Chapter 6

Plotting

MATLAB does an incredible job with plotting data. I still remember the 1st time I played around with the different commands. My impression can be best described as WOW. This chapter deals with graphing capabilities and provides examples.

6.1 The basic plot command

The plot command will "plot" a vector. For vector x

```
>>   plot(x)
```

treats the values stored in x as abscissa values, and plots these against their index. With two vectors of the same length and the same type, i.e., both must be either row or column vectors, we can also plot these against each other like

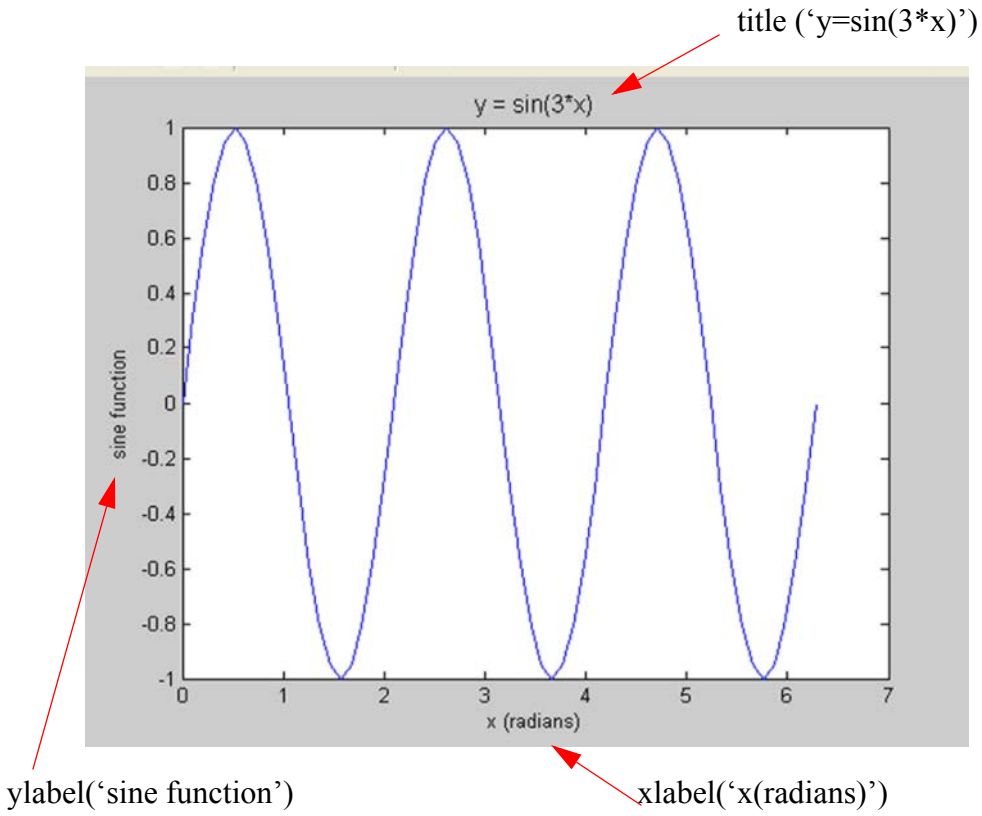
```
>>  plot(x, y)   %produces a graph of y versus x
```

6.1.1 A simple line plot

Here are the MATLAB commands to create a simple plot of `y = sin(3*pi*x)` from `0 to 2*pi`.

```
%  File:  sin3xPlot.m
%
%  A sample MATLAB script to plot y = sin(3*x) and label the plot

x = 0:pi/30:2*pi;         %  x vector, 0 <= x <= 2*pi, increments of pi/30
y = sin(3*x);             %  vector of y values
plot(x,y)                 %  create the plot
xlabel('x (radians)');    %  label the x-axis
ylabel('sine function');  %  label the y-axis
title('y = sin(3*x)','FontSize', 12);    %  put a title on the plot
```

The effect of the labeling commands, xlabel, ylabel, and title are indicated by the text and red arrows in the figure. **Don't skimp on the labels!**



6.2 Basic plotting

We can also combine several lines like

```
plot(x, y, x, f, x, z)
```

or

```
plot(x, y, 'bo:', x, f, 'gV--')
```

6.3 Creating Symbol Plots with MATLAB

Changing symbol or line types

The symbol or line type for the data can by changed by passing an optional third argument to the plot command. For example

```
>> t = 0:pi/30:2*pi;
>> plot(cos(3*t),sin(2*t),'o');                    % blue (default) circles
```

plots data in the x and y vectors using circles drawn in the default color (blue), and

```
>> plot(cos(3*t),sin(2*t),'r:');                    % red dotted line
```

plots data in the x and y vectors by connecting each pair of points with a red dotted line.

The third argument of the plot command is a one, two or three character string of the form 'cs', where 'c' is a single character indicating the color and 's' is a one or two character string indicating the type of marker type or line. The color selection is optional. Allowable color and marker types are summarized in the following tables. Refer to ``help plot'' for further information.

Color, Marker Types and Line Type Selectors for 2-D plots

'c' character	**symbol & line color**
'y'	yellow
'm'	magenta
'c'	cyan
'r'	red
'g'	green
'b'	blue
'w'	white
'k'	black

'ss' string	**marker types and filled marker types**
'.'	point
'o'	circle
'x'	x-mark
'+'	plus
'*'	star
's'	square
'd'	diamond
'^'	up triangle
'v'	down triangle
'>'	right triangle
'<'	left triangle
'p'	pentagram
'h'	hexagram
'none'	no marker

'ss' string	**Linestyle**
'-'	solid line
':'	dotted line
'-.'	dashdot line
'--'	dashed
'none'	no line

6.3.1 A simple symbol plot

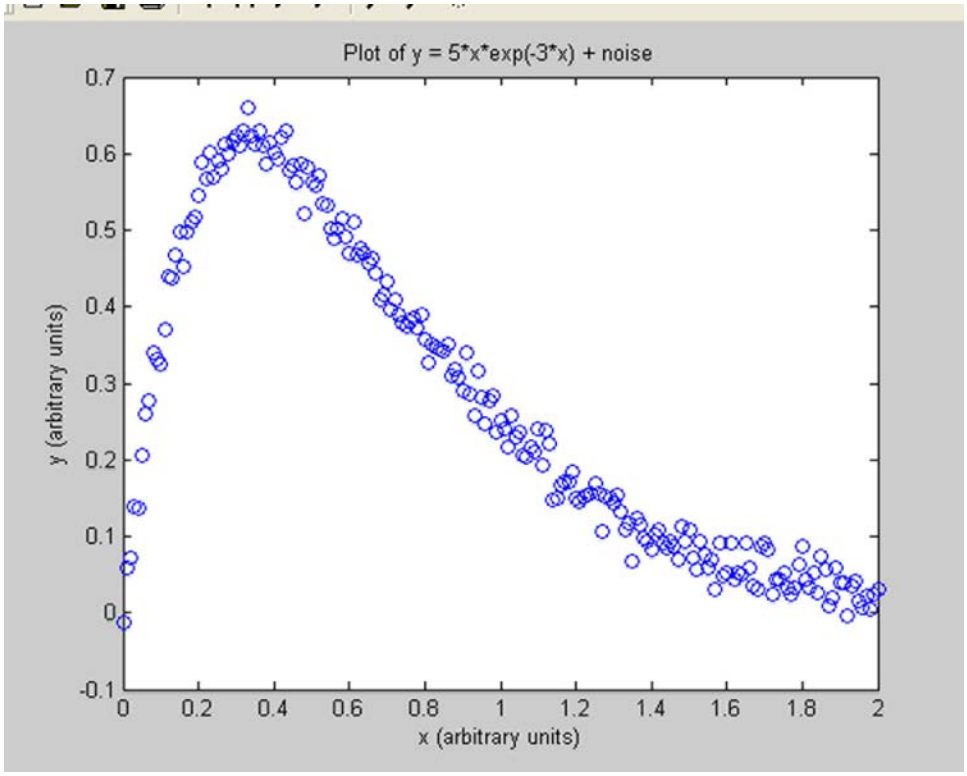
This example shows you how to plot data with symbols. This is done by indicating a marker type but NOT a linestyle. For example,

```
>> plot(x, y, 'ks')
```
plots black ('k') squares ('s') at each point, but does not connect the markers with a line.
The statement
```
>> plot(x, y, 'r:+')
```
plots a red ('r') dotted line (':') and places plus sign markers ('+') at each data point. This type of plot is appropriate, for example, when connecting data points with straight lines would give the misleading impression that the function being plotted is continuous between the sampled data points.

Here are the MATLAB commands to create a symbol plot with the data generated by adding noise to a known function.

```
%  File:  noisyData.m
%
%  Generate a noisy function and plot it.

x = 0:0.01:2;                 %  generate the x-vector
noise = 0.02*randn(size(x));  %  and noise
y = 5*x.*exp(-3*x) + noise;   %  Add noise to known function
plot(x,y,'o');                %  and plot with symbols

xlabel('x (arbitrary units)');          %  add axis labels and plot title
ylabel('y (arbitrary units)');
title('Plot of y = 5*x*exp(-3*x) + noise');
```

Note that the ".*" operator is necessary when multiplying the vector `x` by the vector `exp(-3*x)`.
The preceding statements create the following plot.



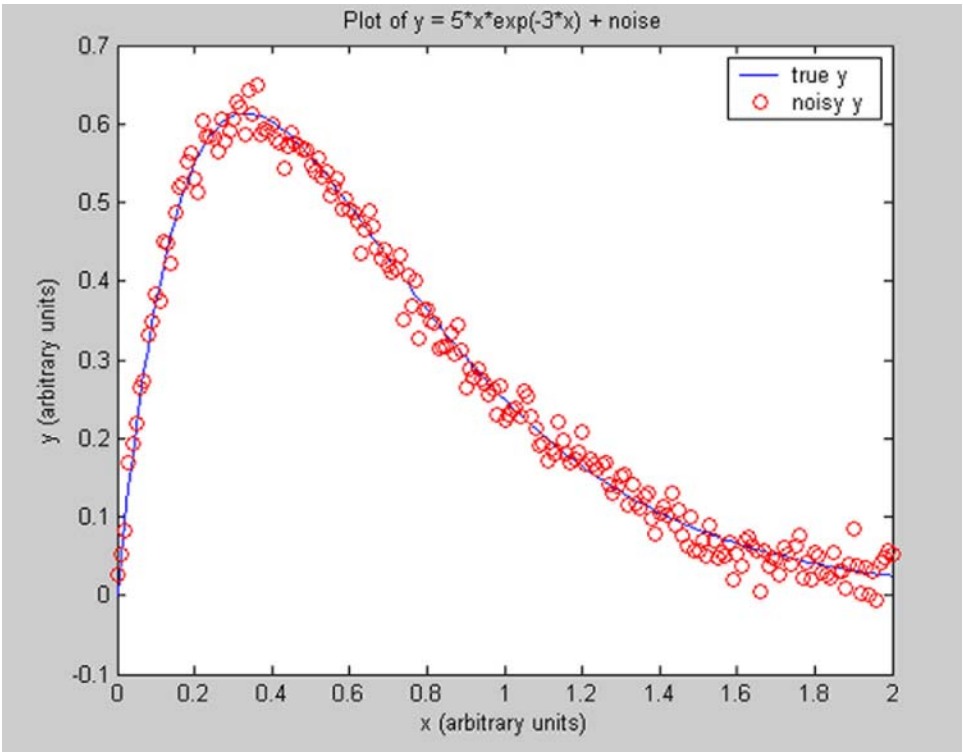
You can control the color and line style of this plot with optional string arguments like

```
plot(x, 'r.')
```

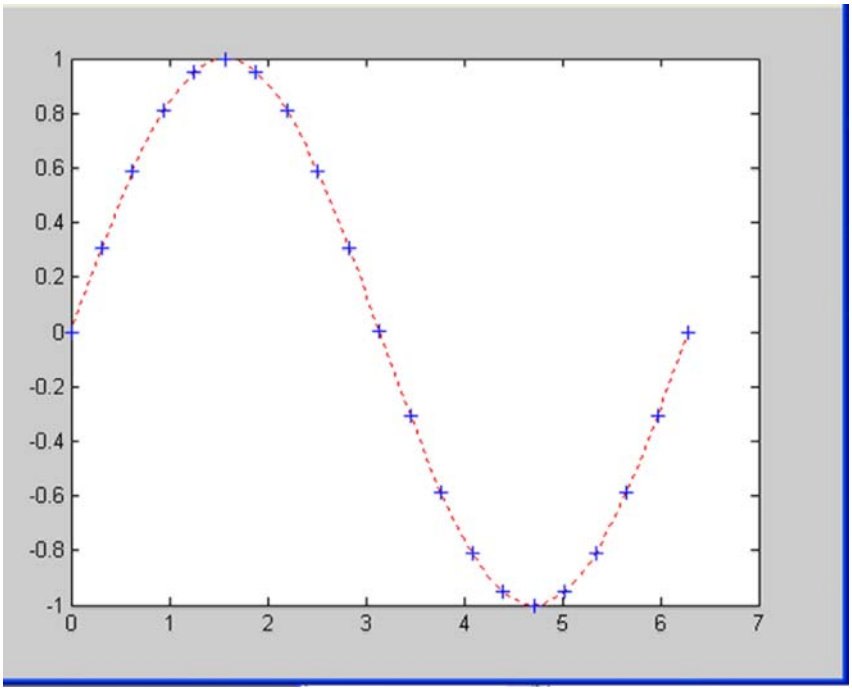
which plots as a set of red dots.

There may be times where you will want the function plotted along with the noisy data. This is done by putting them both into the "plot". See the example below

```
%  File:  noisyDataLine.m
%
%  Generate a function and add noise to it.  Plot the function
%  as a solid line and the noisy data as open circles

x = 0:0.01:2;                 %  generate the x-vector
y = 5*x.*exp(-3*x);           %  and the "true" function, y - by formula
yn = y + 0.02*randn(size(x)); %  Create a noisy version of y
plot(x,y,'-',x,yn,'ro');      %  Plot the function and the noisy one too

xlabel('x (arbitrary units)');          %  add axis labels and plot title
ylabel('y (arbitrary units)');
title('Plot of y = 5*x*exp(-3*x) + noise');
legend('true y','noisy y');
```



When you plot the markers and lines on the same plot, you may want to use fewer data points to plot the markers than you use to plot the lines. This example plots the data twice using a different number of points for the data line and marker plots.



Every time you plot, the new plot replaces the old by default. To change this behavior either use hold on, open a new figure window, or use a subplot.

6.4 hold

The command

hold on

causes subsequent plotting commands to be added to the existing set of plot axes rather than replacing the existing graph. A command of

hold off

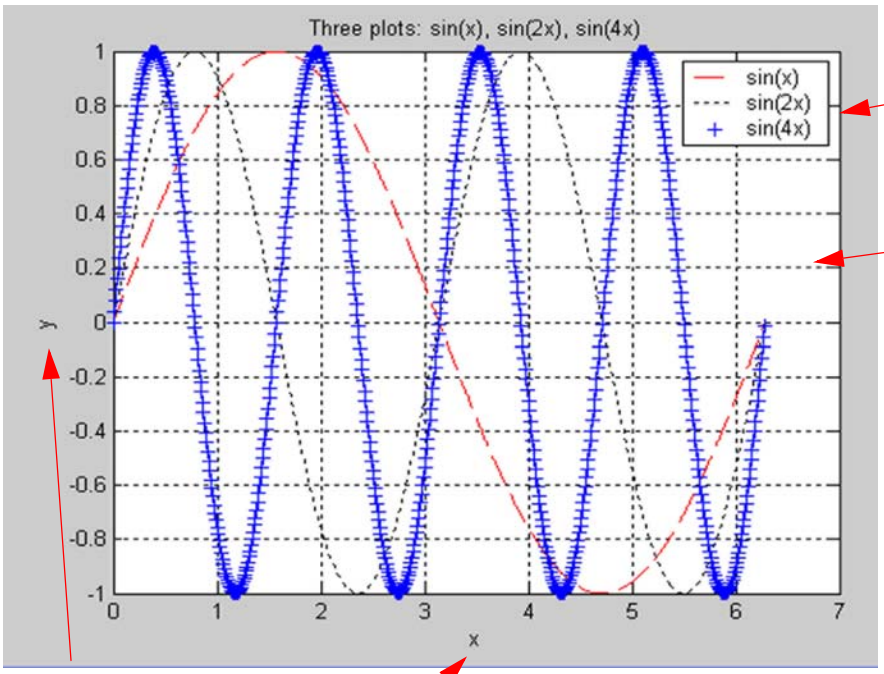
will restore the default behavior of replacing the current plot contents with the new plot.
The command

hold

will toggle between hold on and hold off.

The following puts three plots on the same set of axes.

```
x = 0: 0.01:2*pi;
plot(x, sin(x), 'r--')
hold on
plot(x,sin(2*x),'k:')
plot(x,sin(4*x),'b+')
grid
title ('Three plots: sin(x), sin(2x), sin(4x)')
xlabel('x')
ylabel('y')
legend('sin(x)','sin(2x)','sin(4x)')
```



We have used two new command in this last example:

```
grid on   % puts in the hashing
legend  %the tables associated with the lines plotted.
```

The basic form of legend is:

```
legend ('string1', 'string2', ... , pos)
```

where `string1`, `string2`, and so forth, are the labels associated with the lines plotted, and `pos` is an integer specifying where to place the legend. The command `legend off` will remove the existing `legend`. The possible values for `pos` are:

Values of `pos` in the `legend` Command	
Value	**Meaning**
`0`	Automatic "best" placement (least conflict with data)
`1`	Upper right-hand corner (default)
`2`	Upper left-hand corner
`3`	Lower left-hand corner
`4`	Lower right-hand corner
`-1`	to the right of the plot

6.5 Controlling the Axes

The `axis` command supports a number of options for setting the scaling, orientation, and aspect ratio of plots.

Command	**Description**
`axis([xmin xmax ymin ymax])`	sets scaling for the x- and y-axes on the current plot.
`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 tight`	sets the axis limits to the range of the data

Command	Description
`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 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 off`	turns off all axis labeling, tick marks and background.
`axis on`	turns axis labeling, tick marks and background back on

6.6 Figure Windows

Graphing functions automatically open a new figure window if there are no figure windows already on the screen. If a figure window exists, MATLAB uses that window for graphics output. If there are multiple figure windows open, MATLAB targets the one that is designated the "current figure" (the last figure used or clicked in).

To make an existing figure window the current figure, you can click the mouse while the pointer is in that window or you can type

```
figure(n)
```

where **n** is the number in the figure title bar.

6.6.1 Clearing the Figure for a New Plot

When a figure already exists, most plotting commands clear the axes and use this figure to create a new plot. However, these commands do NOT reset figure properties, such as the background color or the colormap. If you have set any figure properties in the previous plot, you may want to use the **clf** command with the `reset` option

```
clf reset
```

before creating your new plot to set the figure's properties to their defaults.

6.7 Formatting Output

It is possible to enhance plotted text strings (titles, axis labels, etc.) with formatting such as bold face, italics, as well as special characters--Greek and mathematical symbols.

The font used to display the text can be modified by stream modifiers--a special sequence of characters that tells the MATLAB interpreter to change its behavior. Once a stream modifier has been inserted into a text string, it will remain in effect until the end of the string or until cancelled. If a modifier is followed by brackets { }, only the text in the brackets is affected.

Formatting Modifiers			
Modifier	**Effect**	**Example**	**Result**
`\bf`	Boldface	\bfThis is Bold\rm	**This is Bold**
`\it`	Italics	\itThis is Italics\rm	*This is Italics*
`\rm`			
`\fontname{fontname}`	Specify the font name to use	\fontname{courier}	
`\fontsize{fontsize}`	Specify font size	\fontsize{16}	
`_{xxx}`	Characters inside the braces are subscripts	x_{12}	x_{12}
`^{xxx}`	Characters inside the braces are superscripts	y^{3}	y^3

Greek and Mathematical Symbols in Titles/Labels Chapman p 116							
Character Sequence	**Symbol**		**Character Sequence**	**Symbol**		**Character Sequence**	**Symbol**
`\alpha`	α					\int	
`\beta`	β					\cong	
`\gamma`	γ		\Gamma	Γ		\sim	
`\delta`	δ		\Delta	Δ		\infty	
`\epsilon`	ε					\pm	
`\eta`	η					\leq	

Greek and Mathematical Symbols in Titles/Labels Chapman p 116					
Character Sequence	**Symbol**	**Character Sequence**	**Symbol**	**Character Sequence**	**Symbol**
\theta	θ			\geq	
\lambda	λ	\Lambda	Λ	\neq	
\mu	μ			\propto	
\nu	ν			\div	
\pi	π	\Pi	Π	\circ	
\phi	φ	\Phi	Φ	\leftrightarrow	
\rho	ρ			\leftarrow	
\sigma	σ	\Sigma	Σ	\rightarrow	
\tau	τ			\uparrow	
\omega	ω	\Omega	Ω	\downarrow	

6.8 SubPlot

The subplot command enables you to display multiple plots in the same window.
`H = SUBPLOT(m,n,p)`, or `SUBPLOT(mnp)`, breaks the Figure window into an `m-by-n` matrix of small axes, selects the `p-th` axes for the current plot, and returns the axis handle. The axes are counted along the top row of the Figure window, then the second row, etc. For example,

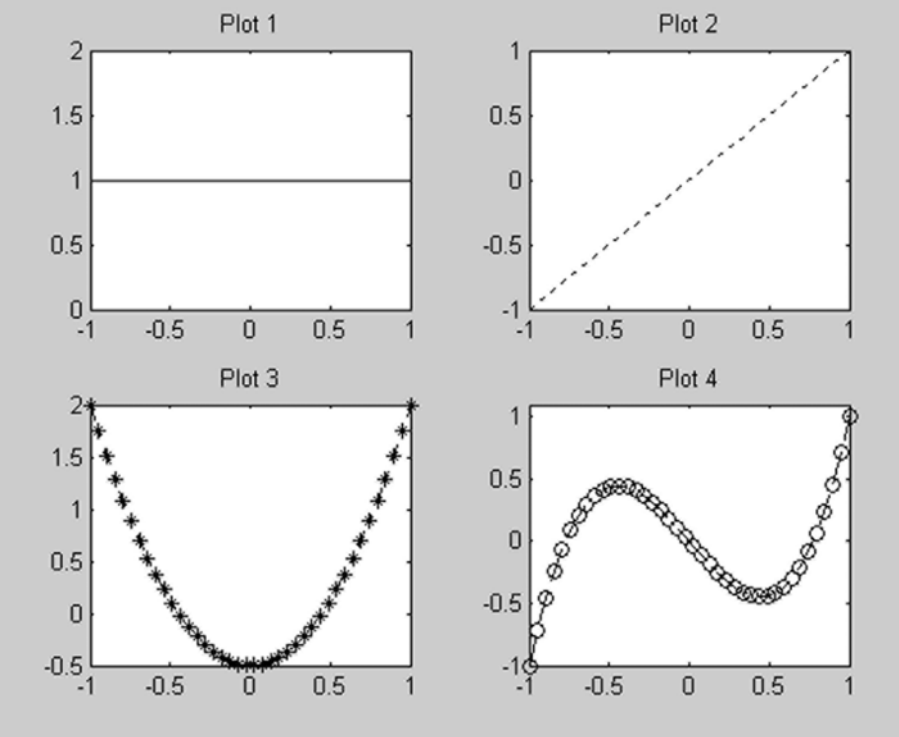
```
SUBPLOT(2,1,1), PLOT(income)
SUBPLOT(2,1,2), PLOT(outgo)
```

plots `income` on the top half of the window and `outgo` on the bottom half.

`SUBPLOT(m,n,p)`, if the axis already exists, makes it current.
`SUBPLOT(m,n,p,'replace')`, if the axis already exists, deletes it and creates a new axis.
`SUBPLOT(m,n,P)`, where `P` is a vector, specifies an axes position that covers all the subplot positions listed in P.

```
%SUBPLOT can be used to partition the graphics screen so that either
% two or four plots are displayed simultaneously
```

```
x = linspace(-1, 1, 40);
f0 = ones(1, length(x));    %constant
f1 = x;                     %linear
f2 = 1/2*(5*x.^2-1);        %quadratic
f3 = 1/2*(5*x.^3-3*x);      %cubic

subplot(2,2,1),plot(x,f0,'k-'),title('Plot 1')
subplot(2,2,2),plot(x,f1,'k:'),title('Plot 2')
subplot(2,2,3),plot(x,f2,'k-.*'),title('Plot 3')
subplot(2,2,4),plot(x,f3,'k--o'),title('Plot 4')
axis([-1,1,-1,1.1])
zoom on
```



6.9 Additional 2-D Plots

Bar Chart

BAR(X,Y) draws the columns of the M-by-N matrix Y as M groups of N vertical bars. The vector X must be monotonically increasing or decreasing.

BAR(Y) uses the default value of X=1:M. For vector inputs, BAR(X,Y) or BAR(Y) draws LENGTH(Y) bars. The colors are set by the colormap.

BAR(X,Y,WIDTH) or BAR(Y,WIDTH) specifies the width of the bars. Values of WIDTH > 1, produce overlapped bars. The default value is WIDTH=0.8

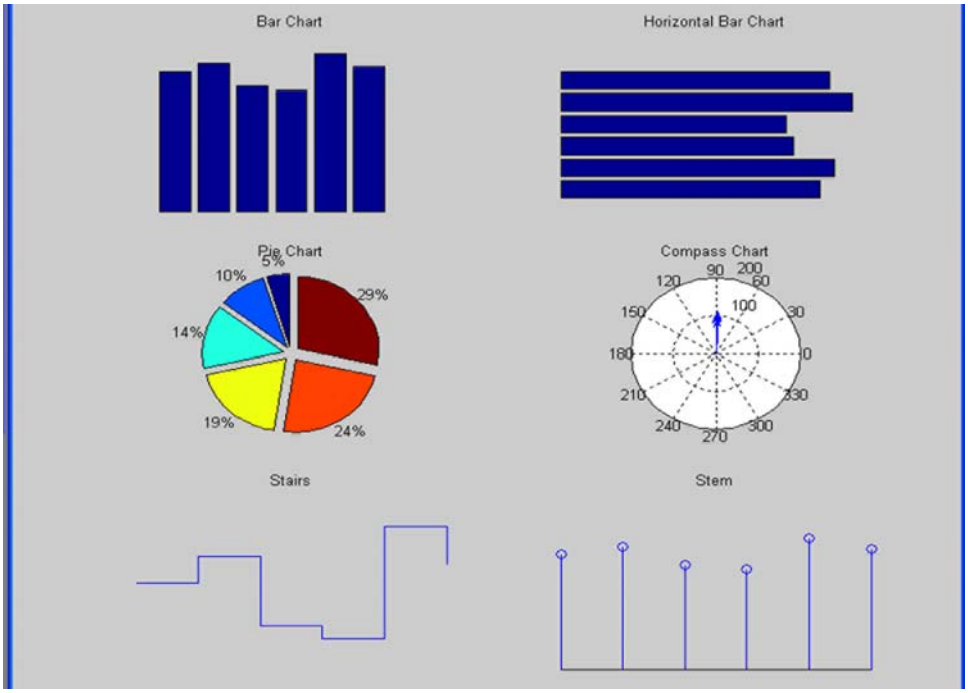
```
BAR(...,'grouped') produces the default vertical grouped bar chart.
BAR(...,'stacked') produces a vertical stacked bar chart.
BAR(...,LINESPEC) uses the line color specified (one of 'rgbymckw').

     H = BAR(...) returns a vector of patch handles.

Use SHADING FACETED to put edges on the bars.  Use SHADING FLAT to turn them
off.

Examples: subplot(3,1,1), bar(rand(10,5),'stacked'), colormap(cool)
          subplot(3,1,2), bar(0:.25:1,rand(5),1)
          subplot(3,1,3), bar(rand(2,3),.75,'grouped')
```

Horizontal Bar Chart - `barh(x,y)`

Compass

```
    COMPASS(U,V) draws a graph that displays the vectors with
    components (U,V) as arrows emanating from the origin.
```

Pie - pie chart
Stairs - Image stair steps
Stem - They remind me of the stems of tulips
Histograms - Yep these are an easy way to draw histograms
Polar - Use when you want polar coordinates




```
x = 1:6;
y = zeros(1,6);
for i = 1:600
```

```
        ii = ceil(6 * rand(1));
        y(1,ii)= y(1,ii) + 1;
end,

subplot (3,2,1)
bar(x,y)
title('Bar Chart')
axis off

subplot (3,2,2)
barh(x,y)
title('Horizontal Bar Chart')

subplot (3,2,3)
pie(x,y)
title('Pie Chart')

subplot (3,2,4)
compass(x,y)
title('Compass Chart')

subplot (3,2,5)
stairs(x,y)
title('Stairs')

subplot (3,2,6)
stem(x,y)
title('Stem')
```

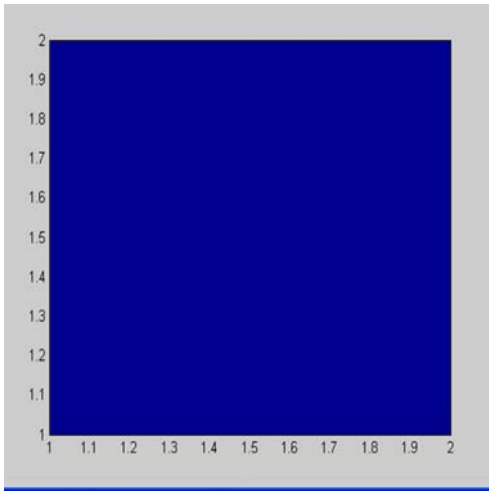
6.10 Surface Plotting

There is a relationship between 2-D data and 3-D plots. By 2-D data, I mean a matrix of data (many rows and columns). The rows and columns implicitly imply `x - y` locations, and the values stored at each row, column are a height. Simplistically, this height can be represented graphically by `pcolor`. With the default shading, the `pcolor` command does not do quite what many expect. Consider

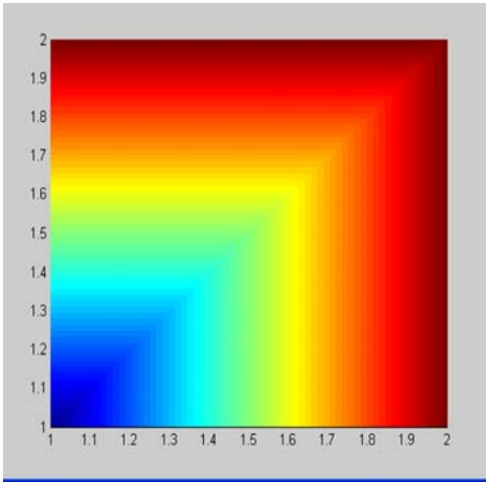
```
f = [1 10; 10 10]
pcolor(f)
```

This shows only a single patch of color. Indeed the 10's make no difference to the plot. But with `shading interp` the color is interpolated to the corners of the plot, where the 10's live. With flat and faceted shading the color of each surface patch is set based on only one corner value (the one with the lowest row and column index). So such data really describes a surface in 3D. MAT-

LAB defines a surface by the z-coordinates of points above a grid in the x-y plane, using straight lines to connect adjacent points.



pcolor



shading interp

Other commands of note for plotting 2-D data: mesh, surf, contour. The `mesh` and `surf` plotting functions display surfaces in three dimensions. `mesh` produces wireframe surfaces that color only the lines connecting the defining points. `surf` displays both the connecting lines and the faces of the surface in color.

6.10.1 Mesh and Surface Plots

Mesh and surface plots are generated in a three step process.

1) create a grid in the x-y plane that covers the domain of the function
2) calculate the value of z at each point of the grid
3) create the plot

Step 1: create a grid in the x-y plane that covers the domain

Example: if our domain is:

```
      -1 <= x <= 3  and 1 <= y <= 4
 x = -1:1:3
x =
    -1     0     1     2     3

>> y = 1:1:4
y =
     1     2     3     4
```

MATLAB has a function that will create the grid:
meshgrid

```
>> [X Y] = meshgrid(x,y)

X =
    -1     0     1     2     3
```

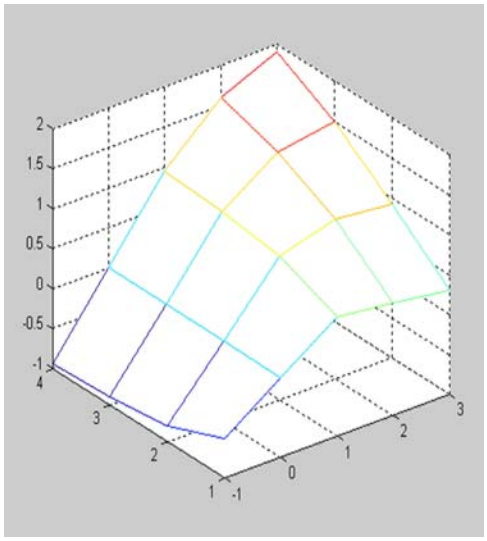
```
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
Y =
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
```

Step 2: calculate the value of z at each point of the grid
The value of z at each point is calculated by using element-by-element calculations. If the formula is: $z = \frac{xy^2}{x^2 + y^2}$ The value of z at each point of the grid is calculated by:

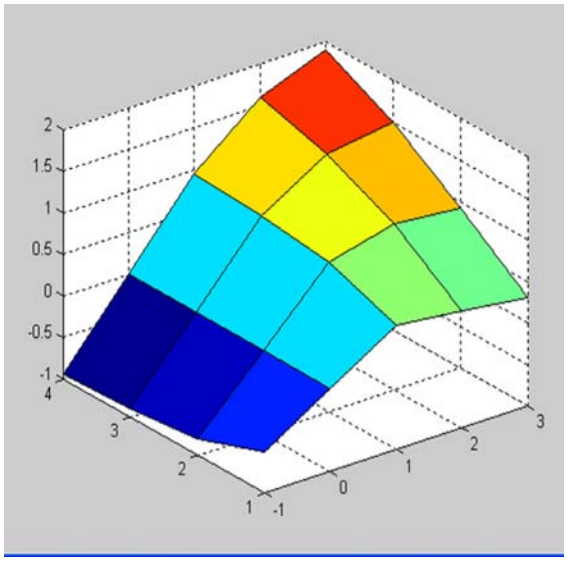
```
Z = X.*Y.^2./(X.^2+Y.^2)

Z =
   -0.5000         0    0.5000    0.4000    0.3000
   -0.8000         0    0.8000    1.0000    0.9231
   -0.9000         0    0.9000    1.3846    1.5000
   -0.9412         0    0.9412    1.6000    1.9200
```

Step 3: create the plot
Now you can create the plot.

```
mesh(X,Y,Z)
```



mesh(X,Y,Z)

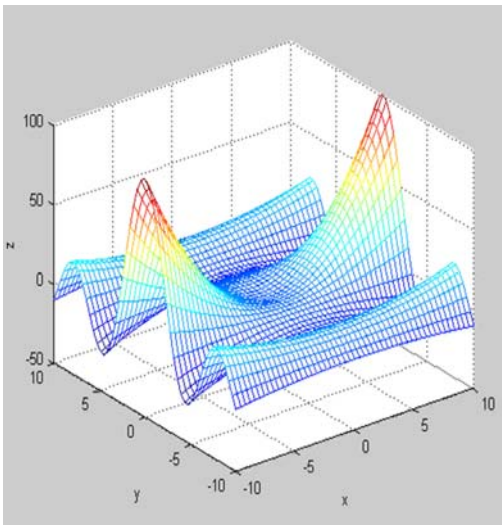


surf(X,Y,Z)

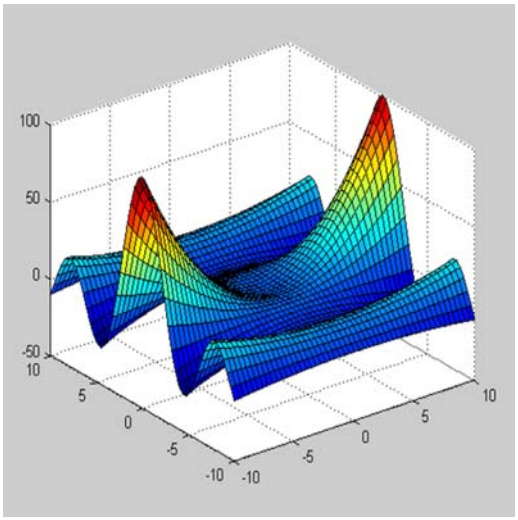
6.10.2 Another Example:

```
xx = -10: 0.4: 10;
```

```
yy = xx;
yy = yy + (yy == 0)*eps
[x,y] = meshgrid(xx,yy);
z = (x.^2 + y.^2).*sin(y)./y
mesh(xx,yy,z)
xlabel('x'),ylabel('y'),zlabel('z')
```



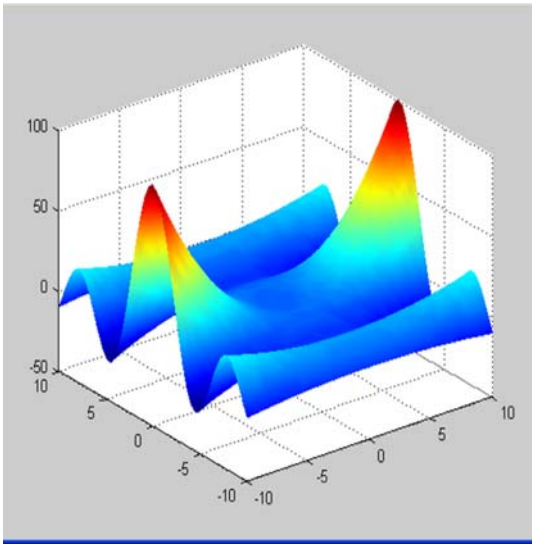
mesh(xx,yy,z)



surf(xx,yy,z)

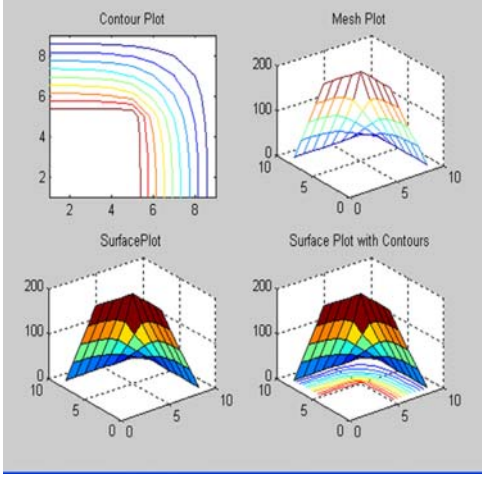
It is strongly suggested that you use `shading interp` following the `surf` and `mesh` commands.

```
shading interp % using linear interpolation on height to color surface
```



shading interp

```
%define array of temperatures in chimney cross section
temp = [NaN     NaN     NaN     NaN     200.0   147.3   96.5    47.7    0.0;
        NaN     NaN     NaN     NaN     200.0   146.9   96.0    47.4    0.0;
        NaN     NaN     NaN     NaN     200.0   145.2   93.9    46.1    0.0;
        NaN     NaN     NaN     NaN     200.0   140.4   88.9    43.2    0.0;
        200.0   200.0   200.0   200.0   200.0   128.2   78.8    37.8    0.0;
        147.3   146.9   145.2   140.4   128.2   94.3    60.6    29.6    0.0;
        96.5    96.0    93.9    88.9    78.8    60.6    40.2    19.9    0.0;
        47.7    47.4    46.1    43.2    37.8    29.6    19.9    10.0    0.0;
        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0];


%generate contour and surface subplots

subplot(2,2,1);  contour(temp)
title('Contour Plot');
subplot(2,2,2);  mesh(temp)
title('Mesh Plot');
subplot(2,2,3);  surf(temp)
title('SurfacePlot');
subplot(2,2,4);  surfc(temp)
title('Surface Plot with Contours');
```



6.10.3 Plot3

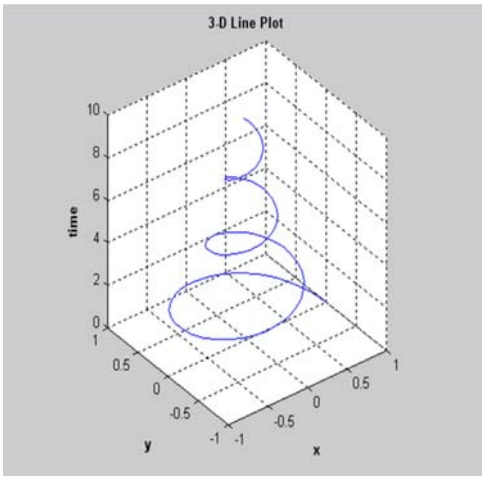
MATLAB uses the function `plot3` to display arrays of data points `(xi, yi, zi), i = 1, 2, 3, ..., n` in three-dimensional space. The syntax is `plot3(x, y, z)`. When x, y, and z are vectors of the same length, this function plots a line in three-dimensional space through the points whose coordinates are the elements of x, y, and z.

Example: a three-dimensional line plot

$x(t) = e^{-0.2t}\cos(2t)$

$y(t) = e^{-0.2t}\sin(2t)$

These functions might represent the decaying oscillations of a mechanical system in two dimensions. Therefore x and y together represent the location of the system at any given time.




```
>> t=0:0.1:10;
   x = exp(-0.2*t).*cos(2*t);
   y = exp(-0.2*t).*sin(2*t);
   plot3(x,y,t);
   title('\bf3-D Line Plot');
   xlabel('\bfx');
   ylabel('\bfy');
   zlabel('\bftime');
   axis square;
   grid on;
```