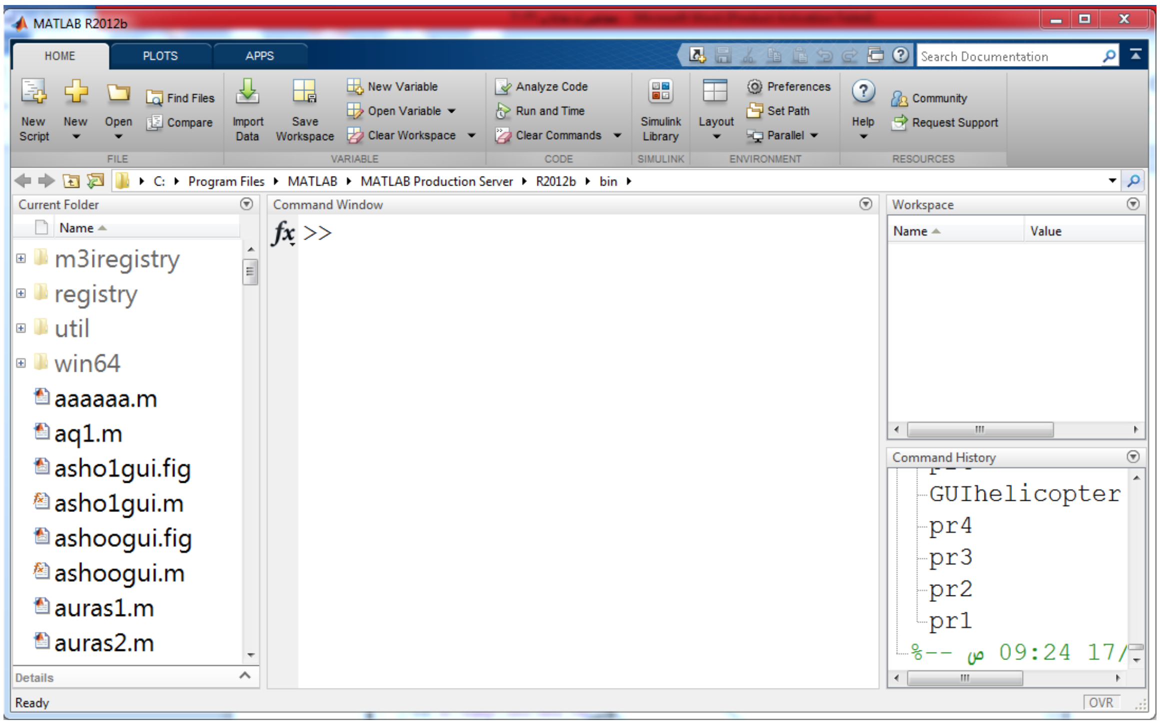
NUMERICAL ANALYSIS USING MATLAB

MATLAB (an abbreviation of "**MAT-LAB"**, "***matrix laboratory***") is a programming language and numeric computing environment which allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages.

Starting MATLAB

After logging into your account, you can enter MATLAB by double-clicking on the MATLAB shortcut icon on your Windows desktop. When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains *other* windows. The major tools within or accessible from the desktop are:

1. **The Command Window**
2. **The Command History**
3. **The Workspace**
4. **The Current Directory**
5. **The Help Browser**
6. **The Start button**



Built- in functions, Help:

MATLAB has many built-in functions. It has many mathematical functions, e.g. **abs** for absolute value, **tan** for tangent, etc. The functions are grouped together logically in what are called “help topics”. The **help** command can be used in MATLAB to find out what functions are built-in, and how to use them. Just typing “help” at the prompt will show a list of the help topics.

Using MATLAB as a calculator

As an example of a simple interactive calculation, just type the expression you want to evaluate. Let's start at the very beginning. For example, let's suppose you want to calculate the expression, 1 + 2 * 3. You type it at the prompt command (>>) as follows,

```
>> 1+2*3
ans =
7
```

You will have noticed that if you do not specify an output variable, MATLAB uses a default variable ans, short for answer, to store the results of the current calculation. Note that the variable ans is created (or overwritten, if it is already existed). To avoid this, you may assign a value to a variable or output argument name. For example,

```
>> x = 1+2*3
x =
7
>> 4*x
ans = 28.0000
```

Table 1.1: Basic arithmetic operators

SYMBOL	OPERATION	EXAMPLE
$+$	Addition	$2+3$
$-$	Subtraction	$2-3$
$*$	Multiplication	$2*3$
$/$	Division	$2/3$

Creating MATLAB variables

After learning the minimum MATLAB session, we will now learn to use some additional operations. MATLAB variables are created with an assignment statement. The syntax of variable assignment is variable name = a value (or an expression).

For example,

```
>> x = expression
```

where expression is a combination of numerical values, mathematical operators, variables, and function calls. On other words, expression can involve:

1- manual entry

2- built-in functions

3- user-defined functions

Making corrections

To make corrections, we can, of course retype the expressions. But if the expression is lengthy, we make more mistakes by typing a second time. A previously typed command can be recalled with the up-arrow key ↑. When the command is displayed at the command prompt, it can be modified if needed and executed.

Controlling the hierarchy of operations or precedence

Let's consider the previous arithmetic operation, but now we will include *parentheses*. For example, 1 + 2 * 3 will become (1 + 2) * 3

```
>> (1+2)*3
ans =
9
```

and, from previous example:

```
>> 1+2*3
ans =7
```

By adding parentheses, these two expressions give different results: 9 and 7. Therefore, to make the evaluation of expressions unambiguous, MATLAB has established a series of rules. The order in which the arithmetic operations are evaluated is given in Table 1.2. MATLAB arithmetic operators obey the same *precedence* rules as those in most computer programs. For operators of *equal* precedence, evaluation is from *left* to *right*.

Table 1.2: Hierarchy of arithmetic operations

PRECEDENCE	MATHEMATICAL OPERATIONS
First	The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward.
Second	All exponentials are evaluated, working from left to right
Third	All multiplications and divisions are evaluated, working from left to right
Fourth	All additions and subtractions are evaluated, starting from left to right

Now, consider another example:

$$\frac{1}{2+3^2}+\frac{4}{5}\times\frac{6}{7}$$

In MATLAB, it becomes

```
>> 1/(2+3^2)+4/5*6/7
ans =0.7766
```

or, if parentheses are missing,

```
>> 1/2+3^2+4/5*6/7
ans =10.1857
```

So here what we get: two different results. Therefore, we want to emphasize the importance of precedence rule in order to avoid ambiguity.

PRECEDENCE

Consider the mathematical expression $a(b + c)$ which you might read as "a times b plus c" which would appear to translate to the MATLAB command $a * b + c$. Hopefully you can see that this actually is equal to $ab + c$. The correct MATLAB command for $a(b + c)$ is $a * (b + c)$. The brackets have been used to force MATLAB to first evaluate the expression $(b + c)$ and then to multiply the result by a.

Recall that the operation brackets take precedence over exponent and the operations of division and multiplication take precedence over addition and subtraction.

Example

Determine the value of the expression $a\big(b + c(c + d)\big)a$, where $a = 2$, $b = 3$, $c = -4$ and $d = -3$.

$\gg a = 2; b = 3; c = -4; d = -3;$

$\gg a * \big(b + c * (c + d)\big) * a$

This gives the answer 124.

Example

Evaluate the MATLAB expressions

$1 + 2/3 * 4 - 5;\qquad 1/2 + 3/4 * 5;\qquad 5 - 2 * 3 * (2 + 7);$

$1/2/3/4;\qquad (1 + 3) * (2 - 3)/3 * 4;\quad \big(2 - 3 * (4 - 3)\big) * 4/5$

by hand and then check answer with MATLAB.

H. W.

Use MATLAB to calculate the expression $b - \frac{a}{b+\frac{b+a}{ca}}$

Where $a = 3,\ b = 5$ and $c = -3$.

Controlling the appearance of floating point number

MATLAB by default displays only 4 decimals in the result of the calculations, for example: -163.6667. However, MATLAB does numerical calculations in *double* precision, which is 15 digits. The command format controls how the results of computations are displayed. Here are some examples of the different formats together with the resulting outputs.

Format: the way in which numbers appear

Before we proceed we stop to discuss this important topic. This can be simply illustrated by the following example:

```
>> format short
>> x=-163.6667
```

If we want to see all 15 digits, we use the command format long:

```
>> format long
>> x= -1.636666666666667e+002
```

To return to the standard format, enter format short, or simply format. There are several other formats. For more details, see the MATLAB documentation, or type help format.

Note: - Up to now, we have let MATLAB repeat everything that we enter at the prompt (>>). Sometimes this is not quite useful, in particular when the output is pages in length. To prevent MATLAB from echoing what we type, simply enter a semicolon (;) at the end of the command. For example,

```
>> x=-163.6667;
```

and then ask about the value of x by typing,

```
>> x
x = -163.6667.
```

Example

Here we give an example of the simple use of brackets:

$\gg$ format rat

$\gg a = 2;\ \ b = 3;\ \ c = 4;$

$\gg a * (b + c)$

$\gg a * b + c$

$\gg \frac{a}{b} + c$

$\gg a/(b + c)$

In this example you should get the answers, 14, 10, $\frac{14}{3}$ and $\frac{2}{7}$. Hopefully this gives you some ideas that brackets make MATLAB perform those calculations first. (The command format rat has been used to force the results to be shown as rational).

Example

Consider the following code

$$s = \left[\frac{1}{2} \quad \frac{1}{3} \quad pi \quad sqrt(2)\right];$$

Format short; s

Format long; s

Format rat; s

Format;   s

This generates the output

```
>> Format short; s=[1/2  1/3   pi   sqrt(2)]
s =
    0.5000    0.3333    3.1416    1.4142
>> Format long; s
s =
 0.500000000000000   0.333333333333333   3.141592653589793   1.414213562373095
>> Format rat; s
```

```
s =

      1/2          1/3         355/113      1393/985

>> Format; s

s =

   0.5000    0.3333    3.1416    1.4142
```

The default option is format short. The above options are

Short - 5 digits

Long - 15 digits

Rat - try to represent the answer as a rational.

Managing the workspace

The contents of the workspace persist between the executions of separate commands. Therefore, it is possible for the results of one problem to have an effect on the next one. To avoid this possibility, it is a good idea to issue a clear command at the start of each new independent calculation.

```
>> clear
```

The command clear or clear all removes all variables from the workspace. This frees up system memory. In order to display a list of the variables currently in the memory, type: who**.**

Entering multiple statements per line

It is possible to enter multiple statements per line. Use commas (,) or semicolons (;) to enter more than one statement at once. Commas (,) allow multiple statements per line without suppressing output.

```
>> a=7; b=cos(a), c=cosh(a)
>> b =
0.6570
>> c =
548.3170
```

Mathematical functions

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

Table 2.1: Elementary functions

cos(x)	Cosine	abs(x)	Absolute value
sin(x)	Sine	sign(x)	Signum function
tan(x)	Tangent	max(x)	Maximum value
acos(x)	Arc cosine	min(x)	Minimum value
asin(x)	Arc sine	ceil(x)	Round towards $+\infty$
atan(x)	Arc tangent	floor(x)	Round towards $-\infty$
exp(x)	Exponential	round(x)	Round to nearest integer
sqrt(x)	Square root	rem(x)	Remainder after division
log(x)	Natural logarithm	angle(x)	Phase angle
log10(x)	Common logarithm	conj(x)	Complex conjugate

In addition to the elementary functions, MATLAB includes a number of predefined constant values. A list of the most common values is given in Table 2.2.

Table 2.2: Predefined constant values

pi	The π number, $\pi = 3.14159\ldots$
i,j	The imaginary unit i, $\sqrt{-1}$
Inf	The infinity, ∞
NaN	Not a number

Linear spacing

On the other hand, there is a command to generate linearly spaced vectors: linspace. It is similar to the colon operator (:), but gives direct control over the number of points. For example,

y = linspace(a,b) generates a row vector y of 100 points linearly spaced between and including a and b.

y = linspace(a,b,n) generates a row vector y of n points linearly spaced between and including a and b. This is useful when we want to divide an interval into a number of subintervals of the same length.

Example:

```
>> theta = linspace(0,2*pi,101)
```

divides the interval $[0; 2\pi]$ into 100 equal subintervals, then creating a vector of 101 elements.

We illustrate here some typical examples which related to the elementary functions previously defined. As a first example, the value of the expression $y = e^{-a}\sin(x) + 10\sqrt{y}$, for $a = 5$, $x = 2$, $y = 8$ is computed by

```
>> a = 5; x = 2; y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y  =
   28.2904
```

The subsequent examples are

```
>> log(142)
ans    =
        4.9558

>> log10(142)
ans    =
        2.1523
```

Note the difference between the natural logarithm log(x) and the decimal logarithm (base 10) log10(x).

To calculate $\sin(\pi/4)$ and e^{10}, we enter the following commands in MATLAB,

```
>> sin(pi/4)
ans    =
        0.7071

>> exp(10)
ans    =
        2.2026e+004
```

VECTORS AND MATRICES IN MATLAB

Entering a vector

A vector is a special case of a matrix. The purpose of this section is to show how to create vectors and matrices in MATLAB. As discussed earlier, an array of dimension 1**n* is called a *row* vector, whereas an array of dimension *m** 1 is called a *column* vector. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, v, type:

```
>> v = [1 4 7 10 13]
v =
1 4 7 10 13
```

Column vectors are created in a similar way; however, semicolon (;) must separate the components of a column vector,

```
>> w = [1;4;7;10;13]
w =
1
4
7
10
13
```

MATLAB is written to work with vectors and matrices; A matrix looks like a table with rows and columns; an *m by n* (or *m x n*) matrix has m rows by n columns (these are the dimensions of the matrix). A scalar is an even simpler case; it is a *1 by 1* matrix, or in other words, a single value.

Transposing a vector:

Transposing a vector or a matrix is done either by the function (transpose) or by adding dot-prime (') after the matrix. A *row* vector is converted to a *column* vector using the *transpose* operator. The *transpose* operation is:

```
>> v=[1 4 7 10 13]
>>w = v'
w =
1
4
7
10
13
```

Thus, v(1) is the first element of vector v, v(2) its second element, and so on. Furthermore, to access *blocks* of elements, we use MATLAB's colon notation (:). For example, to access the first three elements of v, we write:

```
>> v(1:3)
ans =1 4 7
```

If v is a vector, writing

>> v(:) produces a column vector, whereas writing

>> v(1:end) produces a row vector, such that:

```
>> v(:)
ans =
    1
    4
    7
   10
   13
>> v(1:end)
ans =
    1     4     7    10    13
```

Entering a matrix

A matrix is an array of numbers. To type a matrix into MATLAB you must

- begin with a square bracket, [
- separate elements in a row with spaces or commas (,)
- use a semicolon (;) to separate rows
- end the matrix with another square bracket].

Here is a typical example. To enter a matrix A, such as,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

type,

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

MATLAB then displays the 3×3 matrix as follows,

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

Note that the use of semicolons (;) here is different from their use mentioned earlier to suppress output or to write multiple commands in a single line. Once we have entered the matrix, it is automatically stored and remembered in the *Workspace*. We can refer to it simply as matrix A. We can then view a particular element in a matrix by specifying its location. We write:

```
>> A(2,1)
ans =
4
```

A(2,1) is an element located in the second row and first column. Its value is 4.

Matrix indexing

We select elements in a matrix just as we did for vectors, but now we need two indices. The element of row *i* and column *j* of the matrix A is denoted by A(i,j). Thus, A(i,j) in MATLAB refers to the element *Aij* of matrix A. The *first* index is the *row* number and the *second* index is the *column* number. For example, A(1,3) is an element of *first* row and *third* column. Here, A(1,3)=3.

Correcting any entry is easy through indexing. Here we substitute A(3,3)=9 by A(3,3)=0, the result is:

```
>> A(3,3) = 0
A    =
     1    2    3
     4    5    6
     7    8    0
```

Colon operator in a matrix

The colon operator can also be used to pick out a certain row or column. For example, the statement A(m:n,k:l) specifies rows *m* to *n* and column *k* to *l*. Subscript expressions refer to portions of a matrix. For example,

```
>> A(2,:)
ans  =
        4    5    6
```

is the second row elements of `A`.

The colon operator can also be used to extract a sub-matrix from a matrix `A`.

```
>> A(:,2:3)
ans  =
        2    3
        5    6
        8    0
```

`A(:,2:3)` is a sub-matrix with the last two columns of `A`.

A row or a column of a matrix can be deleted by setting it to a *null* vector, [ ].

```
>> A(:,2)=[]
ans  =
        1    3
        4    6
        7    0
```

Creating a sub-matrix

To extract a submatrix B consisting of rows 2 and 3 and columns 1 and 2 of the matrix A, do the following

```
>> B = A([2 3],[1 2])
B =
4 5

7 8
```

Another example:

```
>> a=[1:4;5:8;9:12]

a =

    1    2    3    4

    5    6    7    8

    9   10   11   12

>> b=a([1 2],[3 4])
b =
    3    4
    7    8
```

Dimension

To determine the *dimensions* of a matrix or vector, use the command size. For example,

```
>> size(A)
ans 3 3        means 3 rows and 3 columns.
```

Transposing a matrix

The *transpose* operation is denoted by an apostrophe or a single quote ('). It flips a matrix about its main diagonal and it turns a row vector into a column vector, if A=[1:3;4:6;7:9], then:

```
>> A'
ans =
1 4 7
2 5 8
3 6 9
```

By using linear algebra notation, the transpose of $m * n$ real matrix A is the $n * m$ matrix that results from interchanging the rows and columns of A. The transpose matrix is denoted A^T. It is possible to refer to a whole row (or column) of a matrix.

For example

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

   4 5 6
```

Returns the second row of the matrix A. here the colon indicates all the elements along a particular row. Alternatively to refer to a particular column:

```
>> A(:,3)

ans =

   3

   6

   9
```

which refers to the third column of the matrix A.

Example: Compute A+B, A-B, 5+A, A*B, A^2 for:

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

ans =

    6    8
   10   12
```

This adds each element of A to the corresponding element in B.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

```
>> A + 5
ans =  6     7
       8     9.
```

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}+5=\begin{bmatrix}1+5 & 2+5\\3+5 & 4+5\end{bmatrix}=\begin{bmatrix}6 & 7\\8 & 9\end{bmatrix}$$

This adds 5 to each element of A.

```
>> A-B

ans =

   -4   -4
   -4   -4
>> A*B

ans =

   19   22
   43   50
>> A^2

ans =

    7   10
   15   22
```

Multiply matrix A by itself.

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\times\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}=\begin{bmatrix}1\times1+2\times3 & 1\times2+2\times4\\3\times1+4\times3 & 3\times2+4\times4\end{bmatrix}=\begin{bmatrix}7 & 10\\15 & 22\end{bmatrix}$$

Error messages

If we enter an expression incorrectly, MATLAB will return an error message. For example, in the following, we left out the multiplication sign, *, in the following expression

```
>> x = 10;
>> 5x
Error: Unexpected MATLAB expression.
```

An error message appears if we have two vectors a and b, we cannot multiply them because they have the same dimension. Suppose however that what we really want to achieve is to multiply the elements of vector a by the elements of vector b in an element by element sense. We can also do a term by term division , for example:

```
≫ a = [1  2  3];
≫ b = [4  5  6];
≫ a.∗ b
ans=
      4   10   18
```

The answer shows that MATLAB has returned a vector containing the elements $[a1b1, a2b2, a3b3]$.

We can also do a term by term division with

$\gg$ a = [1 2 3];

$\gg$ b = [4 5 6];

$\gg$ a./b

ans=

```
        0.2500    0.4000       0.5000
```

The result is, as we would expect,

$$\left[\frac{a1}{b1},\frac{a2}{b2},\frac{a3}{b3}\right].$$

Example

We shall create two vectors running from one to six and from six to one and then demonstrate the use of the dot arithmetical operations:

```
s = 1:6;
t = 6:-1:1;
s + t
s - t
s.* t
s./t
s. ^2
1./s
>> s=1:6;
>> t=6:-1:1;
>> s+t
ans =

    7     7     7     7     7     7
>> s-t
ans =
   -5    -3    -1     1     3     5

>>  s.*t
ans =
    6    10    12    12    10     6
>> s./t
ans =
   0.1667    0.4000    0.7500    1.3333    2.5000    6.0000
>> s.^2
```

```
ans =

     1     4     9    16    25    36
>> 1./s

ans =

    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667
```

Special matrices

MATLAB provides a number of special matrices (see Table 2.5). These matrices have interesting properties that make them useful for constructing examples and for testing algorithms.

Table 2.5 Special matrices

hilb	Hilbert matrix
invhilb	Inverse Hilbert matrix
magic	Magic matrix

For example:

```
>> hilb(2)
ans =
    1.0000    0.5000
    0.5000    0.3333
>> invhilb(2)
ans =
     4    -6
    -6    12
>> magic(3)
ans =
     8     1     6
     3     5     7
     4     9     2
```

Matrix generators

MATLAB provides functions that generate elementary matrices. The matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions zeros, ones, and eye, respectively.

Table 2.4: Elementary matrices

`eye(m,n)`	Returns an m-by-n matrix with 1 on the main diagonal
`eye(n)`	Returns an n-by-n square identity matrix
`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones
`diag(A)`	Extracts the diagonal of matrix A
`rand(m,n)`	Returns an m-by-n matrix of random numbers

Moreover, many other important functions also available in MATLAB, thus:

round(x)	Round towards nearest integer. round(X) rounds the elements of X
ceil(x)	Round towards plus infinity. ceil(X) rounds the elements of X to the nearest integers towards infinity.
floor(x)	rounds a number down to the nearest integer
fix(x)	rounds a number to the nearest integer towards zero
rem(x, y)	the remainder left after division
mod(x, y)	the signed remainder left after division
abs(x)	the absolute value of x
sign(x)	Signum function. For each element of X, sign(X) returns 1 if the element is greater than zero, 0 if it equals zero and -1 if it is less than zero.
factor(x)	the prime factors of x

Example:

Calculate the expressions:

floor ([-2.33   2.66]), round ([-2.33   2.66]), fix ([-2.33   2.66]), rem (23, 4), factor (24) and g c d (18,   81).

Solution:

```
>>floor ([-2.33    2.66])

ans =

   -3    2

>>round ([-2.33   2.66])

ans =  -2     3

>> ceil([-2.33     2.66])

ans =

   -2     3

fix ([-2.33    2.66])

ans =
   -2     2
rem (23,  4)
ans =
     3
>> factor(24)
ans =
     2     2     2     3
g c d (18,   81)=
```

ans= 9 where g c d is **the greatest common divisor**.

```
>> eye(2)
ans =
     1     0
     0     1
>> eye(3,3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> zeros(2,3)
ans =
 0 0 0
 0 0 0
>> ones(3,4)
ans =
 1 1 1 1
 1 1 1 1
 1 1 1 1
```

```
>> a=[1:3;4:6]
a =
     1     2     3
     4     5     6
>> diag(a)
```

```
ans =
    1
    5
>> rand(3,2)
ans =
   0.8147    0.9134
   0.9058    0.6324
   0.1270    0.0975
```

Mathematical Functions

Trigonometric functions such as sin(sine), cos(cosine) and tan(tangent)(with their inverses) being obtained by appending an a letter: asin, acos or atan. These functions measured in **radians**. It should be noted that these functions should operate on an input; the syntax of the commands is sin(x) rather than sin x. Exponential functions exp, log, log10 and power functions ^. Notice that the default in MATLAB for a logarithm is the natural logarithm lnx. The final command takes two arguments (and hence is a binary operation) so that a^b gives a^b.

Example

Calculate the expressions: sin60 (and the same quantity squared), exp(ln(4)), cos45-sin45, ln exp(2+cos π)and tan30/(tan($\frac{\pi}{4}$) +tan($\frac{\pi}{3}$)).

$$\gg x = \sin(\frac{60}{180} * \text{pi})$$

$x =$

0.8660

$\gg y = x\text{^}2$

$y = \quad 0.7500$

$\gg \exp(\log(4))$

$\text{ans} =$

4

$$\gg z = \frac{45}{180} * \text{pi};\ \ \cos(z) - \sin(z)$$

$\text{ans} =$

$1.1102e^{-16}$

$\gg \log(\exp(2+\cos(\text{pi})))$

$\text{ans} =$

1

$\gg \tan(\frac{30}{180} * \text{pi})/(\tan\left(\frac{\text{pi}}{4}\right) + \tan\left(\frac{\text{pi}}{3}\right))$

$\gg \text{ans} =$

0.2113

The values of these expressions should be $\frac{\sqrt{3}}{2}, \frac{3}{4}, 4, 0, 1$ and $1/(3+\sqrt{3})$.

BASIC PLOTTING

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible.

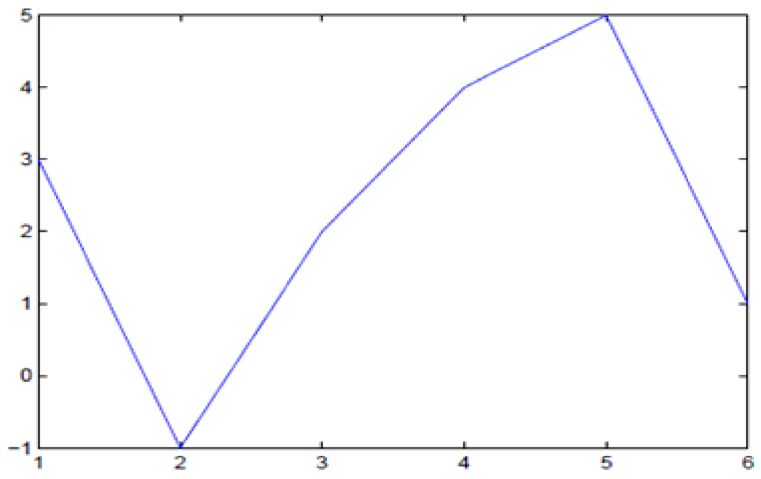
Creating simple plots

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

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

NOTE:

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



```
>> x = 0:pi/100:2*pi;
>> y = sin(x);
>> plot(x,y)
```

Notes:

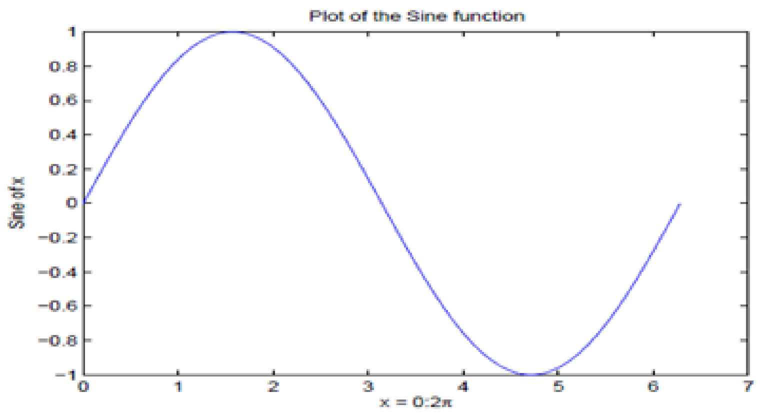
- 0:pi/100:2*pi yields a vector that:

- starts at 0,
- takes steps (or increments) of $\pi/100$,
- stops when 2π is reached.

- If you omit the increment, MATLAB automatically increments by 1.

Adding titles, axis labels, and annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add x- and y-axis labels. Now label the axes and add a title. The character pi creates the symbol π. An example of 2D plot is shown in Figure below.



```
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
>> title('Plot of the Sine function')
```

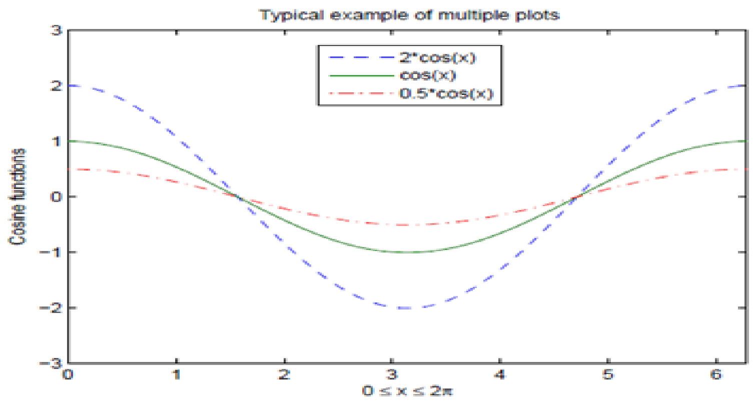
The color of a single curve is, by default, blue, but other colors are possible. The desired color is indicated by a third argument. For example, red is selected by plot(x,y,'r'). Note the single quotes, ' ', around r.

Multiple data sets in one plot

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

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

>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])
```



By default, MATLAB uses *line style* and *color* to distinguish the data sets plotted in the graph. However, you can change the appearance of these graphic components or add annotations to the graph to help explain your data for presentation.

Specifying line styles and colors

It is possible to specify *line styles*, *colors*, and *markers* (e.g., circles, plus signs, . . .) using the plot command:
plot(x,y,'style_color_marker') where style_color_marker is a *triplet* of values from Table 2.3.

Table 2.3: Attributes for plot

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

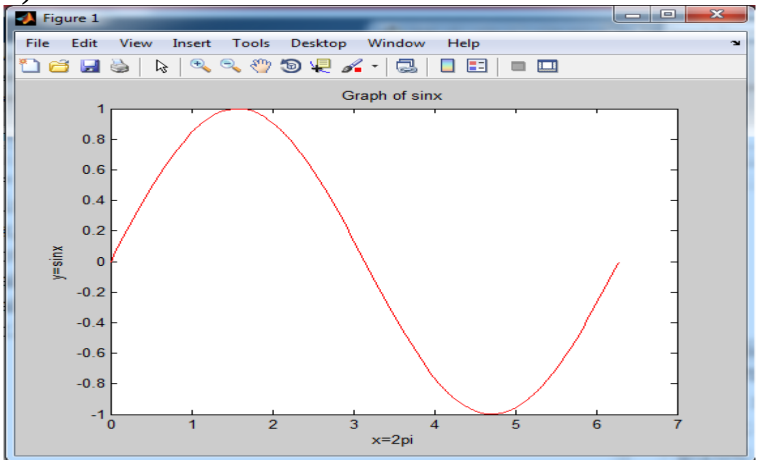
Example:

Create a two-dimensional line plot using the plot function. For example, plot the value of the sine function from 0 to 2π.

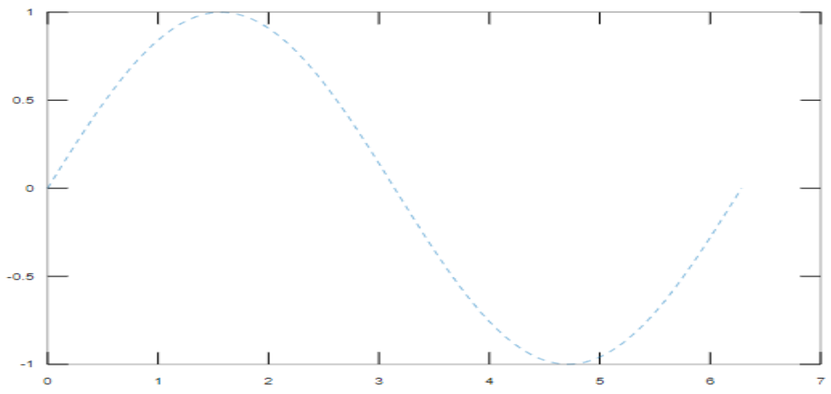
>>x = linspace(0, 2*pi, 100);

>>y = sin(x);

```
>>plot(x, y, 'r')
>> xlabel('x=2pi')
>> ylabel('y=sinx')
>> title('Graph of sinx')
```



Example: x = linspace(0, 2*pi, 100); y = sin(x); plot(x,y, '--')



Example: x = linspace(0, 2*pi, 100); y = exp(x); plot(x, y, 'r')
xlabel('x=2pi')
ylabel('y=exp(x)')
title('Graph of exp(x)')

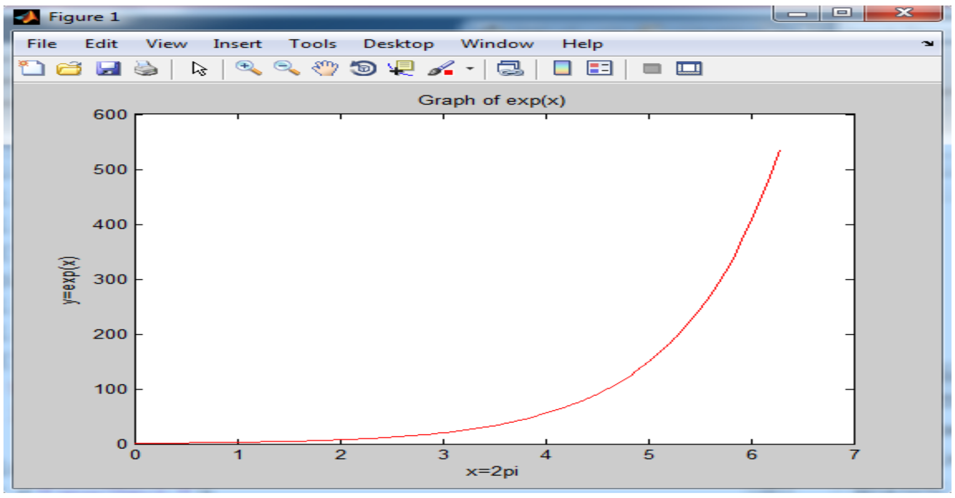


Figure 1
File Edit View Insert Tools Desktop Window Help
Graph of exp(x)
y=exp(x)
x=2pi