MA 302: MATLAB Laboratory, Spring 2004

Graphics in MATLAB: An Overview[1]

1. BASIC PLOTTING

Creating a Plot

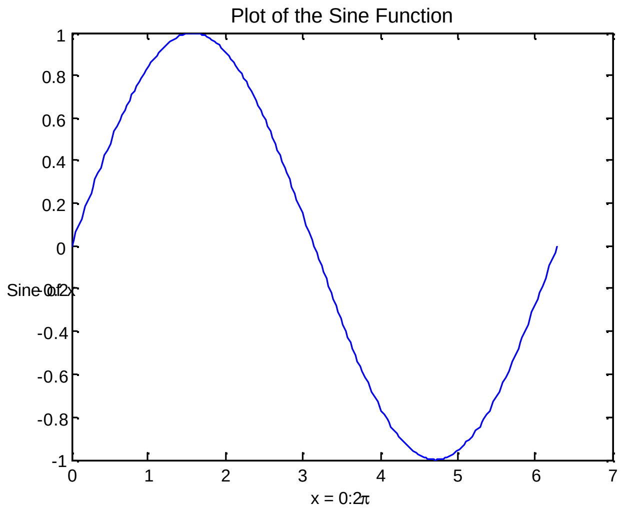
The plot function has different forms, depending on the input arguments. If `y` is a vector, `plot(y)` produces a piecewise linear graph of the elements of `y` versus the index of the elements of `y`. If you specify two vectors as arguments, `plot(x,y)` produces a graph of `y` versus `x`.

For example, these statements use the colon operator to create a vector of `x` values ranging from zero to 2, compute the sine of these values, and plot the result.

```
>>x = 0:pi/100:2*pi;
>>y = sin(x);
>>plot(x,y)
```

Now label the axes and add a title. The characters `\pi` create the symbol π.

```
>>xlabel('x = 0:2\pi')
>>ylabel('Sine of x')
>>title('Plot of the Sine Function','FontSize',12)
```



Multiple Data Sets in One Graph

Multiple *x*-*y* pair arguments create multiple graphs with a single call to `plot`. MATLAB automatically cycles through a predefined (but user settable) list of colors to allow discrimination between each set of data. For example, these statements plot three related functions of `x`, each curve in a separate distinguishing color.

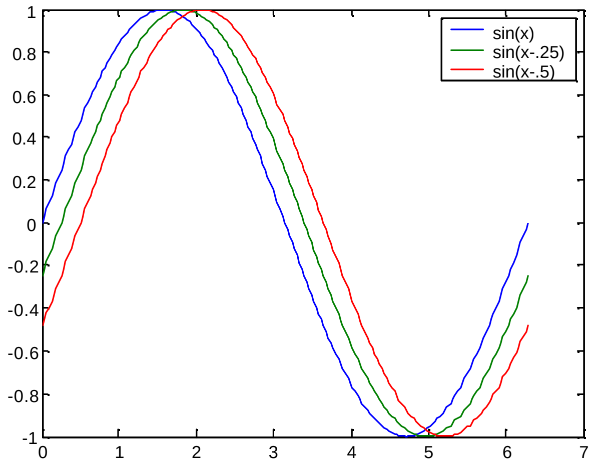
```
>>x = 0:pi/100:2*pi;
>>y2 = sin(x-.25);
>>y3 = sin(x-.5);
```

[1] Based on excerpts from the MATLAB online help feature.

```
>>plot(x,y,x,y2,x,y3)
```

The legend command provides an easy way to identify the individual plots.

```
>>legend('sin(x)','sin(x-.25)','sin(x-.5)')
```

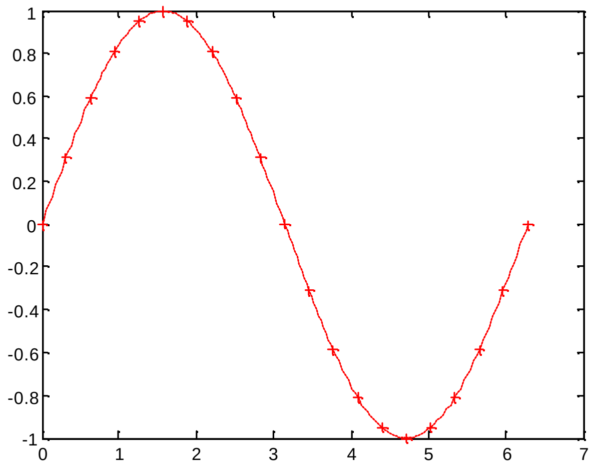


Plotting Lines and Markers

If you specify a marker type but not a linestyle, MATLAB draws only the marker. You may also want to use fewer data points to plot the markers than you use to plot the lines.

For example,

```
>>x1 = 0:pi/100:2*pi;
>>x2 = 0:pi/10:2*pi;
>>plot(x1,sin(x1),'r:',x2,sin(x2),'r+')
```



This plot shows the data twice using a different number of points for the dotted line and marker plots.

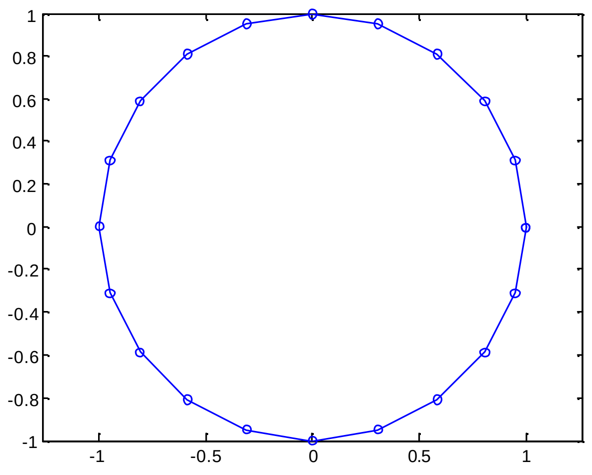
Imaginary and Complex Data

When the arguments to plot are complex, the imaginary part is ignored except when plot is given a single complex argument. For this special case, the command is a shortcut for a plot of the real part versus the imaginary part. Therefore, `plot(Z)` where `Z` is a complex vector or matrix, is equivalent to `plot(real(Z),imag(Z))`

For example,

```
>>t = 0:pi/10:2*pi;
>>plot(exp(i*t),'-o')
>>axis equal
```

draws a 20-sided polygon with little circles at the vertices. The command, axis equal, makes the individual tick mark increments on the *x*- and *y*-axes the same length, which makes this plot more circular in appearance.

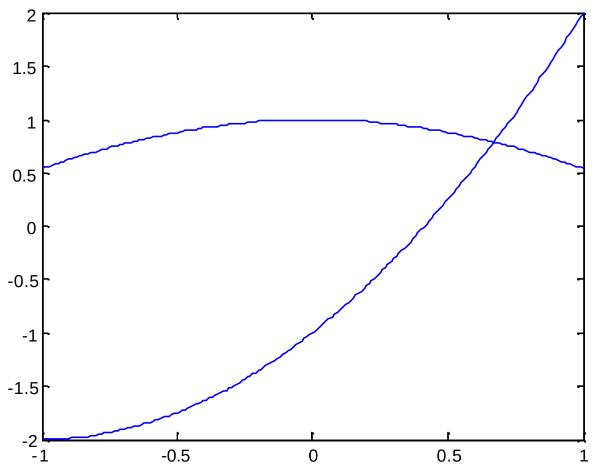


Adding Plots to an Existing Graph

The `hold` command enables you to add plots to an existing graph. When you type `hold on` MATLAB does not replace the existing graph when you issue another plotting command; it adds the new data to the current graph, rescaling the axes if necessary.

For example,

```
>>t=-1:0.01:1;
>>y=t.^2+2*t-1;
>>plot(t,y)
>>hold
Current plot held
>>w=cos(t);
>>plot(t,w)
```

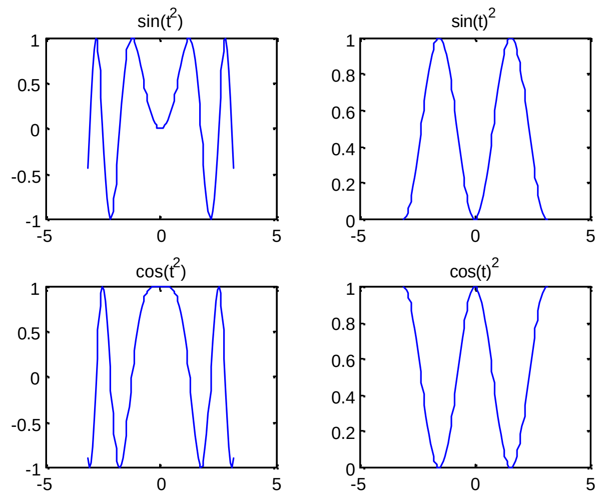


The `hold on` command causes the color of the plot to match with the existing plot in the figure.

Multiple Plots in One Figure

The `subplot` command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing `subplot(m,n,p)` partitions the figure window into an m-by-n matrix of small subplots and selects the p^th subplot for the current plot. The plots are numbered along first the top row of the figure window, then the second row, and so on. For example, these statements plot graphs in four different subregions of the figure window.

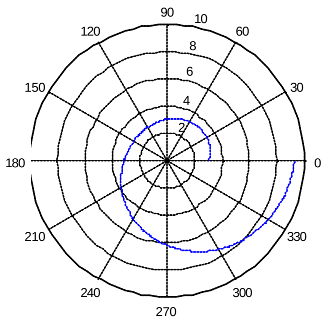
```
>>t = -pi:2*pi/100:pi;
>>f1=sin(t.^2);
>>f2=(sin(t)).^2;
>>f3=cos(t.^2);
>>f4=(cos(t)).^2;
>>subplot(2,2,1);plot(t,f1);
>>title('sin(t^2)')
>>subplot(2,2,2);plot(t,f2);
>>title('sin(t)^2')
>>subplot(2,2,3);plot(t,f3);
>>title('cos(t^2)')
>>subplot(2,2,4);plot(t,f4);
>>title('cos(t)^2')
```



Polar and logarithmic plots

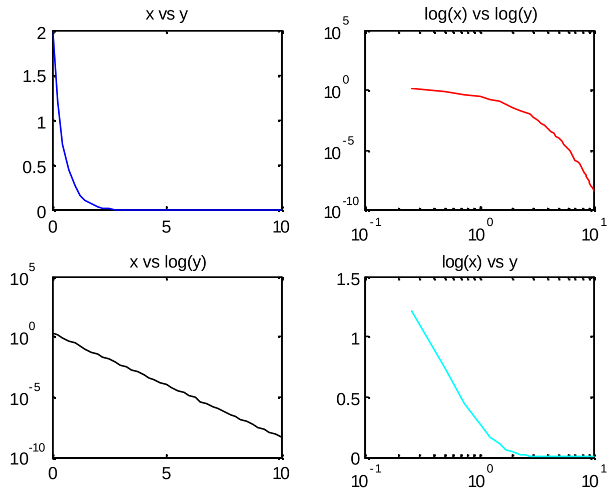
To obtain a plot in polar coordinates, we use the command polar. For example, to plot the function $r = 3\cos^2(q/2)+q$, $0 \le q \le 2p$, we type

```
>> t=0:0.01:2*pi;
>> r=3*cos(t/2).^2+t;
>> polar(t,r)
```



Many science and engineering applications require plots in which one or both axes have a logarithmic scale. The commands for these are `loglog`, `semilogy` and `semilogx`. For example,

```
>> x=0:.25:10;
>> y=2*exp(-2*x);
>> subplot(2,2,1)
>> plot(x,y,'b')
>> title('x vs y')
>> subplot(2,2,2)
>> loglog(x,y,'r')
>> title('log(x) vs log(y)')
>> subplot(2,2,3)
>> semilogy(x,y,'k')
>> title('x vs log(y)')
>> subplot(2,2,4)
>> semilogx(x,y,'c')
>> title('log(x) vs y')
```



Controlling the Axes

The `axis` command supports a number of options for setting the scaling, orientation, and aspect ratio of plots. You can also set these options interactively. Type `help axis` for more information.

By default, MATLAB finds the maxima and minima of the data to choose the axis limits to span this range. The axis command enables you to specify your own limits: `axis([xmin, xmax, ymin, ymax])`, or for three-dimensional graphs, `axis([xmin, xmax, ymin, ymax, zmin, zmax])`

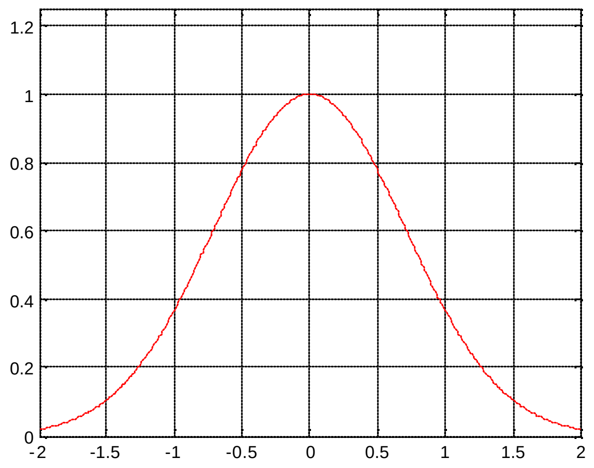
Use the command `axis auto` to re-enable MATLAB's automatic limit selection. `axis` also enables you to specify a number of predefined modes. For example, `axis square` makes the x-axes and y-axes the same length; `axis equal` makes the individual tick mark increments on the *x*- and *y*-axes the same length. `axis auto normal` returns the axis scaling to its default, automatic mode.

You can use the `axis` command to make the axis visible or invisible. `axis on` makes the axis visible. This is the default. `axis off` makes the axis invisible. The `grid` command toggles grid lines on and off. The statement `grid on` turns the grid lines on and `grid off` turns them back off again.

For example,

```
>>x=-2:0.01:2;
>>plot(x,exp(-x.^2),'r')
>>grid on
```

```
>>axis([-2 2 0 1.25])
```



Saving a Figure

To save a figure, select Save from the File menu. To save it using a graphics format, such as JPEG, for use with other applications, select Export from the File menu. You can also save from the command line - use the `saveas` command, including any options to save the figure in a different format.

Using Plot Editing Mode

The MATLAB figure window supports a point-and-click style editing mode that you can use to customize the appearance of your graph. In plot editing mode, you can use a graphical user interface, called the Property Editor, to edit the properties of objects in the graph. The Property Editor provides access to many properties of the root, figure, axes, line, light, patch, image, surfaces rectangle, and text objects. For example, using the Property Editor, you can change the thickness of a line, add titles and axes labels, add lights, and perform many other plot editing tasks.

Printing Graphics

You can print a MATLAB figure directly on a printer connected to your computer or you can export the figure to one of the standard graphic file formats supported by MATLAB. There are two ways to print and export figures: Using the Print option under the File menu, and using the `print` command.

Printing from the Menu – There are four menu options under the File menu that pertain to printing:

The Page Setup option displays a dialog box that enables you to adjust characteristics of the figure on the printed page. The Print Setup option displays a dialog box that sets printing defaults, but does not actually print the figure. The Print Preview option enables you to view the figure the way it will look on the printed page. The Print option displays a dialog box that lets you select standard printing options and print the figure.

Generally, use Print Preview to determine whether the printed output is what you want. If not, use the Page Setup dialog box to change the output settings. Select the Page Setup dialog box Help button to display information on how to set up the page.

Exporting Figure to Graphics Files – The Export option under the File menu enables you to export the figure to a variety of standard graphics file formats (such as JPEF, EPS, etc.).

Using the `print` Command – The `print` command provides more flexibility in the type of output sent to the printer and allows you to control printing from M-files. The result can be sent directly to your default printer or stored in a specified file. A wide variety of output formats, including TIFF, JPEG, and PostScript, is available.

For example, this statement saves the contents of the current figure window as color Encapsulated Level 2 PostScript in the file called magicsquare.eps. It also includes a TIFF preview, which enables most word processors to display the picture

```
>>print -depsc2 -tiff magicsquare.eps
```

To save the same figure as a TIFF file with a resolution of 200 dpi, use the command

```
>>print -dtiff -r200 magicsquare.tiff
```

If you type print on the command line, `print` MATLAB prints the current figure on your default printer.

2. MESH AND SURFACE PLOTS

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

Visualizing Functions of Two Variables

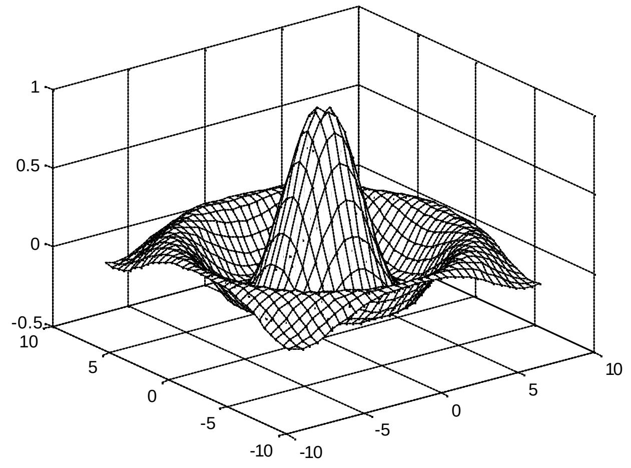
To display a function of two variables, $z = f(x,y)$:

Generate `X` and `Y` matrices consisting of repeated rows and columns, respectively, over the domain of the function. Use `X` and `Y` to evaluate and graph the function.

The `meshgrid` function transforms the domain specified by a single vector or two vectors x and y into matrices `X` and `Y` for use in evaluating functions of two variables. The rows of `X` are copies of the vector *x* and the columns of `Y` are copies of the vector *y*.

The next example evaluates and graphs the two-dimensional *sinc* function, sin(*r*)/*r*, between the *x* and *y* directions. *R* is the distance from origin, which is at the center of the matrix.

```
>>[X,Y] = meshgrid(-8:.5:8);
>>R = sqrt(X.^2 + Y.^2);
>>Z = sin(R)./R;
>>mesh(X,Y,Z,'EdgeColor','black')
```



3. SPECIALIZED GRAPHICS

MATLAB supports a variety of graph types that enable you to present information effectively. The type of graph you select depends, to a large extent, on the nature of your data. The following list can help you select the appropriate graph:

- Bar and area graphs are useful to view results over time, comparing results, and displaying individual contribution to a total amount.
- Pie charts show individual contribution to a total amount.
- Histograms show the distribution of data values.
- Stem and stairstep plots display discrete data.
- Compass, feather, and quiver plots display direction and velocity vectors.
- Contour plots show equivalued regions in data.

Bar and area graphs

Bar and area graphs display vector or matrix data. These types of graphs are useful for viewing results over a period of time, comparing results from different datasets, and showing how individual elements contribute to an aggregate amount. Bar graphs are suitable for displaying discrete data, whereas area graphs are more suitable for displaying continuous data.

`bar`: Displays columns of m-by-n matrix as *m* groups of *n* vertical bars

`barh`: Displays columns of m-by-n matrix as *m* groups of *n* horizontal bars

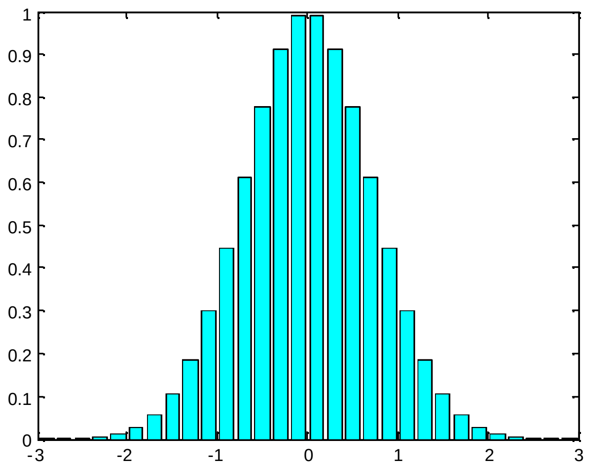
`bar3`: Displays columns of m-by-n matrix as *m* groups of *n* vertical 3-D bars

`bar3h`: Displays columns of m-by-n matrix as *m* groups of *n* horizontal 3-D bars

`area`: Displays vector data as stacked area plots

For example,

```
>> x = -2.9:0.2:2.9;
>>bar(x,exp(-x.^2))
>>colormap cool
```



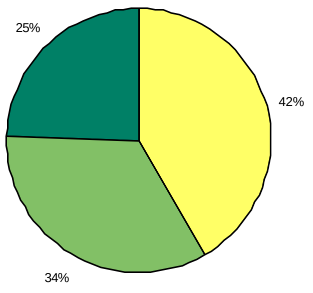
Pie charts display the percentage that each element in a vector or matrix contributes to the sum of all elements. `pie` and `pie3` create 2-D and 3-D pie charts.

Here is an example using the `pie` function to visualize the contribution that three products make to total sales. Given a matrix `X` where each column of `X` contains yearly sales figures for a specific product over a five-year period,

```
>>X = [19.3 22.1 51.6; 34.2 70.3 82.4; 61.4 82.9 90.8; 50.5 54.9 59.1; 29.4
36.3 47.0];
```

Sum each row in `X` to calculate total sales for each product over the five-year period.

```
>>x = sum(X);
>>pie(x)
>>colormap summer
```

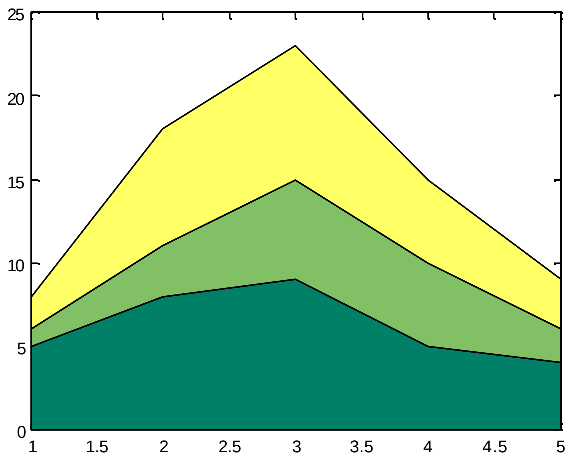


Area graphs are useful for showing how elements in a vector or matrix contribute to the sum of all elements at a particular *x* location. By default, area accumulates all values from each row in a matrix and creates a curve from those values. Using this matrix,

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

the statement,

```
>>area(Y)
```



displays a graph containing three area graphs, one per column. The height of the area graph is the sum of the elements in each row. Each successive curve uses the preceding curve as its base.

Histograms

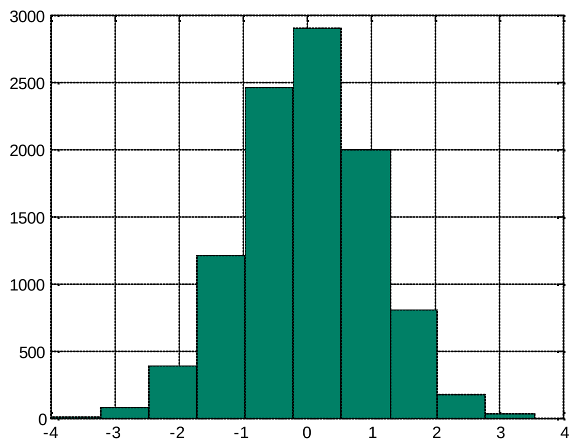
MATLAB's histogram functions show the distribution of data values. The functions that create histograms are `hist` and `rose`.

`hist`: Displays data in a Cartesian coordinate system
`rose`: Displays data in a polar coordinate system

The histogram functions count the number of elements within a range and display each range as a rectangular bin. The height (or length when using rose) of the bins represents the number of values that fall within each range.

For example,

```
>> yn = randn(10000,1);
>>hist(yn)
>> grid
```



Discrete Data Graphs

MATLAB has a number of specialized functions that are appropriate for displaying discrete data. This section describes how to use stem plots and stairstep plots to display this type of data. (Bar charts, discussed earlier, are also suitable for displaying discrete data.)

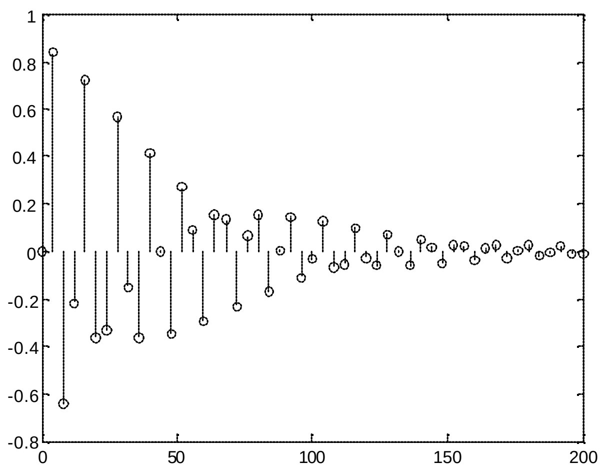
`stem`: Displays a discrete sequence of y-data as stems from x-axis

`stem3`: Displays a discrete sequence of z-data as stems from xy-plane

`stairs`: Displays a discrete sequence of y-data as steps from x-axis

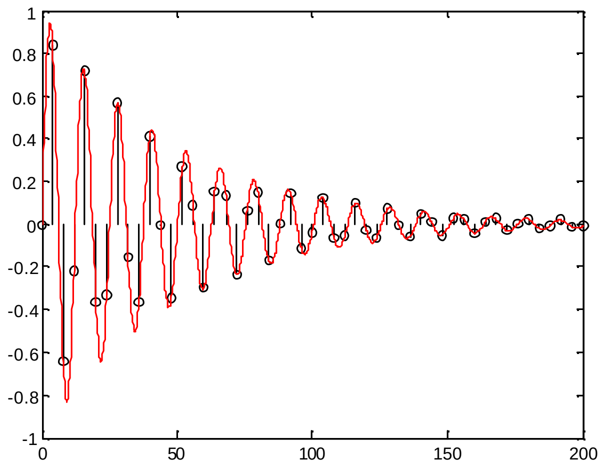
For example,

```
>>alpha = .02; beta = .5; t = 0:4:200;
>>y = exp(-alpha*t).*sin(beta*t);
>> stem(t,y,'k')
```



To see how the actual function graph "comes into play", look at:

```
>> T=0:.1:200;
>> hold
>> Y = exp(-alpha*T).*sin(beta*T);
>> plot(T,Y,'r')
```



Contour Plots

The contour functions create, display, and label isolines determined by one or more matrices.

`clabel`: Generates labels using the contour matrix and displays the labels in the current figure.

`contour`: Displays 2-D isolines generated from values given by a matrix Z.

`contour3`: Displays 3-D isolines generated from values given by a matrix Z.

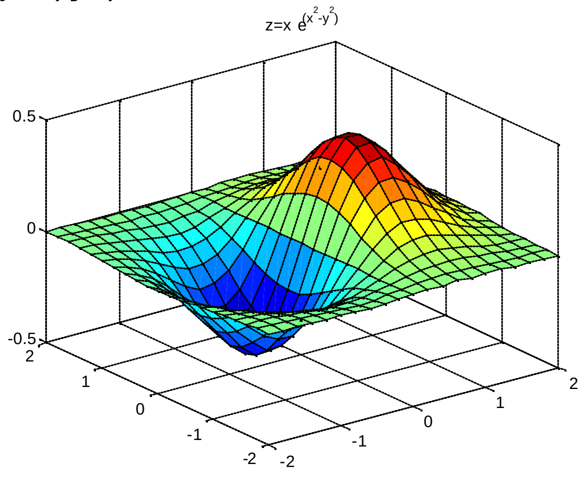
`contourf`: Displays a 2-D contour plot and fills the area between the isolines with a solid color.

`contourc`: Low-level function to calculate the contour matrix used by the other contour functions.

Two other functions also create contours. `meshc` displays a contour in addition to a mesh, and `surfc` displays a contour in addition to a surface.

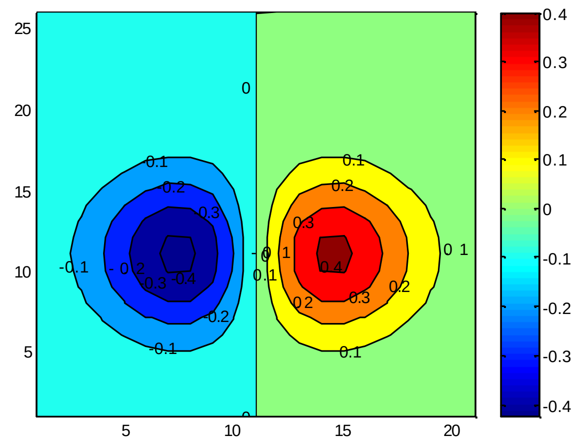
For example, consider the function $z = x\ \exp(-x^2 - y^2)$:

```
>>[X,Y] = meshgrid(-2:.2:2,-2:.2:3);
>>Z = X.*exp(-X.^2-Y.^2);
>> surf(x,y,z)
>> title('z=x e^{(x^2-y^2)}')
```



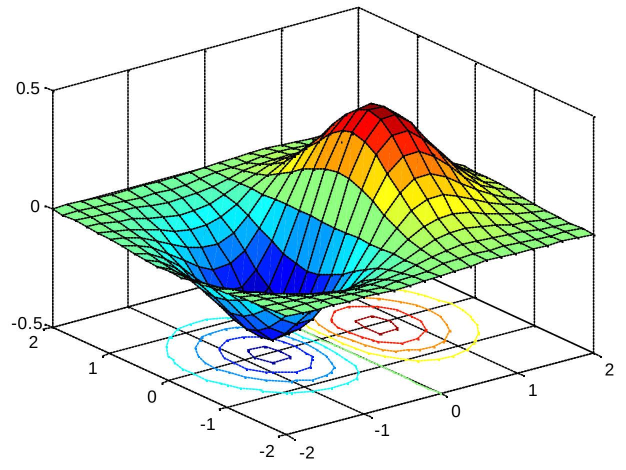
The contour plot is given by:

```
>> [C,f]=contourf(Z);
>>clabel(C,f);
>>colorbar
```



A combination of the two is given by

```
>> surfc(x,y,z)
```



4. Advanced Topics

Images

Two-dimensional arrays can be displayed as images, where the array elements determine brightness or color of the images. For example, the statements

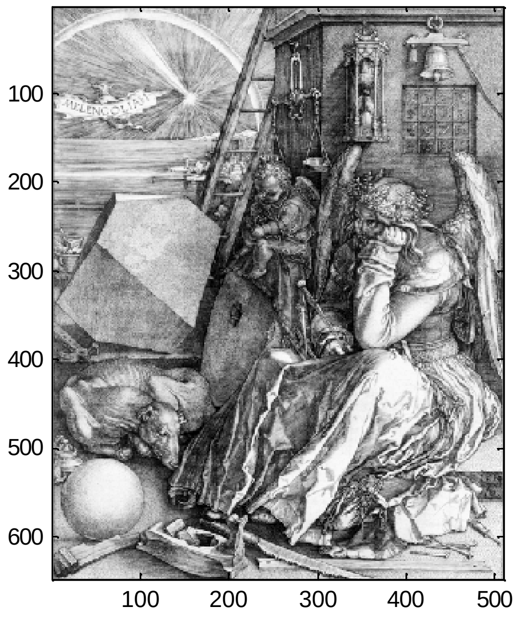
```
>>load durer
>>whos

Name            Size          Bytes    Class
X             648x509      2638656  double array
caption       2x28             112  char array
map           128x3           3072  double array
```

load the (built-in data) file `durer.mat`, adding three variables to the workspace. The matrix `X` is a 648-by-509 matrix and `map` is a 128-by-3 matrix that is the colormap for this image.

Note MAT-files, such as `durer.mat`, are binary files that can be created on one platform and later read by MATLAB on a different platform.

The elements of `X` are integers between 1 and 128, which serve as indices into the colormap, `map`. Then

```
>>image(X)
>>colormap(map)
>>axis image
```



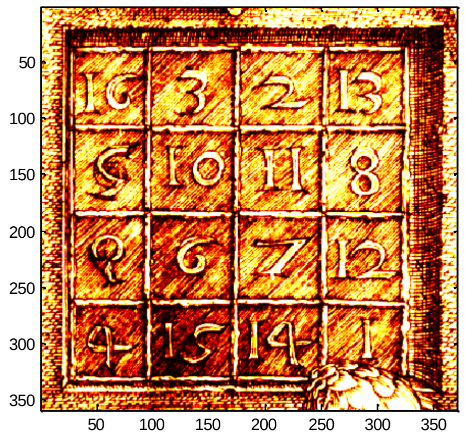
reproduces the Dürer's etching. A high resolution scan of the magic square in the upper right corner is available in another file, called `detail.mat`. Type

```
>>load detail
```

and then use the uparrow key on your keyboard to re-execute the `image`, `colormap`, and `axis` commands. The statement

```
>>colormap(hot)
```

adds some twentieth century colorization to the sixteenth century etching. The function `hot` generates a colormap containing shades of reds, oranges, and yellows. Typically a given image matrix has a specific colormap associated with it. Type `help colormap` for more information.



Animations

MATLAB provides two ways of generating moving, animated graphics. The one is by continually erasing and then redrawing the objects on the screen, making incremental changes with each redraw. (You can save a number of different pictures and then play them back as a movie.)

Here is an example showing simulated *Brownian motion*. Specify a number of points, such as

```
>>n = 20
```

and a temperature or velocity, such as

```
>>s = .02
```

The best values for these two parameters depend upon the speed of your particular computer. Generate *n* random points with (*x*,*y*) coordinates between –1/2 and +1/2.

```
>>x = rand(n,1)-0.5;
>>y = rand(n,1)-0.5;
```

Plot the points in a square with sides at –1 and +1. Save the handle for the vector of points and set its EraseMode to xor. This tells the MATLAB graphics system not to redraw the entire plot when the coordinates of one point are changed, but to restore the background color in the vicinity of the point using an "exclusive or" operation.

```
>>h = plot(x,y,'.');
>>axis([-1 1 -1 1])
>>axis square
>>grid off
>>set(h,'EraseMode','xor','MarkerSize',18)
```

Now begin the animation. Here is an infinite while-loop, which you can eventually exit by typing Ctrl+c. Each time through the loop, add a small amount of normally distributed random noise to the coordinates of the points. Then, instead of creating an entirely new plot, simply change the XData and YData properties of the original plot.

```
>>while 1
   drawnow
   x = x + s*randn(n,1);
   y = y + s*randn(n,1);
   set(h,'XData',x,'YData',y)
end
```

Creating Movies

If you increase the number of points in the Brownian motion example to something like *n* = 300 and *s* = .02, the motion is no longer very fluid; it takes too much time to draw each time step. It becomes more effective to save a predetermined number of frames as bitmaps and to play them back as a movie.

First, decide on the number of frames, say

```
>>nframes = 50;
```

Next, set up the first plot as before, except using the default EraseMode (normal).

```
>>x = rand(n,1)-0.5;
>>y = rand(n,1)-0.5;
>>h = plot(x,y,'.');
>>set(h,'MarkerSize',18);
```

```
>>axis([-1 1 -1 1])
>>axis square
>>grid off
```

Generate the movie and use `getframe` to capture each frame.

```
>>for k = 1:nframes
   x = x + s*randn(n,1);
   y = y + s*randn(n,1);
   set(h,'XData',x,'YData',y)
   M(k) = getframe;
end
```

Finally, play the movie 30 times.

```
>>movie(M,30)
```

We have only touched the surface here … For more information on ALL of MATLAB's graphics capabilities see the help menu in the MATLAB window.