Tutorial lessons-1

>> who: In order to display a list of the variables currently in the memory, type

>> diary: to keep track of everything done during a MATLAB session

>> diary FileName

Entering 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

>> lookfor inverse: the command help

inverse will produce nothing. On the other hand, the command lookfor inverse will

produce detailed information, which includes the function of interest, inv.

>> lookfor FunctionName

Tutorial lessons -2

2.1 Mathematical functions

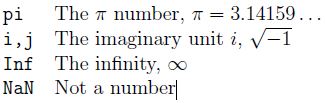
Typing help elfun and help specfun calls up full lists of *elementary* and *special*

functions respectively.

There is a long list of mathematical functions that are *built* into MATLAB. These

functions are called *built-ins*.

**A list of the most common values**



* To avoid any possible confusion, it is suggested to use instead ii or jj as loop indices.

2.2 Basic plotting

>> 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 2*Pi* is reached.
* If you omit the increment, MATLAB automatically increments by 1.

Adding titles, axis labels, and annotations

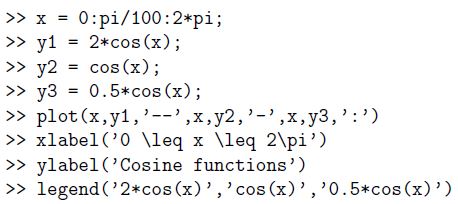
>> xlabel('x = 0:2\pi')

>> ylabel('Sine of x')

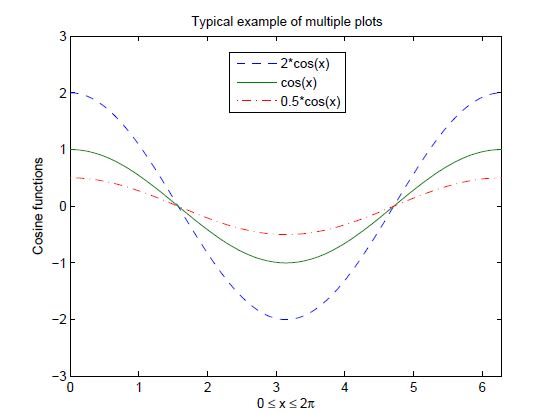
>> title('Plot of the Sine function')

Multiple data sets in one plot

For example, these statements plot three related functions of *x*:

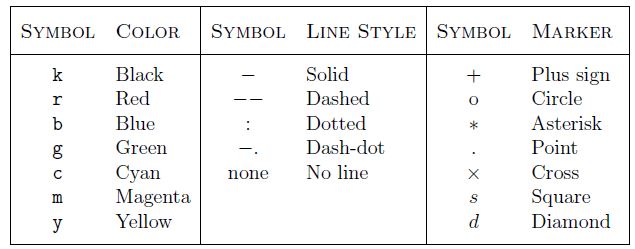


W:\Desktop\Capture.JPG



Specifying line styles and colors

>>plot(x,y,'style\_color\_marker')



Matrix generation:

Entering a vector: A vector is a special case of a matrix. An array of dimension 1\**n* is called a *row* vector, whereas an array of dimension *m\** 1 is called a *column* vector.

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

A *row* vector is converted to a *column* vector using the *transpose* operator.

The *transpose* operation is denoted by an apostrophe or a single quote (').

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

Or, all elements from the third through the last elements,

>> v (3,end)

ans = 7 10 13

If v is a vector, writing

>> v (:)

Produces a column vector, whereas writing

>> v (1:end)

Produces a row vector.

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, ].c

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

We can then view a particular element in a matrix by specifying its location. We write,

>> A (2, 1)

ans = 4

Matrix indexing:

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

A (3, 3) =0.

Single elements of a matrix are accessed as A(i,j), where *i ¸* 1 and *j ¸* 1. Zero or negative subscripts are not supported in MATLAB.

Colon operator:

Often we must deal with matrices or vectors that are too large to enter one element at a time. For example, suppose we want to enter a vector *x* consisting of points

(0,0*.*1,0.2*,* 0.3…5). We can use the command

>> x = 0:0.1:5;

The row vector has 51 elements.

Linear spacing:

This 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.

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

Divides the interval [0*;* 2\*Pi] into 100 equal subintervals, then creating a vector of 101 elements.

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*.

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

A =

1 2 3

4 5 6

7 8 9

>> A (2, :)

ans = 4 5 6

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

>> A (:, 1:2)

ans =

1 2

4 5

7 8

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

A (:, 2) =[]

A =1 3

4 6

7 9

To delete a row or column of a matrix, use the *empty vector* operator, [ ].

>> A(3,:) = []

A =

1 2 3

4 5 6

Third row of matrix A is now deleted.

To restore the third row, we use a technique for creating a matrix

>> A = [A(1,:);A(2,:);[7 8 0]]

A =

1 2 3

4 5 6

7 8 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

To interchange rows 1 and 2 of A, use the vector of row indices together with the colon operator.

>> C = A ([2 1 3], :)

C =

4 5 6

1 2 3

7 8 9

To create a vector version of matrix A, do the following

>> A (:)

ans = 1

2

3

4

5

6

7

8

9

The keyword end, used in A (end,:), denotes the last index in the specified dimension.

A (end, end:-1:1)

ans = 9 8 7

>> A (end:-1:1, end)

Ans = 9

6

3