$  on the same window for  $-2 < x < 6$  and  $-4 < y < 4$ .
- 6) Plot  $\cos(x^2 + 1)$  on  $[-2\pi, 2\pi]$ .

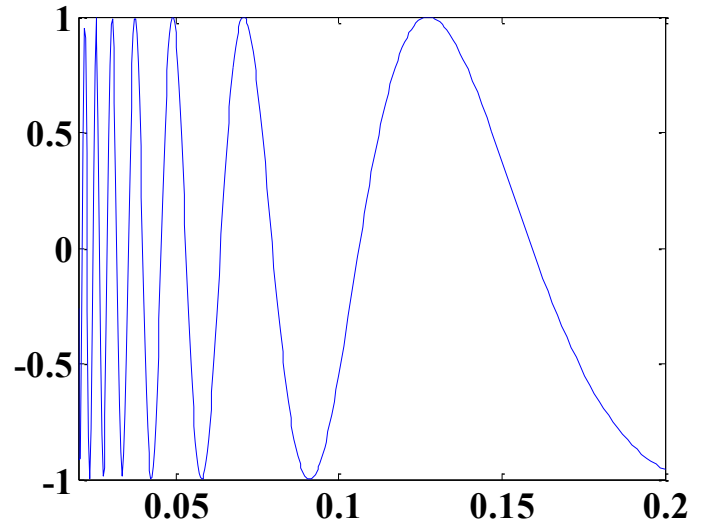
## 8. Specialized 2-D plotting functions

MATLAB includes a variety of specialized plotting in addition to the ones described above. The following below briefly describes some of the other plotting functions available in MATLAB.

**fplot:** evaluates a function and plots the results

**Example:**

**fplot('sin(1/x)', [0.02 0.2]);**

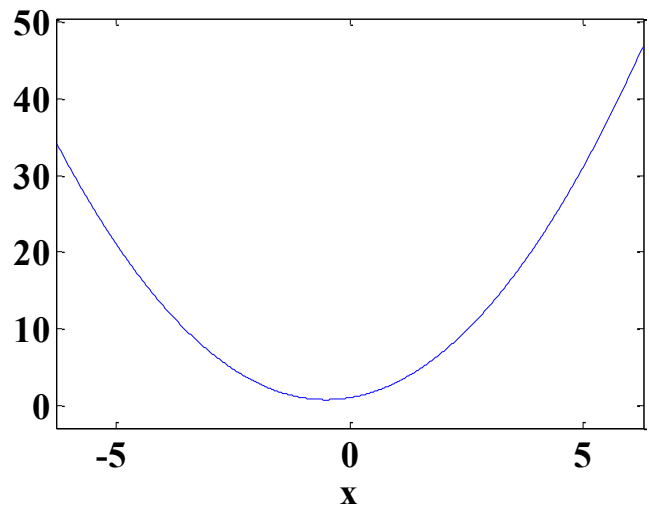


$$x^2+x+1$$

**ezplot:** simplest way to graph a function (easy plot).

**Example:** to graph the function  $y=x^2+x+1$ , you write:

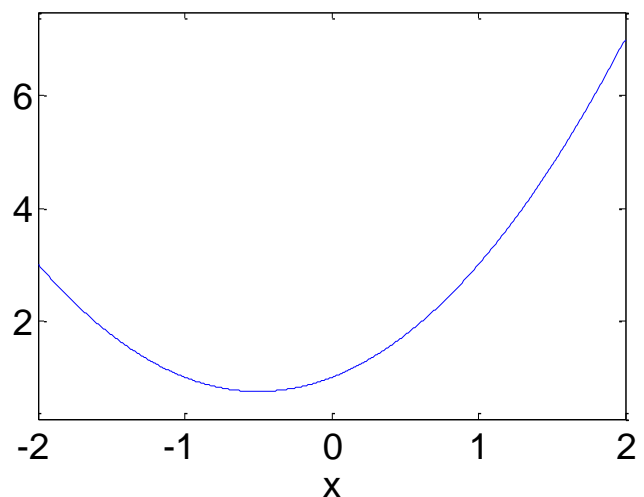
**ezplot('x^2+x+1')**



$$x^2+x+1$$

If you want to sketch the same function in between -2 and 2 you simply write:

**ezplot('x^2+x+1', [-2, 2])**

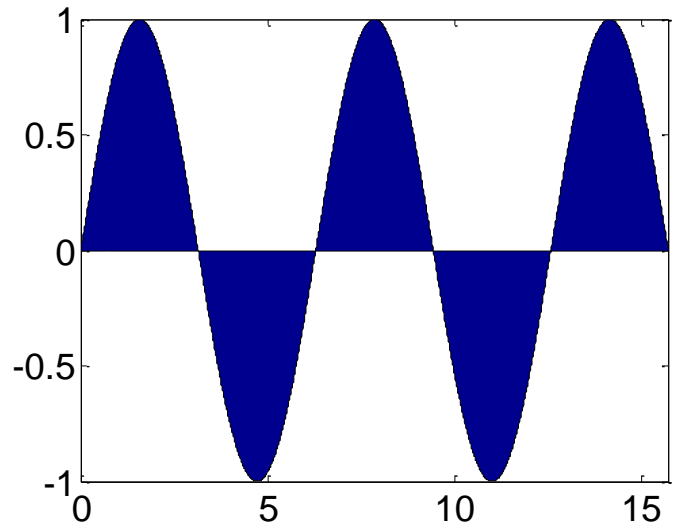


**area** Filled area plot.

**area(X,Y)** produces a stacked area plot suitable for showing the contributions of various components to a whole. For vector X and Y, area(X,Y) is the same as plot(X,Y) except that the area between 0 and Y is filled.

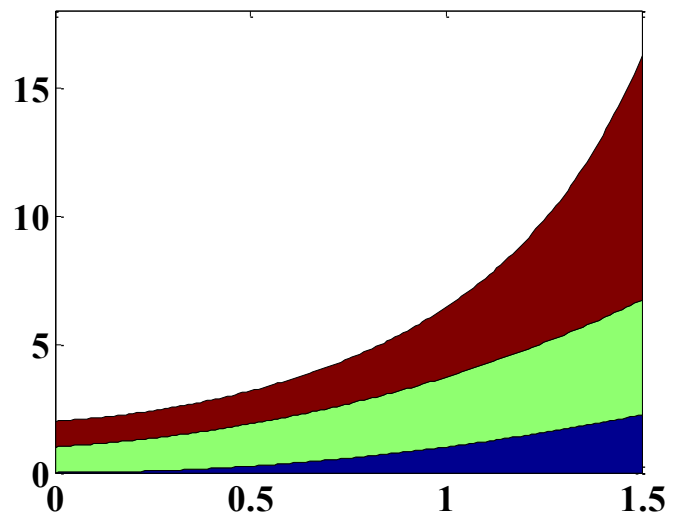
**Example:**

```
t = 0:.01:5*pi;  
area(t,sin(t))
```



**Example:**

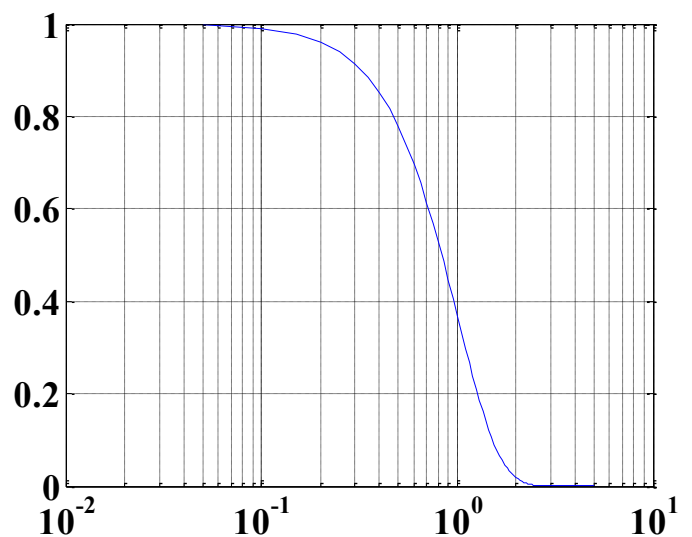
```
x = 0:0.01:1.5;  
area(x,[x.^2]',(exp(x))',(exp(x.^2))'])
```



**semilogx:** log-scaled x axis  
**semilogx** is the same as **plot**, except a logarithmic (base 10) scale is used for the X-axis.

**Example:**

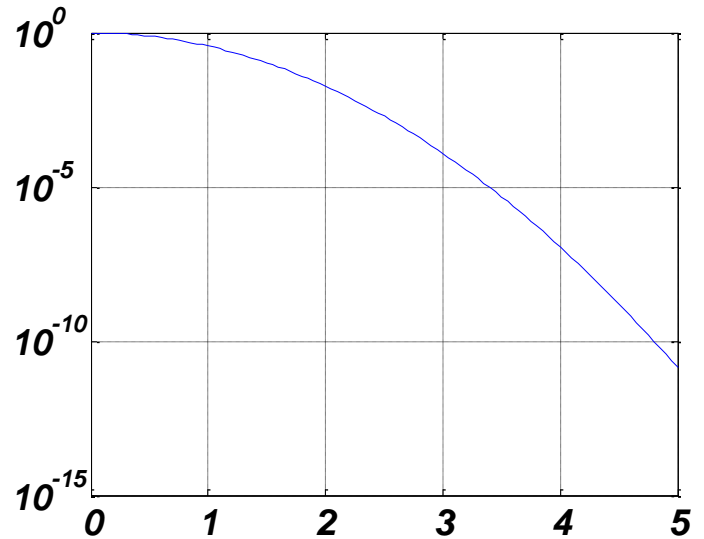
```
x=0:0.05:5;  
y=exp(-x.^2);  
semilogx(x,y);  
grid
```



**semilogy:** log-scaled y axis  
**semilogy** is the same as **plot**, except a logarithmic (base 10) scale is used for the Y-axis.

**Example:**

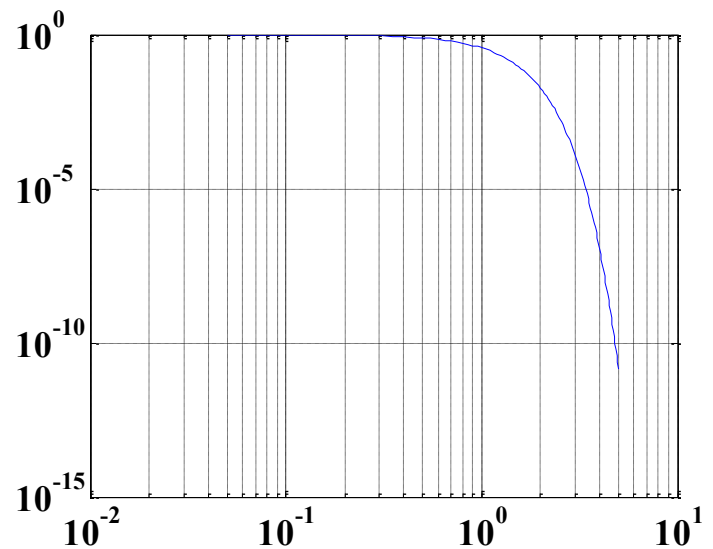
```
x=0:0.05:5;
y=exp(-x.^2);
semilogy(x,y);
grid
```



**Loglog:** Log-log scale plot.  
**Loglog** is the same as **plot**, except logarithmic scales are used for both the X- and Y-axes.

**Example:**

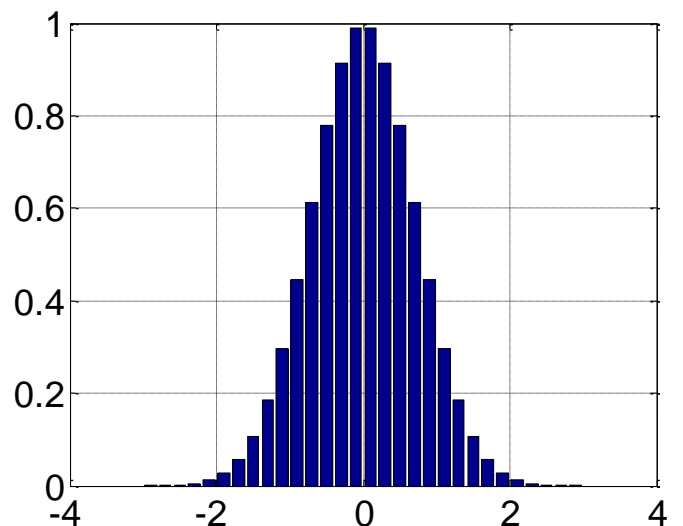
```
x=0:0.05:5;
y=exp(-x.^2);
loglog(x,y);
grid
```



**bar:** creates a bar graph.  
**bar(X,Y)** draws the columns of the M-by-N matrix Y as M groups of N vertical bars. The vector X must be monotonically increasing or decreasing

**Example:**

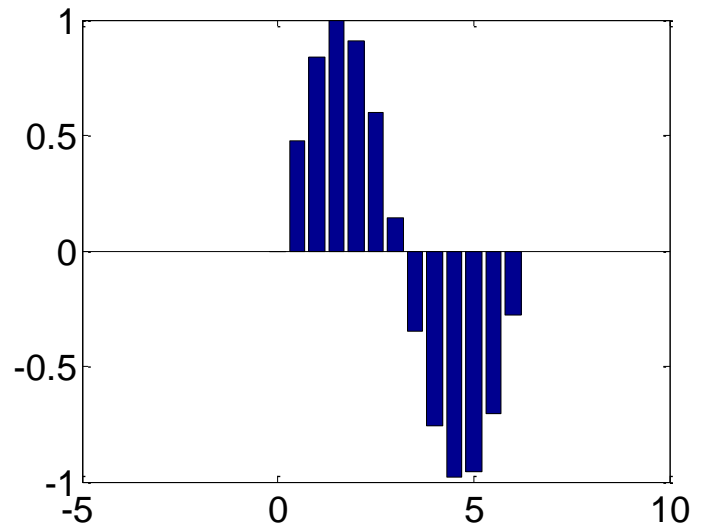
```
x = -2.9:0.2:2.9;
```



```
bar(x,exp(-x.*x));
grid on
```

**Example:**

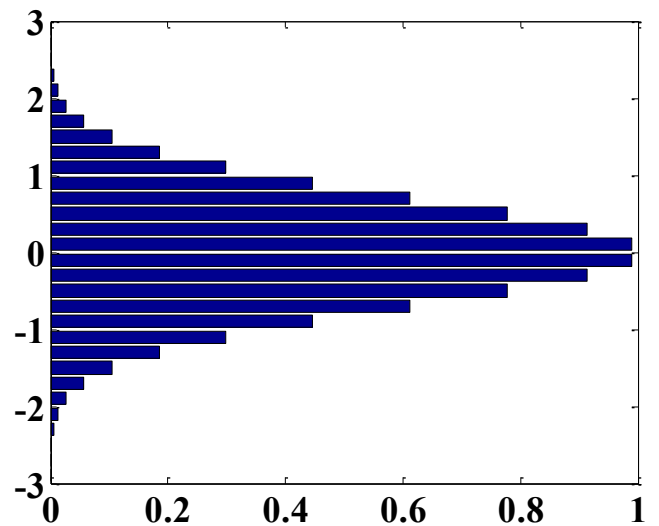
```
x = 0:0.5:2*pi;
bar(x, sin(x));
```



**barh:** horizontal bar graph  
**barh(X,Y)** draws the columns of the M-by-N matrix Y as M groups of N horizontal bars.

**Example:**

```
x = -2.9:.2:2.9;
y = exp(-x.*x);
barh(x,y);
```

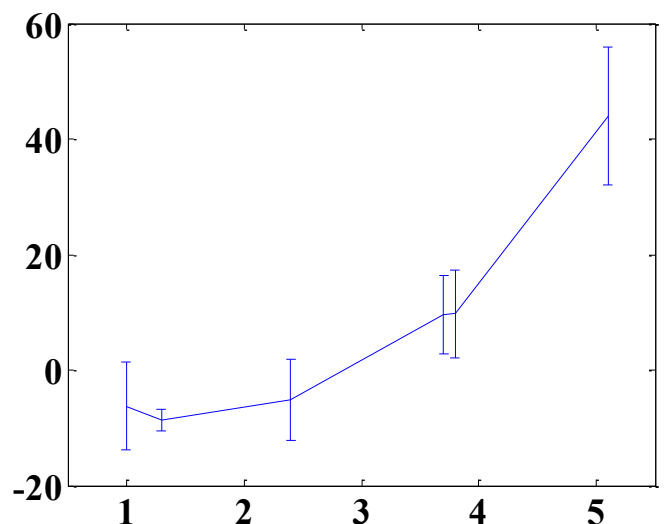


**errorbar:** creates a plot with error bars  
**errorbar (X,Y,L)** plots the graph of vector X vs. vector Y with error bars specified by the vectors L and U.

**Example:**

```
x = [ 1.0 1.3 2.4 3.7 3.8 5.1 ];
y = [ -6.3 -8.7 -5.2 9.5 9.8 43.9 ];
coeff = polyfit(x,y,1)
yp=polyval(coeff,x)
e=abs(yp-y)
errorbar(x,y,e); grid
e =
```

```
7.6079 1.8509 6.9582 6.8052 7.6242 11.9288
```

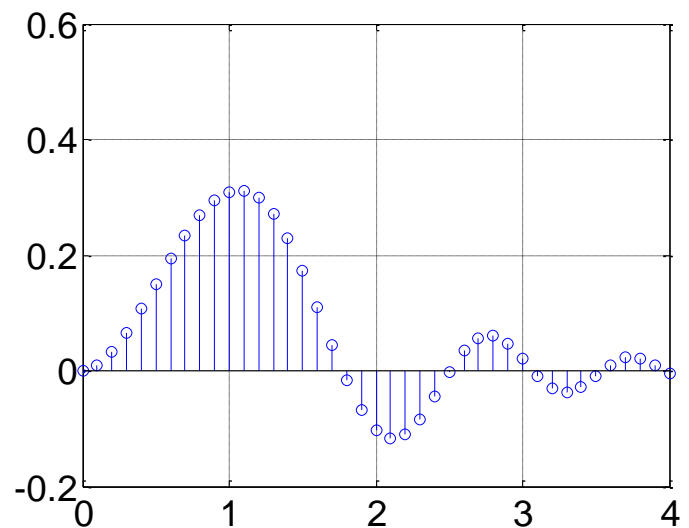
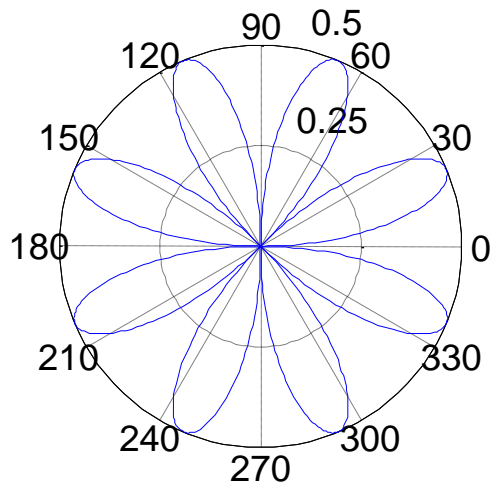


**polar:** creates a plot in polar coordinates of angles versus radius

**polar(theta,rho)** makes a plot using polar coordinates of the angle **theta**, in radians, versus the radius **rho**.

**Example:**

```
t=0:.01:2*pi;
polar(t,sin(2*t).*cos(2*t));
```



**stem:** generates stems at each data point

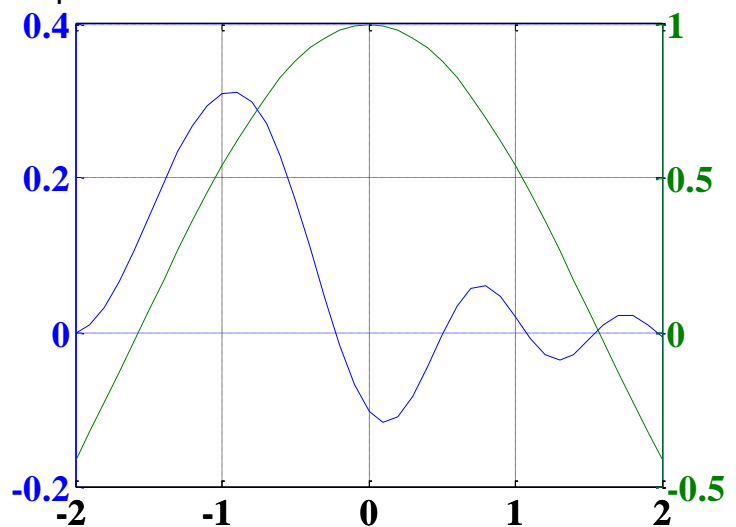
**Example:**

```
x = 0:0.1:4;
y = sin(x.^2).*exp(-x);
stem(x,y);
grid
```

**plotyy:** graphs with y tick labels on the left and right

**plotyy(X1,Y1,X2,Y2)**

plots Y1 versus X1 with y-axis labeling on the left and plots Y2 versus X2 with y-axis labeling on the right.



**Example:**

```
x=-2:0.1:2;  
y1=sin(x);  
y2=cos(x);  
plotyy(x,y,x,y2);  
grid
```

**stairs:** creates a graph similar to a bar graph, but without internal lines

**Example:**

```
x = -2.9:2:2.9;  
y = exp(-x.*x);  
stairs(x,y);  
title('Stair Chart');
```

**pie:** Pie chart.

**pie(X)** draws a pie plot of the data in the vector **X**. The values in X are normalized via  $X/\text{sum}(X)$  to determine the area of each slice of pie.

**Example:**

```
x = [1 6 2 1 3 9];  
label = {'UK','USA','CH','Dk','SU','NL'};  
pie(x, label)
```

**hist:** creates a histogram

**Example:**

```
x=rand(1,100);  
hist(x);  
grid
```

