



Introduction to MATLAB

MATLAB

Built in constants and variables

ans	most recent answer
eps	small constant $\sim 10^{-16}$
i or j	imaginary unit
inf	infinity
pi	3.14159 ...

Standard mathematical operations

\pm	addition, subtraction
*	multiplication
/	division
^	exponentiation e.g. $y^n = y^n$

Some common functions

sin(x)	sine
cos(x)	cosine
exp(x)	exponential
sqrt(x)	square root
log(x)	natural log
log10(x)	log to base 10
abs(x)	absolute value, magnitude of complex quantity
angle(x)	phase angle
real(x)	real part of
imag(x)	imaginary part of

Generation of (row) vectors

```
>> x = 0:0.1:0.5
x =
    0    0.1000    0.2000    0.3000    0.4000    0.5000
```

```
>> y = linspace(0, 0.5, 4)
y =
    0    0.1667    0.3333    0.5000
```

```
>> z = [ 1 2 3 4 5]
z =
    1     2     3     4     5
```

Suppression of echoing of output (put semi-colon at end of line)

```
>> z = [ 1 2 3 4 5];
>>
>> z
z =
    1     2     3     4     5
```

Comment line

```
>> % This is a comment
>>
```

Functions can have vector (matrix) arguments

```
>>      x = [ 1 2 3 4 5];
>>      y = exp(x)
y =
  2.7183  7.3891  20.0855  54.5982  148.4132
```

Element by element operations on vector-valued functions

\pm	addition, subtraction
$.*$	multiplication
$./$	division
$.^$	exponentiation

```
>>      x = [ 1 2 3 4]
x =
   1   2   3   4
>>      y = x + 2
y =
   3   4   5   6
>>      z = x.^2
z =
   1   4   9  16
>>      f = x .* x.^2
f =
   1   8  27  64
```

MATLAB does complex arithmetic with scalars and vectors

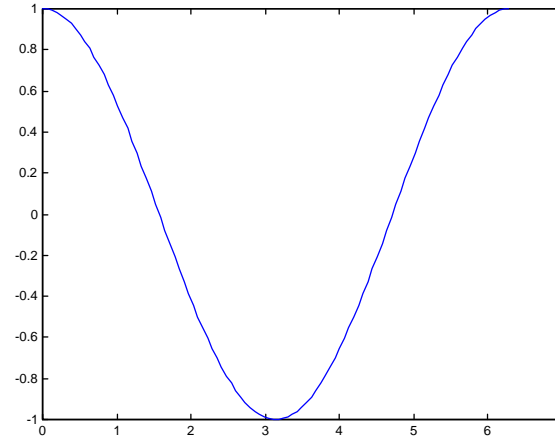
```
>>      z = 3 + 4*i
z =
  3.0000+ 4.0000i
>>      y = i*z
y =
 -4.0000+ 3.0000i
>>      x = [ 1 + 3*i  2*i]
x =
  1.0000+ 3.0000i    0+ 2.0000i
>>y = [ i  i]
y =
  0+ 1.0000i    0+ 1.0000i
>>      z = x .*y
z =
 -3.0000+ 1.0000i   -2.0000
```

Common functions take complex arguments (scalars or vectors)

```
>>      exp(i*pi)
ans =
  -1
>>      x = [ 0 pi/2  pi] ;
>>      exp(i*x)
ans =
  1.0000  0+ 1.0000i   -1.0000
```

Simple plotting

```
>> x = linspace( 0, 2*pi, 100);  
>> y = cos(x);  
>> plot(x, y)
```



Multiple plots on same graph

```
>> x = linspace( 0, 2*pi, 100);  
>> y1 = cos(x);  
>> y2 = sin(x);  
>> plot(x, y1, x, y2)
```

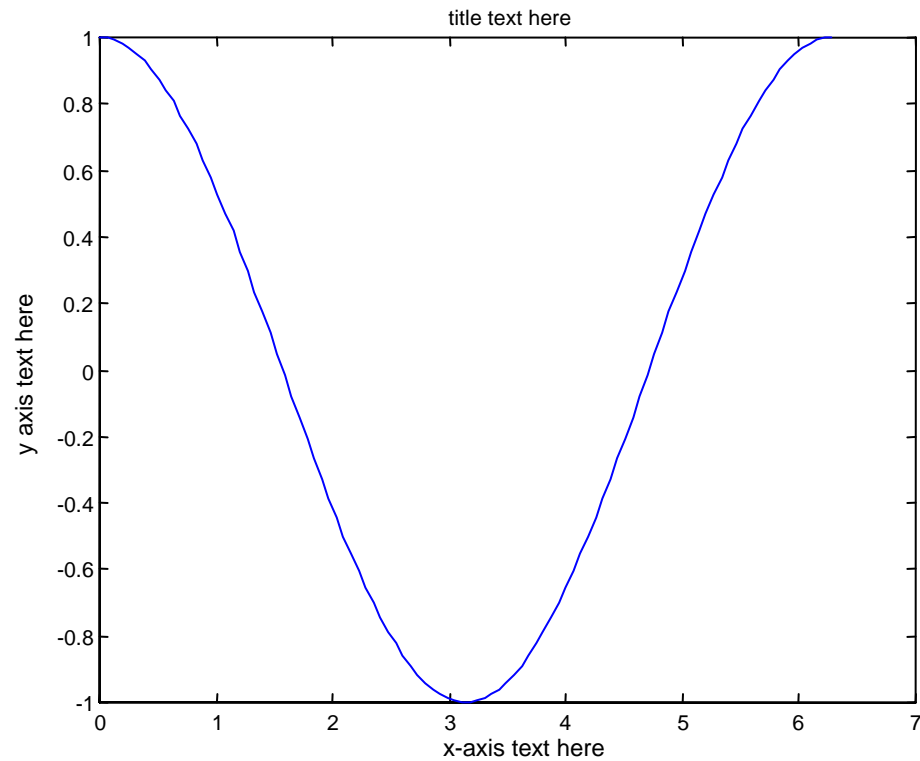
or

```
>> x = linspace(0, 2*pi, 100);  
>> y1 = cos(x);  
>> plot(x, y1)  
>> hold on  
>> y2 = sin(x);  
>> plot(x, y2)  
>> hold off
```



Adding a x-axis label, a y-axis label, and a title to a plot

```
>> x = linspace( 0, 2*pi, 100);  
>> y = cos(x);  
>> plot(x, y)  
>> xlabel(' x-axis text here')  
>> ylabel(' y axis text here')  
>> title('title text here')
```



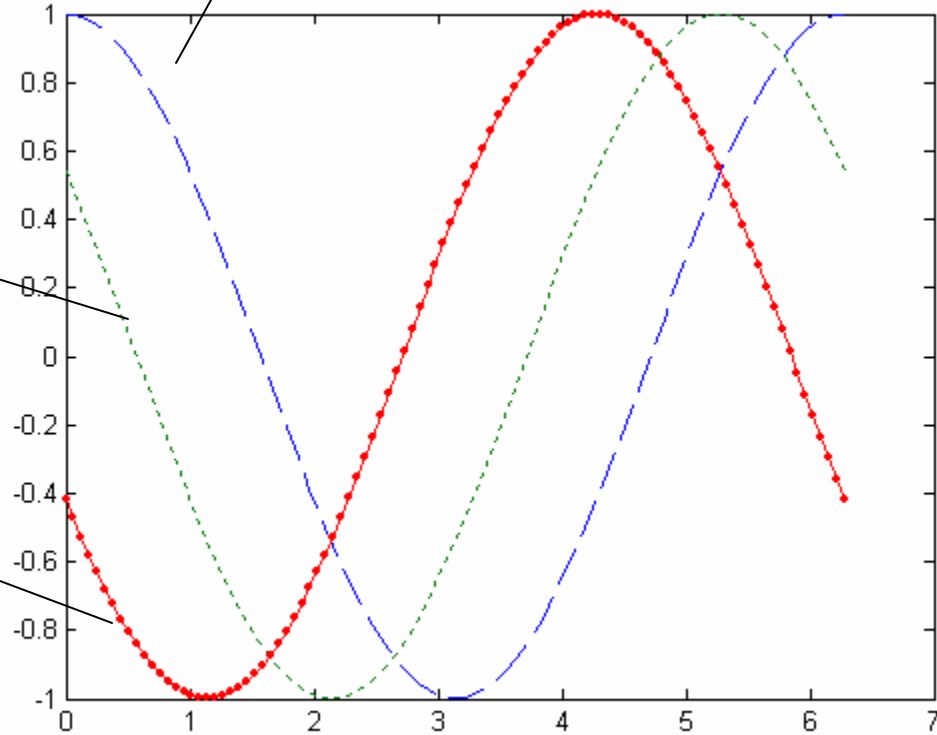
Plotting with different line styles

```
>> x = linspace( 0, 2*pi, 100);  
>> y1 = cos(x);  
>> y2 = cos(x+1);  
>> y3 = cos(x+2);  
>> plot(x, y1, '--',x,y2,':',x, y3,'-')
```

y1 dashed line

y2 dotted line

y3 dot-dash

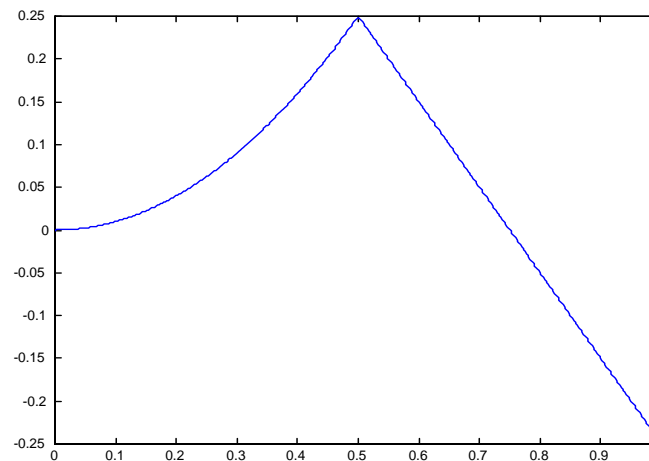


Logical (0-1) vectors

```
>>      x = [ 1 2 3 4 5];  
>>      x > 3  
ans =  
      0      0      0      1      1  
>>      x <= 4  
ans =  
      1      1      1      1      0
```

Use of logical vectors for defining piecewise function

```
>>      x = linspace(0, 1, 500);  
>>      y = (x.^2).*(x < 0.5) + (0.75 - x).*(x >= 0.5);  
>>      plot(x, y)
```



Defining parts of vectors

```
>>      x = [ 5 6 3 8 7 9];  
>>      x(1:3)  
ans =  
    5    6    3  
>>      x(2:4)  
ans =  
    6    3    8  
>>      x(3:end)  
ans =  
    3    8    7    9
```

Other vector operations

Vector magnitude, length

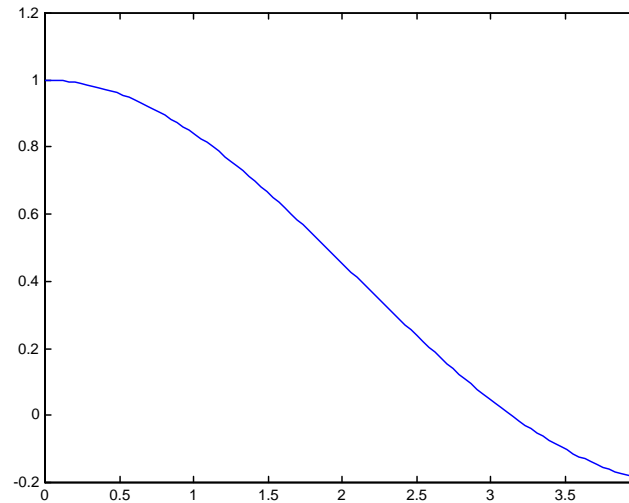
```
>>      x = [3 4];  
>>      norm(x)  
ans =  
    5  
  
>>      length(x)  
ans =  
    2
```

Use of eps to avoid division by zero

```
>> x=linspace(0, 4, 100);  
>> y=sin(x)./x;
```

Warning: Divide by zero.

```
>> x = x + eps*(x == 0);  
>> y=sin(x)./x;  
>> plot(x,y)
```



Use of inf to evaluate expression when a variable goes to infinity

```
function y = infinitytest(x)  
x = x + eps*( x == 0);  
y = 3/(1 + 4/x);
```

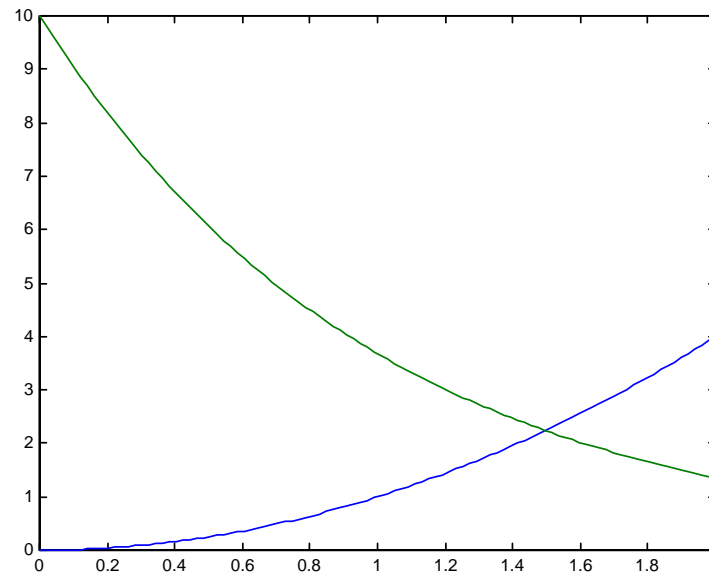
```
>> infinitytest(1)  
ans =  
    0.6000  
>> infinitytest(0)  
ans =  
    1.6653e-16  
>> infinitytest(inf)  
ans =  
    3
```

MATLAB Functions

Functions are defined in the editor and saved as M-files. The name of the M-file is the name of the function with a .m extension, e.g. myfunction.m etc. As shown in the example below, functions can have multiple outputs (as well as multiple inputs).

```
function [y , z] = test(x)
y = x.^2;
z = 10*exp(-x);
```

```
>> x = linspace(0, 2, 100);
>> [s, t] = test(x);
>> plot(x, s, x, t)
```



All the variables appearing in a function are local to that function, i.e. they do not change any similarly named variables in the MATLAB workspace.

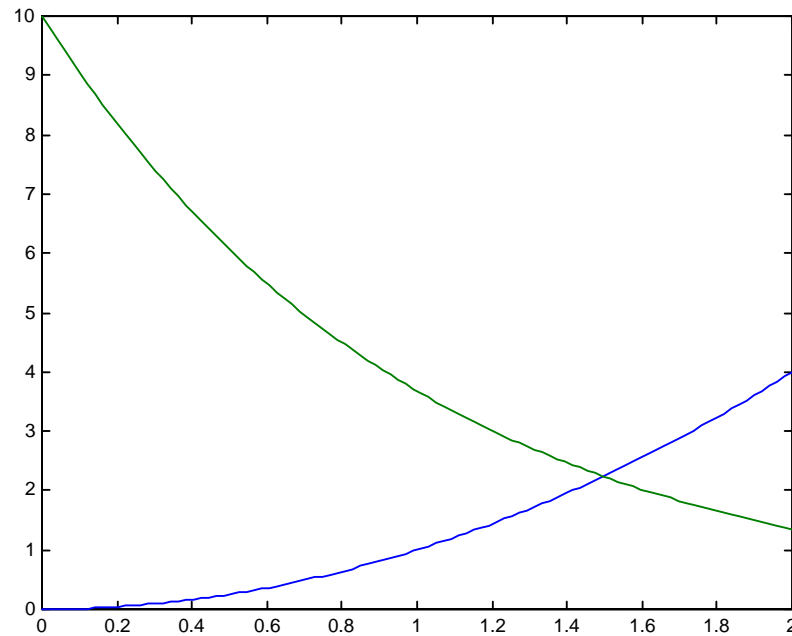
MATLAB Scripts

A MATLAB script is a sequence of ordinary MATLAB statements, defined in the editor and then saved as a file. The file has the name of the script followed by a .m extension, e.g. myscript.m.

Typing the script name at the MATLAB prompt then causes the script to be executed. Variables in the script change any values of variables of the same name that exist in the current MATLAB session.

```
% testscript  
x = linspace(0,2,100);  
y = x.^2;  
z = 10*exp(-x);  
plot(x,y,x,z)
```

```
>> testscript
```



MATLAB Matrices

MATLAB was designed to perform operations with matrices very effectively. Entering matrices manually is as easy as vectors:

```
>>      matrix = [ 4 0 3;
                  0 3 5;
                  3 5 7 ]

matrix =
     4     0     3
     0     3     5
     3     5     7
```

Accessing individual components

```
>>      matrix(2,3)
ans =
     5
```

Accessing rows or columns

```
>>matrix(:, 2)
ans =
     0
     3
     5

>>matrix(3, :)
ans =
     3     5     7
```

Some matrix functions

`size(M)` returns number of rows, `nr`, and number of columns, `nc`, as a vector `[nr , nc]`

`trace(M)` trace of `M` (sum of diagonal terms)

`det(M)` determinant of `M`

`M'` transpose of `M` (interchange rows and columns)

```
>> size(matrix)
```

```
ans =
```

```
3 3
```

```
>> trace(matrix)
```

```
ans =
```

```
14
```

```
>> det(matrix)
```

```
ans =
```

```
-43
```

```
>> matrix'
```

```
ans =
```

```
4 0 3
```

```
0 3 5
```

```
3 5 7
```

(no change since matrix is symmetric)

Multiplying matrices and vectors

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

```
>> v*matrix
```

```
ans =
```

```
13 21 34
```

```
>> vt = v'
```

```
vt =
```

```
1
```

```
2
```

```
3
```

```
>> matrix*vt
```

```
ans =
```

```
13
```

```
21
```

```
34
```

```
>> v*matrix*vt
```

```
ans =
```

```
157
```

transpose of vector v (turn row vector to column vector or vice-versa)

Special matrices

`zeros(m,n)` matrix of all zeros with m rows and n columns
`ones(m,n)` matrix of all ones with m rows and n columns
`eye(m,n)` identity matrix with m rows and n columns

```
>>            zeros(3,3)
```

```
ans =
```

```
  0  0  0  
  0  0  0  
  0  0  0
```

```
>>            ones(3,3)
```

```
ans =
```

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

```
>>            eye(3,3)
```

```
ans =
```

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

Solving a system of linear equations $Mx = b$

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

```
M =  
    1    3    5  
    2    1    1  
    4    3    6
```

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

```
b =
```

```
    3  
    2  
    1
```

```
>>      x = M\b
```

```
x =  
-0.0909  
 3.9091  
-1.7273
```

$$1x_1 + 3x_2 + 5x_3 = 3$$

$$2x_1 + 1x_2 + 1x_3 = 2$$

$$4x_1 + 3x_2 + 6x_3 = 1$$

Determining the eigenvectors and eigenvalues of a matrix M
(For a real symmetric matrix the eigenvalues are real and the eigenvectors are real and orthogonal to each other)

[eigenvects, eigenvals] = eig(M)

```
>>      [evects, evals] = eig(matrix)
evects =
```

```
-0.8752  0.3507  0.3332
 0.4783  0.7300  0.4881
 0.0720 -0.5865  0.8067
```

Eigenvectors (in columns)

```
evals =
```

```
3.7531    0    0
 0 -1.0172    0
 0    0 11.2641
```

Corresponding eigenvalues