ECE 201 – Matlab Lesson #2
Basic Arithmetic and Plotting

Element-by-Element Arithmetic for Vectors and Matrices

Syntax

```
A+B
A-B
A.*B
A./B
A.\B
A.^B
A.'
```

Description

MATLAB has two different types of arithmetic operations. Matrix arithmetic operations are defined by the rules of linear algebra. Array arithmetic operations are carried out element-by-element. The period character (.) distinguishes the array operations from the matrix operations. However, since the matrix and array operations are the same for addition and subtraction, the character pairs `.+` and `.-` are not used.

`+`	Addition or unary plus. `A+B` adds `A` and `B`. `A` and `B` must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.
`-`	Subtraction or unary minus. `A-B` subtracts `B` from `A`. `A` and `B` must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.
`.*`	Array multiplication. `A.*B` is the element-by-element product of the arrays `A` and `B`. `A` and `B` must have the same size, unless one of them is a scalar.
`./`	Array right division. `A./B` is the matrix with elements `A(i,j)/B(i,j)`. `A` and `B` must have the same size, unless one of them is a scalar.
`.\`	Array left division. `A.\B` is the matrix with elements `B(i,j)/A(i,j)` . `A` and `B` must have the same size, unless one of them is a scalar.
`.^`	Array power. `A.^B` is the matrix with elements `A(i,j)` to the `B(i,j)` power. `A` and `B` must have the same size, unless one of them is a scalar.
`.'`	Array transpose. `A.'` is the array transpose of `A`. For complex matrices, this does not involve conjugation.

Remarks

The arithmetic operators have M-file function equivalents, as shown:

Binary addition	`A+B`	`plus(A,B)`
Binary subtraction	`A-B`	`minus(A,B)`
Array-wise multiplication	`A.*B`	`times(A,B)`
Array-wise right division	`A./B`	`rdivide(A,B)`

Written by Eddie Mayhew - Consultant for The MathWorks, Inc. and GMU Adjunct Faculty member.
Updated by Dr. Paris

Array-wise left division	`A.\B`	`ldivide(A,B)`
Array-wise power	`A.^B`	`power(A,B)`

Examples

Array Operations	
x = [1 2 3]	y = [4 5 6]
y'	4 5 6
x-y	-3 -3 -3
x-2	-1 0 1
x.*y	4 10 18
x.*2	2 4 6
x.\y	4 5/2 2
2./x	2 1 2/3
x./y	1/4 2/5 ½
x./2	1/2 1 3/2
x.^y	1 32 729
x.^2	1 4 9

Built-in Operations on Arrays and Matrices

Note: all functions may be called with complex-valued arguments.

☻ sum

Sum of array elements

Syntax

B = sum(A)

Description

If A is a vector, `sum(A)` returns the sum of the elements.
If A is a matrix, `sum(A)` treats the columns of A as vectors, returning a row vector of the sums of each column.

Examples

The magic square of order 3 is:

```
>> M = magic(3)
```

which returns

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

This is called a magic square because the sums of the elements in each column are the same.

```
>> sum(M)
```

returns

```
ans =
    15    15    15
```

as are the sums of the elements in each row, obtained by transposing:

```
>> sum(M')
```

returns

```
ans =
    15    15    15
```

power

.^ Array power.

`Z = X.^Y` denotes element-by-element powers. X and Y must have the same dimensions unless one is a scalar. A scalar can operate into anything.

`C = POWER(A,B)` is called for the syntax '`A .^ B`'.

Example
```
>> E = [2 3 4 5];
>> F = [3 2 1 0];
>> G = power(E,F)     % Does G return the same result as E.^F?
```

pow2
Base 2 power

Syntax
```
      X = pow2(Y)
```

Description
`X = pow2(Y)` returns an array `X` whose elements are 2 raised to the power `Y`.

Examples
```
>> X = [0 1 2 3 4 5 6];
>> Y = pow2(X)       % Of course this must be the same length as X, right?
>> Z = 0.0004;
>> pow2(Z)           % Which is equal to 2^Z.
>> Z = -Z;
>> pow2(Z)           % Which is equal to 2^Z.
```

☻ **sqrt**
Square root

Syntax
```
      B = sqrt(A)
```

Description
`B = sqrt(A)` returns the square root of each element of the array `X`.

Examples
```
>> A = [2 4 9 16];
>> B = sqrt(A)
>> C = A.^ 0.5     % Does B = C?
```

☻ **exp**
Exponential

Syntax
```
   Y = exp(X)
```

Description
The `exp` function is an elementary function that operates element-wise on arrays.
`Y = exp(X)` returns the exponential for each element of `X`.

Examples
```
>> W = 0:0.01:5;
>> X = exp(W);
>> Y = -exp(W);
>> plot(W,X,'c')
>> hold on
>> plot(W,Y,'-b')
% Please do not clear any of these variables from your workspace.
% They will be needed in future examples.
```

log
Natural logarithm

Syntax
```
Y = log(X)
```

Description
The `log` function operates element-wise on arrays. Its domain includes complex and negative numbers, which may lead to unexpected results if used unintentionally.
`Y = log(X)` returns the natural logarithm of the elements of X.

Examples
```
>> hold off
>> L = log(W);          % What does L(1) equal?
>> plot(W,L)            % Where does the value of L(1) show up on the plot?
>> zoom
% Zoom in on 0 (zero) on the X-axis.  Notice how the X-axis is avoided.
```

log10
Common (base 10) logarithm

Syntax
```
Y =  log10(X)
```

Description
The `log10` function operates element-by-element on arrays. Its domain includes complex numbers, which may lead to unexpected results if used unintentionally.
`Y = log10(X)` returns the base 10 logarithm of the elements of `X`.

Examples

```
>> log10(1000000)
>> log10(1e100)             % 1e100 is 1 followed by 100 zeros.
>> log10(0.000001)
```

log2

Base 2 logarithm and dissect floating-point numbers into exponent and mantissa.

Syntax

```
 Y = log2(X)
```

Description

`Y = log2(X)` computes the base 2 logarithm of the elements of `X`.

Using $2^{14} = 16{,}384$. If X = 16384 then `Y = log2(X) = 14.`

Examples

```
>> X = [1 2 4 8 16 32 64 101];
>> Y = log2(X)
```

☻ **pi**

Ratio of a circle's circumference to its diameter.

Syntax

```
     pi
```

Description

`pi` returns the floating-point number nearest the value of (circumference/diameter).

Examples

The expression `sin(pi)` is not exactly zero because "`pi`" is not exactly 3.1415...:

```
>> sin(pi)
```

returns

```
        ans =

        1.2246e-16
```

☻ **i,j**

imaginary unity- square root of -1.

Syntax

```
i
j
```

Description

`i` or `j` returns the complex unity, i.e., the square root of -1.

Examples

The expression `exp(j*pi/2)` equals 0+j:

```
>> exp(j*pi/2)
```

returns

```
ans =

   0.0000 + 1.0000i
```

Note: be very careful when using i or j as variables; this is best avoided!

☻ sin

Sine

Syntax

```
Y = sin(X)
```

Description

The `sin` command operates on an element-by-element basis. All angles are in radians.
`Y = sin(X)` returns the circular sine of the elements of `X`.

Examples

Graph the sine function over the domain

```
>> t = linspace(0,1,1000);    % Represents Time
>> A = 2;                     % Represents Amplitude
>> f = 5;                     % Represents Frequency
>> theta = pi/4;              % Represents Phase
>> SineWave = A * sin(2*pi*f*t + theta);
>> plot(t,SineWave)
```

☻ cos

Cosine

Syntax

```
Y = cos(X)
```

Description

The `cos` function operates element-wise on arrays. All angles are in radians.
`Y = cos(X)` returns the circular cosine for each element of `X`.

Examples

Graph the cosine function over the domain:

```
>> x = -pi:0.01:pi;
>> plot(x,cos(x),'r');
```

Now try and expand the range to two full cycles. Hint: Double the width of x.

tan

Tangent

Syntax

```
     Y = tan(X)
```

Description

The `tan` command operates element-wise on arrays. All angles are in radians.
`Y = tan(X)` returns the circular tangent of each element of `X`.

Examples

Graph the tangent function over the domain:

```
>> x = (-pi/2)+0.01:0.01:(pi/2)-0.01;
>> plot(x,tan(x))
```

% Notice how evaluating `tan(pi/2)` is being avoided.

Plotting

☻ linspace

Generate linearly spaced vectors

Syntax:

```
     y = linspace(a,b)
     y = linspace(a,b,n)
```

Description:

The linspace function generates linearly spaced vectors. It is similar to the colon operator ":", but gives direct control over the number of points.

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

`y = linspace(a,b,n)` generates n points.

Remarks:

All the arguments to `linspace` must be scalars.

Examples:

```
>> W = linspace(4,8);
>> length(W)

ans =

   100

>> W = linspace(1,45,7)

W = 1.0000   8.3333   15.6667   23.0000    30.3333    37.6667    45.0000
```

☻ logspace

Generate logarithmically spaced vectors

Syntax:

```
     y = logspace(a,b)
     y = logspace(a,b,n)
     y = logspace(a,pi)
```

Description:

The logspace function generates logarithmically spaced vectors. This means that the ratio of two consecutive elements is the same! Especially useful for creating frequency vectors, it is a logarithmic equivalent of linspace and the ":" or colon operator.

`y = logspace(a,b)` generates a row vector y of 50 logarithmically spaced points between decades 10^a and 10^b.

`y = logspace(a,b,n)` generates n points between decades 10^a and 10^b.

`y = logspace(a,pi)` generates the points between 10^a and pi, which is useful for digital signal processing where frequencies over this interval go around the unit circle.

Remarks:
All the arguments to `logspace` must be scalars.

Examples:

```
>> Y = logspace(0,5);
>> length(Y)

ans =

    50

>> Y = logspace(0,1,7)

Y =

  1.0000  1.4678   2.1544   3.1623   4.6416   6.8129  10.0000

>> Y = logspace(0,2,7)

Y =

   1.0000   2.1544   4.6416  10.0000   21.5443   46.4159  100.0000
```

☻ plot

Linear 2-D plot

Syntax

```
  plot(Y)
  plot(X,Y)
  plot(X,Y,LineSpec,...)
```

Description
`plot(Y)` plots the columns of `Y` versus their index if `Y` is a real number.
`plot(X,Y1,X,Y2)` plots all lines defined by `Xn` versus `Yn` pairs. If only `Xn` or `Yn` is a matrix, the vector is plotted versus the rows or columns of the matrix, depending on whether the vector's row or column dimension matches the matrix.

`plot(X1,Y1,LineSpec,...)` plots all lines defined by the `Xn,Yn,LineSpec` triples, where `LineSpec` is a line specification that determines line type, marker symbol, and color of the plotted lines.

Examples

```
>> X = 0:100;
>> Y1 = X +7.9;
>> Y2 = (X * 4) - Y1;          % Notice how X, Y1, and Y2 are the same size.
                               % MATLAB will produce an error if they are not.
```

```
>> plot(X,Y1)
```

plots X (the independent variable) along the X-Axis and plots Y1 (the dependent variable) along the Y-Axis with a solid blue line. A solid line is the default line specification and blue is the default color.

```
>> plot(X,Y1,X,Y2)
```

plots `X` vs. `Y1` as well as X vs. Y2 all on the same plot.

% Please do not close the plot that you have just generated. We will add on to it later.

☻ hold

Hold current graph in the Figure

Syntax

```
hold on
hold off
hold
```

Description

The `hold` function determines whether new graphics objects are added to the graph or replace objects in the graph.

`hold on` retains the current plot and certain Axes properties so that subsequent graphing commands add to the existing graph.

`hold off` resets Axes properties to their defaults before drawing new plots.

`hold off` is the default.

`hold` toggles the hold state between adding to the graph and replacing the graph.

Examples

```
>> hold on
>>  t = 0:pi/20:30*pi;
>> plot(t,50*sin(t),'green')   % Notice that the original data was not overwritten.
```

% Please do not close the plot that you have just generated. We will add on to it later.

```
>> hold off    % Turn off hold now - we won't need it for the next example.
```

☻ LineSpec

Line specification syntax

Description

A line specification is an argument to plotting functions, such as `plot`, that defines three components used to specify lines in MATLAB:

Line style
Color
Marker symbol

For example,

```
plot(x,y,'-.ro')
```

plots `y` versus `x` using a dash-dot line (`-.`), colored red (`r`), and places circular markers (`o`) at the data points. If you specify a marker, but no a linesytle, MATLAB plots only the markers.

The following tables list the symbols you use to define the line style, color, and marker. Specify these symbols (in any order) as a quoted string after the data arguments.

Symbol	**Line Style**	**Symbol**	**Line Style**
`-`	solid line (default)	`:`	dotted line
`--`	dashed line	`-.`	dash-dot line

Symbol	**Color**	**Symbol**	**Color**
`y`	yellow	`g`	green
`m`	magenta	`b`	blue
`c`	cyan	`w`	white
`r`	red	`k`	black

Marker Specifier	**Description**
`+`	plus sign
`o`	circle
`*`	asterisk
`.`	point
`x`	cross
`s`	square

d	diamond
^	upward pointing triangle
v	downward pointing triangle
>	right pointing triangle
<	left pointing triangle
p	pentagram
h	hexagram

Examples

Specifying Line Styles

Line styles and markers allow you to discriminate different data sets on the same plot when color is not available. For example:

```
>> plot(X,Y1,'-k*',X,Y2,'-.ko')
```

creates a graph of `X,Y1 and X,Y2` using solid and dash-dot lines as well as different marker symbols. Notice that the old data was overwritten due to the `hold off` above.

☻ figure

Create a Figure graphics object

Syntax

```
figure
figure(n)
```

Description

`figure` creates Figure graphics objects. Figure objects are the individual windows on the screen in which MATLAB displays graphical output. The default figure title will be "Figure n" with 'n' being the next Figure number displayed.

`figure(n)` creates a new Figure object using default property values. The figure generated will be given a default title of "Figure n" .

Examples

```
>> figure(2)
>> figure(3)
>> figure      % Notice that this Figure is called Figure 4. Calling a Figure up this way
               % will automatically generate the next one in the sequence.
```

clf

Clear current Figure window

Syntax

```
clf
```

```
clf reset
```

Description

`clf` deletes all data from the current Figure. Note - the window is not deleted, just the contents inside the window.

Example:

% Click on Figure 1 to bring it forward. This will make it the "Current Figure".

```
>> clf             % Notice that the data went away.
```

close

Delete specified Figure.

Syntax

```
close
close(h)
close name
close all
```

Description

`close` deletes the current Figure or the specified Figure(s). It optionally returns the status of the close operation.

`close(h)` deletes the Figure identified by h. If h is a vector or matrix, `close` deletes all Figures identified by h.

`close all` deletes all Figures.

Examples:

```
>> close(1)
>> close(3)
>> close all
```

☻ subplot

Create and control multiple Axes

Syntax

```
subplot(m,n,p)
```

Description

`subplot` divides the current Figure into rectangular panes that are numbered row-wise. Each pane contains an Axes. Subsequent plots are output to the current pane.

`subplot(m,n,p)` creates an Axes in the `p`-th pane of a Figure divided into an `m`-by-`n` matrix of rectangular panes. The new Axes becomes the current Axes.

Remarks

You can omit the parentheses and specify subplot as:

```
   subplot mnp
```

where `m` refers to the row, `n` refers to the column, and `p` specifies the pane.

Examples

```
>> figure(1)
>> Linear = 1:100;
>> Spike = 0.01*ones(1,100);
>> Spike(1:2:end) = -0.01;            % Notice how this was done.
>> Spike(50) = 1;
>> subplot(2,1,1); plot(Linear)
>> subplot(2,1,2); plot(Spike)
>> figure(2)
>> subplot(3,2,1); plot(Linear)
>> subplot(3,2,2); plot(Spike)
>> subplot(3,2,3); plot(Linear)
>> subplot(3,2,4); plot(Spike)
>> subplot(3,2,5); plot(Linear)
>> subplot(3,2,6); plot(Spike)
>> figure(3)
>> subplot(3,2,1); plot(Linear)
>> subplot(3,2,2); plot(Spike)
>> subplot(3,2,3.5); plot(Linear)
>> subplot(3,2,5); plot(Linear)
>> subplot(3,2,6); plot(Spike)
```

line

Create a line

Syntax

```
   line(X,Y)
   line('PropertyName',PropertyValue,...)
```

Description

`line` creates a Line object in the current Axes. You can specify the color, width, line style, and marker type, as well as other characteristics. `line(X,Y)` adds the Line defined in vectors `X` and `Y` to the current Axes.

Examples

This example uses the line function to add a shadow to plotted data. First, generate and plot some data:

```
>> close all
>> t = 0:pi/20:2*pi;
>> plot(t,sin(t),'k')
```

Next, add a shadow by offsetting the *x* coordinates. Make the shadow line different colors and wider than the default LineWidth:

```
% Take note of the RGB row vectors
>> line(t,sin(t),'LineWidth',1,'Color',[.5 .5 .5])  % The color here is a medium gray
>> line(t +.06,sin(t),'LineWidth',2,'Color',[.01 .8 .8])      % Blue-Green
>> line(t+.16,sin(t),'LineWidth',3,'Color',[.8 .01 .8])       % Purple
>> line(t+.26,sin(t),'LineWidth',4,'Color',[.8 .8 .01])       % Weird Dark Yellow
```

☻ axis

Axis scaling and apprarance

Syntax

```
axis([xmin xmax ymin ymax])
axis square
axis off
axis on
```

Description

`axis` manipulates commonly used Axes properties.

`axis([xmin xmax ymin ymax])` sets the limits for the *x*- and *y*-axis of the current Axes.

`axis square` makes the current Axes region square (or cubed when three-dimensional). MATLAB adjusts the *x*-axis, *y*-axis, and *z*-axis so that they have equal lengths and adjusts the increments between data units accordingly.

`axis off` turns off all axis lines, tick marks, and labels.

`axis on` turns on all axis lines, tick marks, and labels.

Examples

```
>> close all
>> x = 0:1000;
>> plot(exp(-x),'red')
% Now "back off" the view so that no data touches a boundary.
>> axis([-100 1100 -0.05 1.05])
```

☻ title

Add title to current Axes

Syntax

```
title('string')
title(fname)
```

Description

Each Axes graphics object can have one title. The title is located at the top and in the center of the axes.
`title('string')` outputs the string at the top and in the center of the current axes.
`title(fname)` evaluates the function that returns a string and displays the string at the top and in the center of the current axes.
`title(...,'PropertyName',PropertyValue,...)` specifies property name and property value pairs for the Text graphics object that title creates.

Examples

Display a title for the plot below:

```
>> figure(5)
>> plot([0 1 2 3 4 5],[5 4 3 2 1 0])
>> title('Line with slope of -1')
```

Include Greek symbols in a title:

```
>> title('\ite^{\omega\tau} = cos(\omega\tau) + isin(\omega\tau)')
%  FYI - The "ite" stands for italicized ("it") exponential ("e").

% Please do not close Figure 5.  We will be using it in the next example.
```

☻ **xlabel, ylabel**

Label the *x*-, *y*-, and *z*-axis

Syntax

```
   xlabel('string')
   xlabel(...,'PropertyName',PropertyValue,...)
```

Description

Each Axes graphics object can have one label for the *x*-, *y*-, and *z*-axis. The label appears beneath its respective axis in a two-dimensional plot and to the side or beneath the axis in a three-dimensional plot.
`xlabel('string')` labels the x-axis of the current Axes.
`xlabel(...,'PropertName',PropertyValue,...)` specifies property name and property value pairs for the Text graphics object created by xlabel.

Remarks

Re-issuing an xlabel or ylabel command causes the new label to replace the old label.

Examples

% Apply these examples to the figures from `title` example plots.

```
>> xlabel('The date in the X-Axis is [0 1 2 3 4 5]')
>> Label_for_Y = ('The data for Y is equal to 5:-1:0')
>> ylabel(Label_for_Y)                % Notice the absence of single quotes.
```

num2str

Number to string conversion.

Syntax

```
str = num2str(A)
```

Description

The `num2str` function converts numbers to their string representations. This function is useful for labeling and titling plots with numeric values.
`str = num2str(a)` converts array A into a string representation `str` with roughly four digits of precision and an exponent if required.

Examples

Include a variable's value in a title:

```
>> f = 70;
>> c = (f--32)/1.8;
>> figure(4)
>> title(['Temperature is ',num2str(c),'C'])
```

Include a variable's value in a title and set the color of the title to blue:

```
>> n = 3;
>> xlabel(['Case number #',num2str(n)],'Color','b')
```

☻ text

Create a Text object in the current Axes

Syntax

```
text(x,y,'string')
text(x,y,z,'string')
text(...'PropertyName',PropertyValue...)
```

Description

text is the low-level function for creating Text graphics objects. Use `text` to place character strings at specified locations.
`text(x,y,'string')` adds the string in quotes to the location specified by the point `(x,y)`.
`text(x,y,z,'string','PropertyName',PropertyValue....)` adds the string in quotes to location defined by the coordinates and uses the values for the specified Text properties. See the "Text Properties" section for a list of Text object properties.
`text('PropertyName',PropertyValue....)` omits the coordinates entirely and specifies all properties using property name/property value pairs.

Examples

```
>> plot(1:200,1:200)
>> text(1,100,'\fontname{courier} \fontsize{10} \tau_{\rho} * \gamma^{17}')
>> text(5, 170, 'You can display just about any symbol you wish this way.')
```

Adding Titles, Axis Labels, and Annotations

```
>> close all;
>> t = 0:pi/20:30*pi;
>> plot(t,50*sin(t),'green')
>> xlabel('-\pi \leq \Theta \leq \pi')
>> ylabel('sin(\Theta)')
>> title('Plot of sin(\Theta)')
>> text(pi/4,sin(pi/4),'\leftarrow sin(\pi\div4)')
```

% Big help: To find out more about displaying symbols (Greek or otherwise) enter the
% following at the MATLAB Prompt. (You'll have to scroll down some.)

```
>> doc text_props
```

gtext

Mouse placement of text in two-dimensional view.

Syntax

```
  gtext('string')
```

Description

`gtext` displays a text string in the current Figure window after you select a location with the mouse.
`gtext('string')` waits for you to press a mouse button or keyboard key while the pointer is within a Figure window. Pressing a mouse button or any key places '*string*' on the plot at the selected location.

Remarks

As you move the pointer into a Figure window, the pointer becomes a crosshair to indicate that `gtext` is waiting for you to select a location.

Examples

Place a label on the current plot:

```
>> gtext('Note the big crosshairs!')
```

% Just move your mouse to the plot and click (with the left button) anywhere you like.

☻ legend

Display a legend for an Axes

Syntax

```
legend('string1','string2',...)
legend(Strings)
legend(h,Strings)
legend('off')
```

Description

`legend` places a legend on a graph. For each line in the plot, the legend shows a sample of the line type, marker symbol, and color beside the text label you specify. When plotting filled areas, the legend contains a sample of the face color next to the text label. After the legend appears, you can move it using the mouse.
`legend('`*`string1`*`','`*`string2`*`',...)` displays a legend in the current Axes using the specified strings to label each set of data.
`legend(Strings)` adds a legend containing the rows of the matrix `Strings` as labels. This is the same as `legend(Strings(1,:),Strings(2,:),...)`.
`legend(h,Strings)` associates each row of the matrix `Strings` with the corresponding graphics object in the vector `h`.
`legend('off')` removes the legend from the current Axes or the Axes specified by `h`.

- `pos = -1` places the legend outside the Axes boundary on the right side.
- `pos = 0` places the legend inside the Axes boundary, avoiding data if possible.
- `pos = 1` places the legend in the upper-right corner of the Axes (default).
- `pos = 2` places the legend in the upper-left corner of the Axes.
- `pos = 3` places the legend in the lower-left corner of the Axes.
- `pos = 4` places the legend in the lower-right corner of the Axes.
- `pos = [XlowerLeft YlowerLeft]` explicitly specifies the lower-left legend position in normalized coordinates.

Remarks

MATLAB displays only one legend per Axes. `legend` positions the legend based on a variety of factors, such as what objects the legend obscures. You move the legend by pressing the mouse button while the cursor is over the legend and dragging the legend to a new location. If your mouse has more than one button, you press the left mouse button.

Examples

Add a legend to a graph showing a sine and cosine function:

```
>> close all;
>> x = -pi:pi/20:pi;
>> plot(x,cos(x),'-r',x,sin(x),'-.b')       % A sine wave - one full cycle
>> legend('cos','sin',0)                    % Legend should avoid any data here
>> legend('off')                            % Legend should disappear
>> legend('cos','sin',1)                    % Legend in upper-right corner
>> legend('cos','sin',-1)                   % Legend outside of axis box
```

Notice how easily the legend box can be moved just by clicking on it and dragging it (using the left mouse button).

```
>> legend('off')
```

☻ grid

Grid lines for two- and three-dimensional plots.

Syntax

```
   grid on
   grid off
   grid
```

Description

The `grid` function turns the current Axes' grid lines on and off.
`grid on` adds grid lines to the current Axes.
`grid off` removes grid lines from the current Axes.
`grid` toggles the grid visibility state.

Note - The grid marks follow/align with the ticks marks on the axis.

Examples

```
>> D = 0:100;
>> E = D.^2:
>> plot(D,E)
>> grid on
>> grid off
>> grid
```

Odds and Ends

inf
Infinity

Syntax
```
      Inf
```

Description
`Inf` returns the IEEE arithmetic representation for positive infinity. Infinity results from operations like division by zero and overflow, which lead to results too large to represent as conventional floating-point values.

Examples
```
>> 1/0
>> 1.e1000
>> 2^10000
>> exp(1000)
```
all produce `Inf`.

```
>> log(0)                % As we now know.
```
produces `-Inf.`

NaN
Not-a-Number

Syntax
```
   NaN
```

Description
`NaN` returns the IEEE arithmetic representation for Not-a-Number `NaN`. These result from operations which have undefined numerical results.

Remarks
Logical operations involving `NaN`s always return false, except ~= (not equal). Consequently, the statement `NaN ~= NaN` is true while the statement `NaN == NaN` is false.

Examples
These operations produce `NaN`:

- Any arithmetic operation on a `NaN`, such as `sqrt(NaN)`

- Addition or subtraction, such as magnitude subtraction of infinities as `(+Inf)+(-Inf)`
- Multiplication, such as `0*Inf`
- Division, such as `0/0` and `Inf/Inf`

```
>> Inf-Inf
>> Inf/Inf
```
both produce `NaN`, Not-a-Number.

format

Control the output display format.

Syntax

MATLAB performs all computations in double precision. The `format` command described below switches among different display formats. See table below:

Command	**Result**	**Example**
`format`	Default. Same as `short`.	
`format short`	5 digit scaled fixed point	`3.1416`
`format long`	15 digit scaled fixed point	`3.14159265358979`
`format short e`	5 digit floating-point	`3.1416e+00`
`format long e`	15 digit floating-point	`3.141592653589793 e+00`
`format short g`	Best of 5 digit fixed or floating	`3.1416`
`format long g`	Best of 15 digit fixed or floating	`3.14159265358979`
`format hex`	Hexadecimal	`400921fb54442d18`
`format bank`	Fixed dollars and cents	`3.14`
`format +`	+,-, blank	`+`

Algorithms

The command `format +` displays +, -, and blank characters for positive, negative, and zero elements.

`format hex` displays the hexadecimal representation of a binary double-precision number.

Note - Be aware that just executing a format command will not print or output anything.

Changes from the format command to the command window will only occur when output is displayed.

Examples

```
>> format short
>> pi
>> format short e
>> pi
>> format long
>> pi
>> format long e
>> pi
>> format hex
>> pi
```

save

Save workspace variables on disk

Syntax:

```
     save
     save filename
     save filename variables
     save filename options
```

Description:

`save`, by itself, stores all workspace variables in a binary format in the file named matlab.mat. The data can then be retrieved with the command load.

`save filename` stores all workspace variables in filename.mat instead of the default matlab.mat.

`save filename variables` saves only the workspace variables you list after the filename.

Remarks:

The `save` and `load` commands retrieve and store MATLAB variables on disk. They can also import and export numeric matrices as ASCII data files.

Variables saved in the binary MAT-file format can be saved in a file with the same filename as the same variables saved in ASCII data format.

Example:

```
>> dir FILE1.*;  A = 57;  B = 5;
>> save FILE1 A B -ascii
```

will create an ASCII file named FILE1 that contains A and B as text.

```
>> save FILE1 A B
```

will create a MAT-file named FILE1.mat that contains the numerical values of A and B.

```
>> dir FILE1.*
```

Notice that both of these files are now listed in the working subdirectory.

Alternative syntax:

The function form of the syntax, save('filename'), is also permitted.

load

Retrieve variables from disk

Syntax:

```
     load
     load filename
     load (str)
     load filename.ext
     load filename -ascii
     load filename -mat
```

Description:

The `load` and `save` commands retrieve and store MATLAB variables on disk.

`load` by itself, loads all the variables saved in the file 'matlab.mat'.

`load filename` retrieves the variables from 'filename.mat'.

The following code will work as an alternative using the load command:

```
str = 'filename.mat'; load (str)
```

Doing so will retrieve the variables from 'filename.mat' as long as filename.mat exists.

Examples:

```
>> clear all
>> whos       % Notice that there are no variables
>> load FILE1
>> whos       % Notice that the variables A and B are back.
```

To display FILE1 as ascii text in the command window, enter:

```
>> type FILE1
```

Remarks:

MAT-files are double-precision binary MATLAB format files created by the save command and readable by the load command.

References:

- The MathWorks, Inc. Online (Helpdesk) Documentation
- Using MATLAB Version 5.0
  by The MathWorks, Inc.
- Getting Started with MATLAB Version 5.0
  by The MathWorks, Inc.