MATLAB FORMULA LIST

Special Characters

[ ]	forms matrices
( )	forms subscripts
,	separates subscripts or matrix elements
;	separates commands or matrix rows
%	indicates comments
:	generates matrices
+	scalar and array addition
-	scalar and array subtraction
*	scalar multiplication
.*	array multiplication
/	scalar division
./	array division
^	scalar exponentiation
.^	array exponentiation
‘	transpose

Some simple tricks

Using the colon (“:”) character
Make a series of numbers from a given beginning and ending number
-*Example:*

>> 0:5 returns 0 1 2 3 4 5
>> -7:5 returns -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5

-*End Example*

Using the semicolon (“;”) character
The semicolon (“;”) can be used if you don’t want to see the output.
-*Example:*

Say for example you want make “x” to be the numbers between 1 and 100. Now you know what numbers “x” are going to be, and don’t want them to be outputted, then you can put a “;” at the end of the command and it won’t be.

NOTE: written like this, it will return every value of “x” as shown below

```
>> x=0:100
x =
  Columns 1 through 23
    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
17   18   19   20   21   22
  Columns 24 through 46
```

```
   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38
39   40   41   42   43   44   45
  Columns 47 through 69
   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61
62   63   64   65   66   67   68
  Columns 70 through 92
   69   70   71   72   73   74   75   76   77   78   79   80   81   82   83   84
85   86   87   88   89   90   91
  Columns 93 through 101
   92   93   94   95   96   97   98   99  100
```

NOTE: by putting a ";" at the end, you prevent it from returning every value of "x"

```
>> x=0:100;
>>
```

-*End Example*

How to set up a matrix

-*Example:*

NOTE: "," separate columns and ";" separate rows

```
>> [2] returns
    2
>> [2,3] returns
    2    3
>> [2;3] returns
    2
    3
>> [2,3,4] returns
    2    3    4
>> [2,3,4;5,8,7] returns
    2    3    4
    5    8    7
```

-*End Example*

Raise things to a power

If you want to raise a scalar number by a power, then you use "^"

If you want to raise an array (such as an matrix) by a power, then use ".^"

-*Example:*

```
>> 5^3 returns 125
>> [2,3].^2 returns 4    9
```

(raises every number in the matrix by a power of 2, $2^2=4$ and $3^2=9$)

-*End Example*

How to transpose a matrix

' represents transpose

-*Example:*

>> [1,2,3;4,5,6;7,8,9] returns
1 2 3
4 5 6
7 8 9
>> [1,2,3;4,5,6;7,8,9]' returns
1 4 7
2 5 8
3 6 9

-End Example

Definitions

abs computes absolute value or magnitude

-Examples

>> abs(5) returns 5
>> abs(-5) returns 5
>> abs([1,2;6,-5]) returns
1 2
6 5

-End Example

acos computes arccosine

-Example:

NOTE: the "pi" represents π
>> cos(pi/4) returns 0.7071
>> acos(0.7071) returns 0.7854 (which is (π/4) written as a decimal)

-End Example

all determines if all values are true

ans stores expression value

-Example:

"ans" stores the answer to the last operation you did.
>> 27*9
ans =
243
>> ans
ans =
243
>> 7-2
ans =
5
>> ans
ans =
5

-End Example

any determines if any values are true

asin computes arcsine

-Example:

NOTE: “pi” represents π

>> sin(pi/3) returns *0.8660*

>> asin(0.8660) returns *1.0471* (which is (π/3) written as a decimal)

-End Example

atan computes 2-quadrant arctangent

-Example:

NOTE: “pi” represents π

>> tan(pi/6) returns *0.5774*

>> atan(0.5774) returns *0.5236* (which is (π/6) written as a decimal)

-End Example

atan2 computes 4-quadrant arctangent

axis controls axis scaling

-Example:

The following was gotten by typing “**help axis**”

AXIS([XMIN XMAX YMIN YMAX]) sets scaling for the x- and y-axes on the current plot.

AXIS([XMIN XMAX YMIN YMAX ZMIN ZMAX]) sets the scaling for the x-, y- and z-axes on the current 3-D plot.

AXIS([XMIN XMAX YMIN YMAX ZMIN ZMAX CMIN CMAX]) sets the scaling for the x-, y-, z-axes and color scaling limits on the current axis (type “help CAXIS” for more info on “CAXIS”).

V = AXIS returns a row vector containing the scaling for the current plot. If the current view is 2-D, V has four components; if it is 3-D, V has six components.

AXIS AUTO returns the axis scaling to its default, automatic mode where, for each dimension, 'nice' limits are chosen based on the extents of all line, surface, patch, and image children.

AXIS MANUAL freezes the scaling at the current limits, so that if HOLD is turned on, subsequent plots will use the same limits.

AXIS TIGHT sets the axis limits to the range of the data.

AXIS FILL sets the axis limits and PlotBoxAspectRatio so that the axis fills the position rectangle. This option only has an effect if PlotBoxAspectRatioMode or DataAspectRatioMode are manual.

AXIS IJ puts MATLAB into its "matrix" axes mode. The coordinate system origin is at the upper left corner. The i axis is vertical and is numbered from top to bottom. The j axis is horizontal and is numbered from left to right.

AXIS XY puts MATLAB into its default "Cartesian" axes mode. The coordinate system origin is at the lower left corner. The x axis is horizontal and is numbered from left to right. The y axis is vertical and is numbered from bottom to top.

AXIS EQUAL sets the aspect ratio so that equal tick mark increments on the x-,y- and z-axis are equal in size. This makes SPHERE(25) look like a sphere, instead of an ellipsoid.

AXIS IMAGE is the same as AXIS EQUAL except that the plot box fits tightly around the data.
AXIS SQUARE makes the current axis box square in size.
AXIS NORMAL restores the current axis box to full size and removes any restrictions on the scaling of the units. This undoes the effects of AXIS SQUARE and AXIS EQUAL.
AXIS VIS3D freezes aspect ratio properties to enable rotation of 3-D objects and overrides stretch-to-fill.

AXIS OFF turns off all axis labeling, tick marks and background.
AXIS ON turns axis labeling, tick marks and background back on.

AXIS(H,...) changes the axes handles listed in vector H.

-*End Example*

bode computes magnitude and phase response
butter designs a Butterworth digital filter
c2d converts continuous state-space to discrete state-space
cd change current working directory
cd *DIRECTORY* sets the current directory to the one specified

-*Exanple*:

cd a:
sets the current directory to the **a** directory

-*End Example*

cd .. moves to the directory above the current one

-*Example:*

'ex' is the name of a folder which contains 1 file named 'Untitled.m'
>>cd a: (opens up the **a** directory)
>>cd (prints out what directory your currently in)
A:\
>>ls (lists the files in the directory)
Matlab Formulas List.doc
ex
>>cd ex (opens up the folder **ex**)
>> ls
Untitled.m
>> cd .. (closes the folder **ex** and returns to base of the **a** directory)
>> ls
Matlab Formulas List.doc
ex

-*End Example*

cd *BY ITSELF* prints out what directory your currently in (see "**cd..**" for an example)
ceil rounds to ∞
cheby1 designs a Chebyshev Type I digital filter
cheby2 designs a Chebyshev Type II digital filter
clc clears command screen

It will clear everything displayed on the screen so your staring at a blank screen, but it won't remove any variables or anything stored.

clear removes all variables from the workspace
(Deletes all the stored variables)

clear *(followed by a name)* clears only the variable specified

For example "clear p1" will only clear variable "p1".

NOTE: The wildcard character **'*'** can be used to clear variables that match a pattern

- *Example* **clear x*** clears all the variables in the workspace that start with **x**
- If **x** is global variable, then:
  - **clear x** removes **x** from the current workspace, but leaves it accessible to any functions declaring it global.
  - **clear global x** completely removes the global variable **x**

clf clears figures

clock represents the current time

collect collects coefficients of a symbolic expression

cos computes cosine of angle

-Example:

NOTE: the "pi" represents π

>> cos(pi/4) returns 0.7071

-End Example

cross computes the cross product of two vectors

-Example:

Let's do the cross product to the matrices:

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

>> cross([1,2;3,4;7,8],[2,4;5,6;2,5]) returns

```
 -29  -28
  12   22
  -1   -4
```

-End Example

cumprod determines cumulative products

Each term becomes the product of all the previous terms including itself.

-Example:

>> cumsum([0,1,2]) returns 0 0 0

>> cumsum([1,2,3]) returns 1 2 6

>> cumsum([1,2,3,4,5,6]) returns 1 3 6 10 15 21

-End Example

cumsum determines cumulative sums

Each term becomes the sum of all the previous terms including itself.

-Example:

>> cumsum([0,1,2]) returns 0 1 3

>> cumsum([1,2,3]) returns 1 3 6

>> cumsum([1,2,3,4,5,6]) returns 1 2 6 24 120 720

-End Example

date prints out current date
demo runs demonstrations
det computes the determinant of a matrix
-Example:
Let's find the determinate of the matrix
7 5
3 4
>> det([7,5;3,4]) returns 13
-End Example
diff computes the differences between adjacent values; differentiates a symbolic expression
dir lists the files in the current directory (same as the command **ls**)
(see "**cd ..**" for an example, look for the command "**ls**")
disp displays matrix or text
-Example:
>> x = [1,2;3,4]
x =
1 2
3 4
>> disp(x)
1 2
3 4
>> disp([1,2;3,4])
1 2
3 4
-End Example
dot computes the dot product of two vectors
-Example:
Let's do the dot product to the matrices:
1 2 and 2 4
3 4 5 6
>> dot([1,2;3,4],[2,4;5,6]) returns 17 32
-End Example
dsolve solves an ordinary differential equation
-Example:
>> dsolve('Dy+y=0') returns C1*exp(-t)
>> dsolve('D2y = -a^2*y', 'y(0) = 1, Dy(pi/a) = 0') returns cos(a*t)
-End Example
eig computes the eigenvalues and eigenvectors of a matrix
ellip designs an elliptic digital filter
else optional clause in **if** structure
elseif optional clause in **if** structure
end defines end of a control structure
eps represents floating-point precision
exit terminates Matlab (same as the command **quit**)

exp computes value with base ***e***

-Example:

Generally on calculators or in books, you would represent this as e^x, where "x" is some number. Let's do e^2.

>> exp(2) returns 7.3891

-End Example

expand expands a symbolic expression

-Example:

NOTE: you need to do ">> syms x" (where "x" is the variable your using in the **expand** call) before you can do this command

>> syms x

>>expand((x+1)^3) returns x^3+3*x^2+3*x+1

>>expand(sin(x+y)) returns sin(x)*cos(y)+cos(x)*sin(y)

>>expand(exp(x+y)) returns exp(x)*exp(y)

-End Example

eye generates identity matrix

ezplot generates a plot of a symbolic expression

factor can do two things:

1) Factor polynomial equations
2) generates the prime factors

-Example:

factor(polynomial) factors the polynomial

NOTE: you need to do ">> syms x" (where "x" is the variable your using in the **factor** call) before you can do this command

>> syms x

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

>> factor (x^2-1) returns (x-1)*(x+1)

factor(N) returns a vector containing the prime factors of N.

>> factor(5) returns 5

>> factor(9) returns 3 3

>> factor(125) returns 5 5 5

>> factor(126) returns 2 3 3 7

-End Example

fft computes the frequency content of a signal

filter applies a digital filter to an input signal

find locates nonzero values

finite determines if values are finite

-Example:

Returns "1" for true and "0" for false.

NOTE: for this example, ">>x=Inf (Inf represents ∞)"

>>finite(52) returns 1

>>finite (x) returns 0

-End Example

fix rounds towards zero

floor rounds towards -∞

for generates loop structure
format 1 sets format to plus and minus signs only
format compact sets format to compact form
format long sets format to long decimal
format long e sets format to long exponential
format loose sets format to non-compact form
format short sets format to short decimal
format short e sets format to short exponential
fprintf prints formatted information
freqs computes the analog frequency content
freqz computes the digital frequency content
function generates user-defined function
grid inserts gird in plot
(see **plot** for an example)
grpdelay measure the group delay of a digital filter
help invokes help facility
help *COMMAND* invokes help facility for the command specified
- *Example* **help abs** invokes help facility for the command **abs**
hist plots histogram
hold toggles the hold state
hold on holds the current plot and all axis properties so that subsequent graphing commands add to the existing graph
hold off returns to the default mode whereby **plot** commands erase the previous plots and reset all axis properties before drawing new plots
horner converts a symbolic expression into a nested form
i represents the value sqrt(-1) (where sqrt = squrare root)
if tests logical expression
Inf represents the value ∞
input accepts input from keyboard
int integrates a symbolic expression
-Example:

NOTE: you need to do ">> syms x x1 alpha u t" (where "x", "x1", "alpha", "u", and "t" are the variables your using in the **int** call) before you can do this command

```
>> syms x x1 alpha u t;
>> A = [cos(x*t),sin(x*t);-sin(x*t),cos(x*t)];
>> int(x) returns 1/2*x^2
>> int(x+2) returns 1/2*x^2+2*x
>> int(1/(1+x^2)) returns atan(x)
>> int(sin(alpha*u),alpha) returns -cos(alpha*u)/u
>> int(x1*log(1+x1),0,1) returns 1/4
>> int(4*x*t,x,2,sin(t)) returns    2*sin(t)^2*t-8*t
```

-End Example
interpl computes linear and cubic spline interpolation
inv computes the inverse of a matrix

-Example:

NOTE: in order to do an inverse you to have an n x n matrix.

Let's do the inverse of this matrix

```
5    4    7
7    8    4
2    4    3
```

```
>> inv([5,4,7;7,8,4;2,4,3])
ans =
   0.1111    0.2222   -0.5556
  -0.1806    0.0139    0.4028
   0.1667   -0.1667    0.1667
```

-End Example

isempty determines if matrix is empty

isnan determines if values are **NaN**s (**NaN**: Not-a-Number)

j represents the value sqrt(-1) (where sqrt = squrare root)

length determines number of values in a vector

-Example:

```
>> length([1,2]) returns 2
>> length([1;2]) returns 2
>> length([1,2,3]) returns 3
>> length([1;2;3]) returns 3
>> length([5,4,7;7,8,4;2,4,3]) returns 3
```

-End Example

linspace generates a linearly spaced vector

linspace(X1, X2) generates a row vector of 100 linearly equally spaced points between X1 and X2.

linspace(X1, X2, N) generates N points between X1 and X2. For N < 2, **linspace** returns X2.

load loads matrices from a file

log computes natural logarithm (**ln** on most calculators)

-Example:

```
>> log(0.55) returns -0.5978
```

-End Example

log10 computes common logarithm (**log** on most calculators)

-Example:

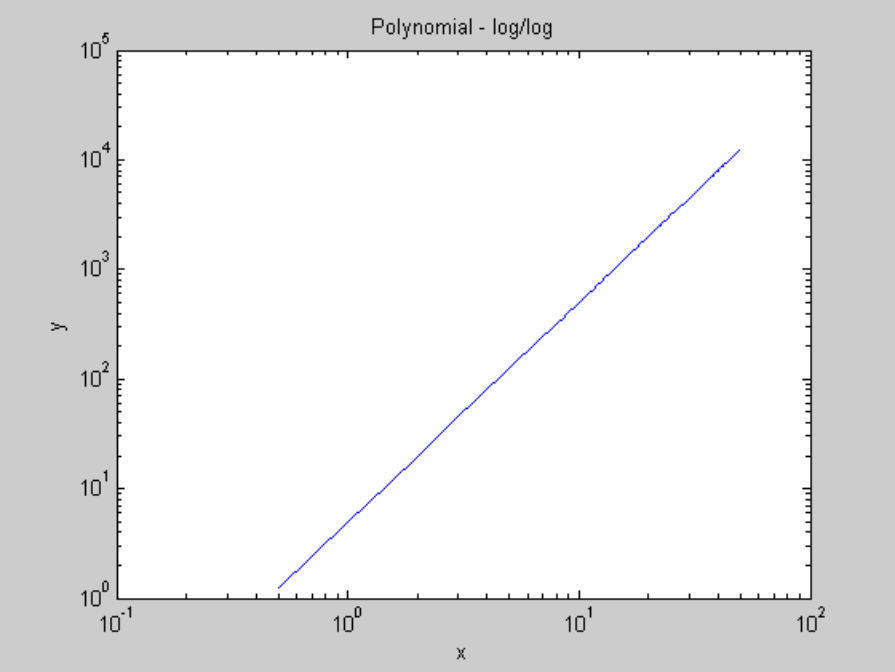
```
>> log10(0.55) returns -0.2596
```

-End Example

loglog generates a log-log plot

-Example:

```
>> x=0:0.5:50;
>> y=5*x.^2;
>> loglog(x,y)
>> title('Polynomial - log/log'),xlabel('x'),ylabel('y')
```



-End Example

logspace generates a logarithmically spaced vector

logspace(X1, X2) generates a row vector of 50 logarithmically equally spaced points between decades 10^X1 and 10^X2. If X2 is pi, then the points are between 10^X1 and pi.

logspace(X1, X2, N) generates N points. For N < 2, LOGSPACE returns 10^X2.

ls lists the files in the current directory (same as the command **dir**)
(see "**cd ..**" for an example)

lu computes the LU factorization of a matrix

max determines maximum value

-Example:

>> max([1,2,5]) returns 5

>> max([2,5,6;7,8,0]) returns 7 8 6

-End Example

mean determines mean value

-Example:

>> mean([5,2,7,8,6,10,25,69]) returns 7.5000

-End Example

median determines median value

-Example:

>> median([5,2,7,8,6,10,25,69]) returns 16.5000

-End Example

min determine minimum value

-Example:

>> min([1,2,5]) returns 1

>> min([2,5,6;7,8,0]) returns 2 5 0

-End Example

NaN represents the value Not-a-Number
numden returns the numerator and denominator expressions
numeric converts a symbolic expression to a number
nyquist computes the Nyquist frequency response
ode23 computes a second / third - order Runger-Kutta solution
When to use: For problems with crude error tolerances or for solving moderately stiff problems. It may be more efficient than ode45 at crude tolerances and in the presence of moderate stiffness.
To see an example, look at **ode45** it is set up the same way. You call both **ode23** and **ode45** the same way.
For examples of both **ode23** and **ode45** see Prof. White's Stuff at:
http://www.profjrwhite.com/math_methods/maltab_demos.htm
ode45 computes a fourth / fifth - order Runger-Kutta solution
When to use: Most of the time. This should be the first solver you try.
Let's create an example.
-Example:

First create a file called **rigid.m** that contains the following lines of code:

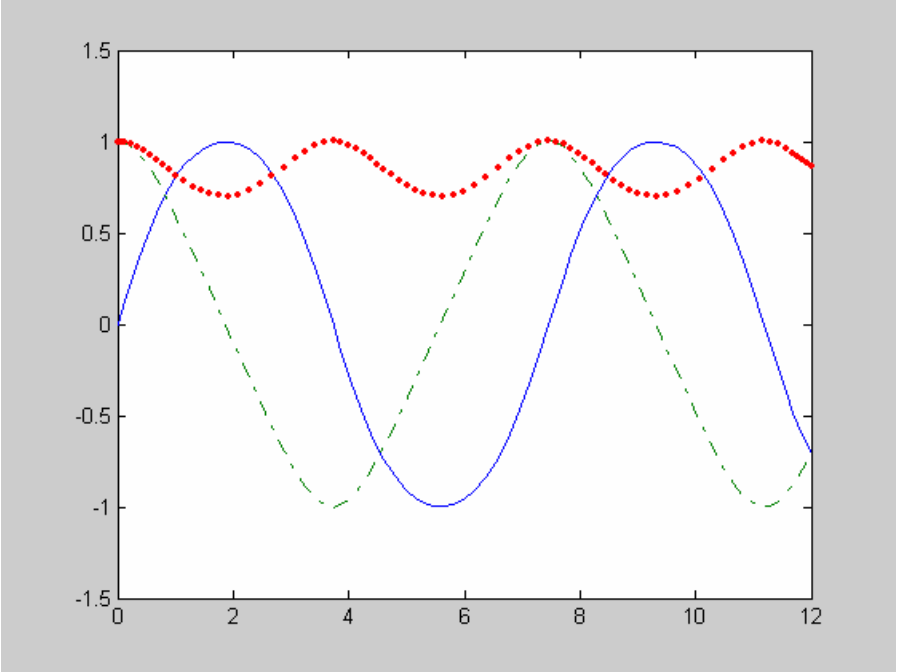
```
function dy = rigid(t,y)
dy = zeros(3,1); % a column vector
dy(1) = y(2) * y(3);
dy(2) = -y(1) * y(3);
dy(3) = -0.51 * y(1) * y(2);
```

Second in the matlab window, make sure your in the right directory, look at the command **cd** to see what directory your in and then use **ls** to list the files in that directory)

```
>> cd
D:\matlab
>> ls
.                rigid.m
```

Third in the matlab window type the following lines of code to execute the ode45.

```
>> options = odeset('RelTol',1e-4,'AbsTol',[1e-4 1e-4 1e-5]);
>> [t,y] = ode45('rigid',[0 12],[0 1 1],options);
>> plot(t,y(:,1),'-',t,y(:,2),'-.',t,y(:,3),'.')
```



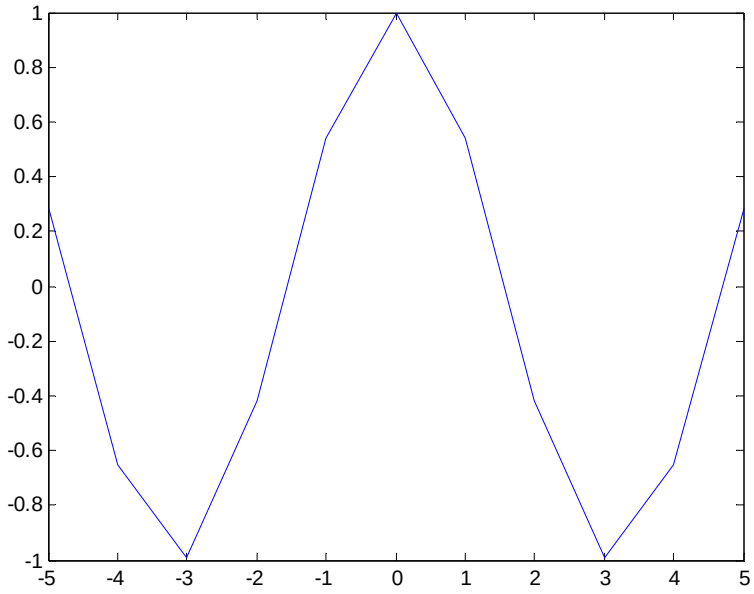
-End Example

ones	generates matrix of ones
pause	temporarily halts a program
pi	represents the value π
plot	generates a linear xy plot

-Example:

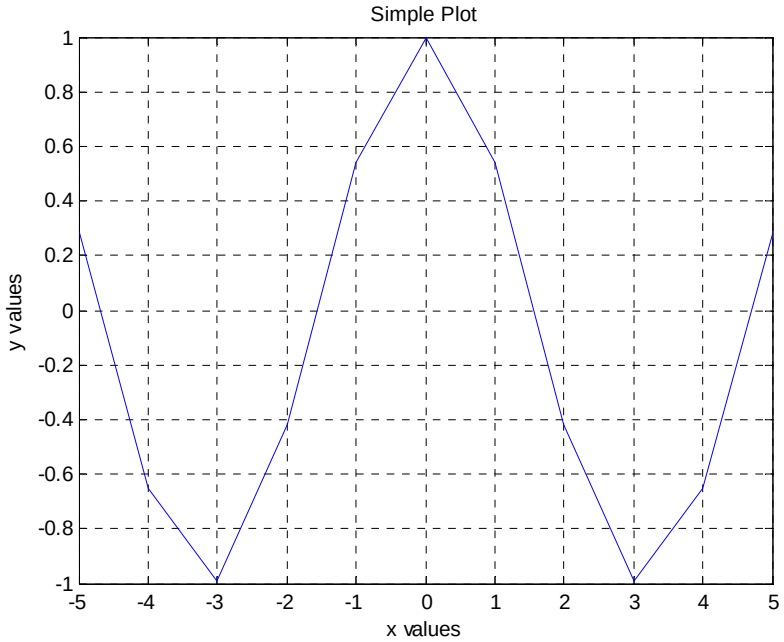
Here is an example to create a simple plot.
First let's create two variables "x" and "y".

```
>> x=-5.0:5
x =
   -5    -4    -3    -2    -1     0     1     2     3     4     5
>> y=cos(x)
y =
  Columns 1 through 7
    0.2837   -0.6536   -0.9900   -0.4161    0.5403    1.0000    0.5403
  Columns 8 through 11
   -0.4161   -0.9900   -0.6536    0.2837
>> plot(x,y)
```



Now let's make some labels on the figure and insert grids in the plot.

```
>>grid
>> title('Simple Plot')
>> xlabel('x values')
>> ylabel('y values')
```



-End Example

NOTE: (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	blue	.	point	-	solid

g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
		v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

This was gotten by typing "**help plot**"

poly2sym converts a vector to a symbolic polynomial
polyfit computes a least-squares polynomial
polyval evaluates a polynomial'
pretty prints a symbolic expression in typeset form
print print the graphics window
prod determines product of values
qr computes the QR factorization of a matrix
quad computes the integral under a curve (Simpson)
quad8 computes the integral under ac curve (Newton-Cote)
quit terminates Matlab (same as the command **exit**)
rand generates a uniform random number
randn generates a Gaussian random number
rem computes remainder from division

-*Example:*

NOTE: rem(x,y) = the remainder of "x / y"
>> rem(6,3) returns 0
>> rem(4,3) returns 1
>> rem(5,3) returns 2

-*End Example*

remez designs an optimal FIR digital filter
residue performs a partial-fraction expansion
rlocus computes the root locus
round rounds to the nearest integer

-*Example:*

>> round(2.3) returns 2
>> round(2.5) returns 3
>> round(-2.3) returns -2
>> round(-2.5) returns -3

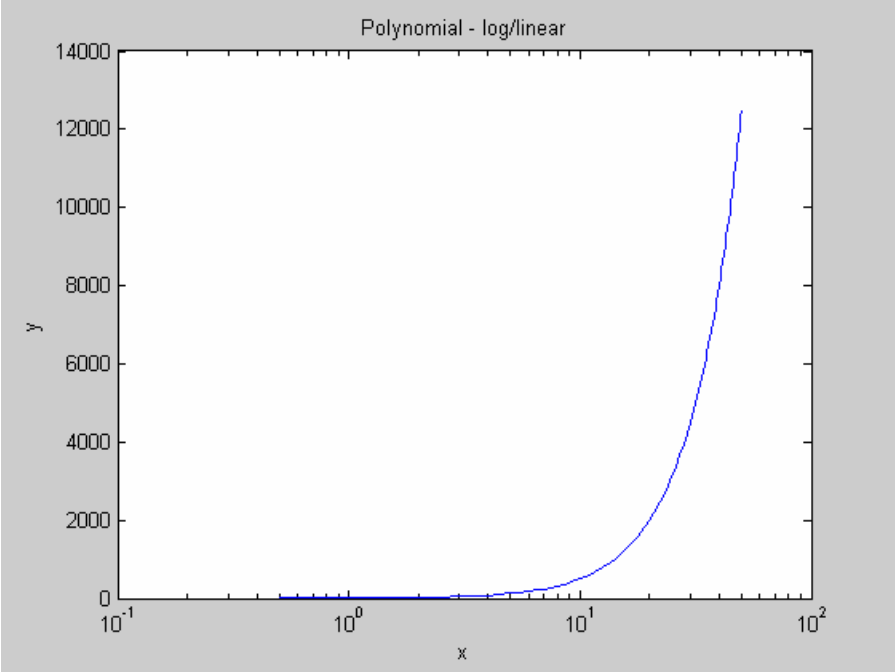
-*End Example*

save saves variables in a file
semilogx generates a log-linear plot

-*Example:*

>> x=0:0.5:50;

```
>> y=5*x.^2;
>> semilogx(x,y)
>> title('Polynomial - log/linear'),xlabel('x'),ylabel('y')
```

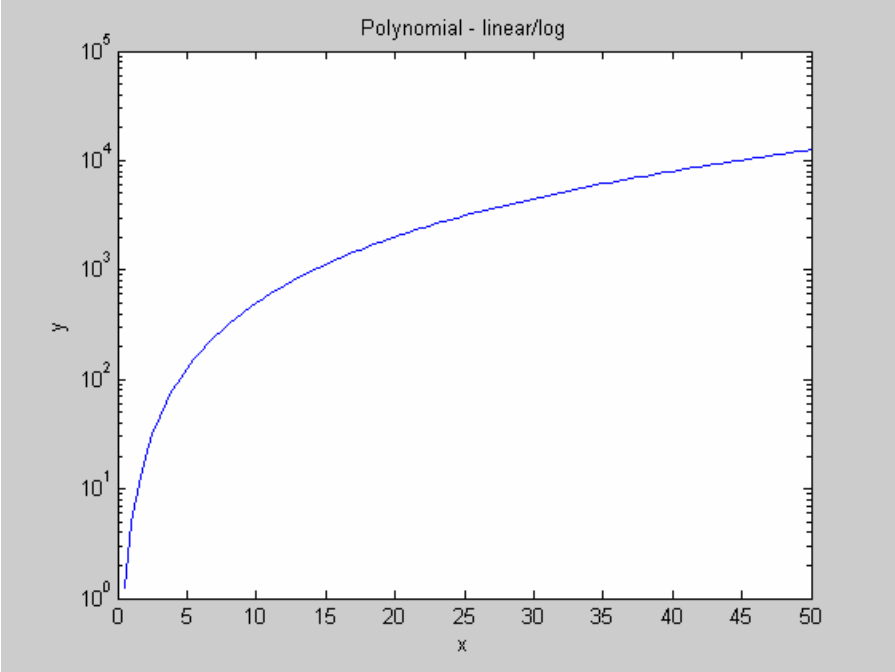


-End Example

semilogy generates a linear-log plot

-Example:

```
>> x=0:0.5:50;
>> y=5*x.^2;
>> semilogy(x,y)
>> title('Polynomial - linear/log'),xlabel('x'),ylabel('y')
```



-End Example

sign generates –1,0,1 based of sign

-Example:

>> sign(0) returns 0
>> sign(5) returns 1
>> sign(-5) returns -1

-End Example

simple shortens a symbolic expression

-Example:

NOTE: you need to do ">> syms x" (where "x" is the variable your using in the **simple** call) before you can do this command
>> syms x
>> simple(cos(x)^2+sin(x)^2) returns 1
>> simple(2*cos(x)^2-sin(x)^2) returns 3*cos(x)^2-1
>> simple(cos(x)^2-sin(x)^2) returns cos(2*x)
>> simple(cos(x)+(-sin(x)^2)^(1/2)) returns cos(x)+i*sin(x)
>> simple(cos(x)+i*sin(x)) returns exp(i*x)
>> simple((x+1)*x*(x-1)) returns x^3-x
>> simple(x^3+3*x^2+3*x+1) returns (x+1)^3
>> simple(cos(3*acos(x))) returns 4*x^3-3*x

-End Example

simplify simplifies a symbolic expression

-Example:

NOTE: you need to do ">> syms x c alpha beta" (where "x", "c", "alpha", and "beta" is the variable your using in the **simplify** call) before you can do this command
>> syms x c alpha beta
>> simplify(sin(x)^2 + cos(x)^2) returns 1
>> simplify(exp(c*log(sqrt(alpha+beta)))) returns (alpha+beta)^(1/2*c)

-End Example

sin computes sine of angle

-Example:

NOTE: the "pi" represents π
>> sin(pi/3) returns 0.8660
>> sin(pi/5) returns 0.5878

End Example

size prints row and column dimensions

-Example:

>> size([1,2]) returns 1 2
>> size([1;2]) returns 2 1
>> size([1,2;2,4]) returns 2 2
>> size([1,2,3;2,4,7]) returns 2 3

-End Example

solve solves an equation

-Example:

NOTE: in these examples, what is being returned is what "x" equals.

>> solve('x+1=0') returns -1
>> solve('x^2-1=0') returns
[1]
[-1]
>> solve('x^2+5*x+7=3') returns
[-4]
[-1]
You can also use solve to solve multiple equations with multiple unknowns.
>> [x,y] = solve('x^2 + x*y + y = 3','x^2 - 4*x + 3 = 0') returns
x =
[1]
[3]
y =
[1]
[-3/2]

-End Example

sort sorts values

-Example:

>> sort([5,2,3]) returns 2 3 5
>> sort([5,2,3;4,7,8]) returns
4 2 3
5 7 8

-End Example

sqrt computes square root

-Example:

>> sqrt(4) returns 2
>> sqrt(2) returns 1.4142

-End Example

ss2tf converts state-space to transfer function

ss2zp converts state-space to zero-pole-gain

std computes standard deviation

-Example:

>> std([5,2,7,8,6,10,25,69]) returns *22.3159*

-End Example

step computes the unit step response

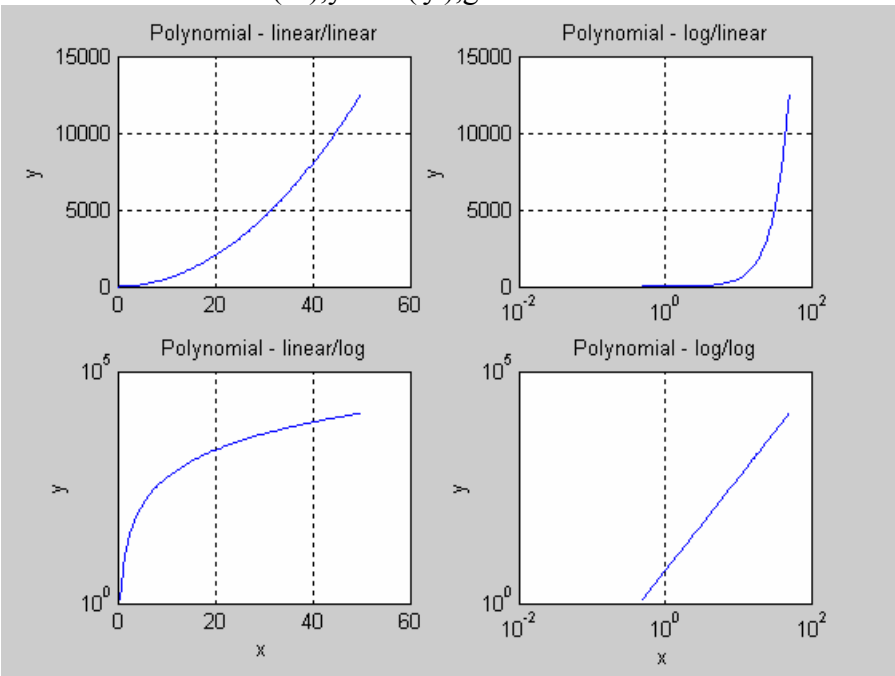
subplot splits graphics window into subwindows

Subplot is a very useful command. Say you have a value for x and y, and you want to graph it as a linear, log, or combination of the two. Using the subplot command you can do it all in one figure, allowing you to compare the graphs.

-Example:

```
>> x=0:0.5:50;
>> y=5*x.^2;
>> subplot(2,2,1),plot(x,y),...
        title('Polynomial - linear/linear'),...
        ylabel('y'),grid,...
    subplot(2,2,2),semilogx(x,y),...
```

```
    title('Polynomial - log/linear'),...
    ylabel('y'),grid,...
subplot(2,2,3),semilogy(x,y),...
    title('Polynomial - linear/log'),...
    xlabel('x'),ylabel('y'),grid,...
subplot(2,2,4),loglog(x,y),...
    title('Polynomial - log/log'),...
    xlabel('x'),ylabel('y'),grid
```



-End Example

sum	determines sum of values
svd	converts the SVD factorization of a matrix
sym2poly	converts a symbolic expression to a coefficient vector
symadd	adds two symbolic expressions
symdiv	divides two symbolic expressions
symmul	multiplies two symbolic expressions
sympow	raises a symbolic expression to a power
symsub	subtracts two symbolic expressions
symvar	returns independent variable
tan	computes tangent of angle

-Example:

NOTE: the "pi" represents π
>> tan(pi/5) returns 0.7265
>> tan(pi/23) returns 0.1374

-End Example

tf2ss	converts transfer function to state-space
tf2zp	converts transfer function to zero-pole-gain
title	adds title to a plot (see **plot** for an example)

unwrap	removes 2π discontinuities in a phrase angle
what	list variables
while	generates a loop structure
who	lists variables in memory (lists all the variables you made)
whos	lists variables in memory plus sizes
xlabel	adds x-axis label to a plot (see **plot** for an example)
ylabel	adds y-axis label to a plot (see **plot** for an example)
yulewalk	designs an optimal IIR digital filter
zeros	generates matrix of zeros
zp2ss	converts zero-pole-gain to state-space
zp2tf	converts zero-pole-gain to transfer function