Graphics in MATLAB

Responsible teacher: Anatoliy Malyarenko

November 10, 2003

Abstract

Contents of the lecture:

☞ Two-dimensional graphics.
☞ Formatting graphs.
☞ Three-dimensional graphics.
☞ Specialised plots.

Basic Plotting Commands

MATLAB provides a variety of functions for displaying vector data as line plots, as well as functions for annotating and printing these graphs. The following table summarises the functions that produce basic line plots. These functions differ in the way they scale the plot's axes. Each accepts input in the form of vectors or matrices and automatically scales the axes to accommodate the data.

Function	**Description**
`plot`	Graph 2-D data with linear scales for both axes
`plot3`	Graph 3-D data with linear scales for both axes
`loglog`	Graph with logarithmic scales for both axes
`semilogx`	Graph with a logarithmic scale for the x-axis and a linear scale for the y-axis
`semilogy`	Graph with a logarithmic scale for the y-axis and a linear scale for the x-axis
`plotyy`	Graph with y-tick labels on the left and right side

Creating Line Plots

The plot function has different forms depending on the input arguments. For example, if `y` is a vector, `plot(y)` produces a 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 create a vector of values in the range $[0, 2\pi]$ in increments of $\pi/100$ and then use this vector to evaluate the sine function over that range. MATLAB plots the vector on the x-axis and the value of the sine function on the y-axis.

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
grid on
```

MATLAB automatically selects appropriate axis ranges and tick mark locations (Figure 1).

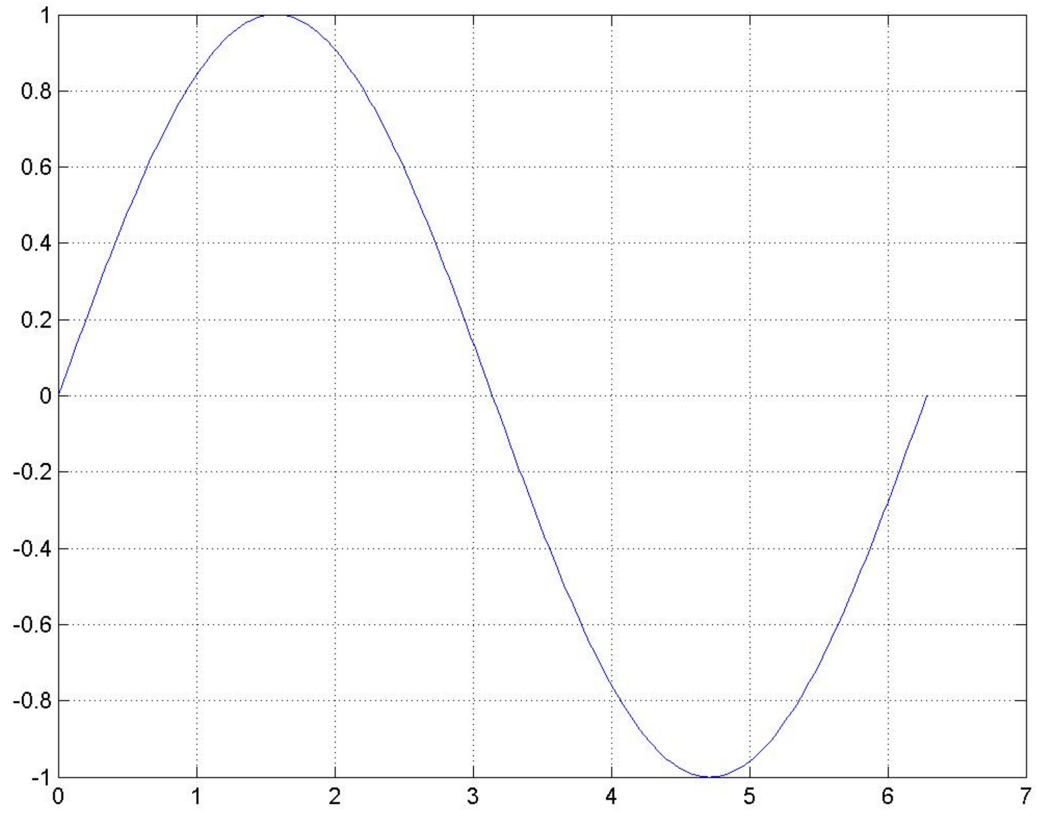

Figure 1: $y = \sin x$, $0 \le x \le 2\pi$

You can plot multiple graphs in one call to plot using `x-y` pairs. MATLAB automatically cycles through a predefined list of colours to allow discrimination between each set of data. Plotting three curves as a function of `t` produces

```
y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,t,y2,t,y3)
```

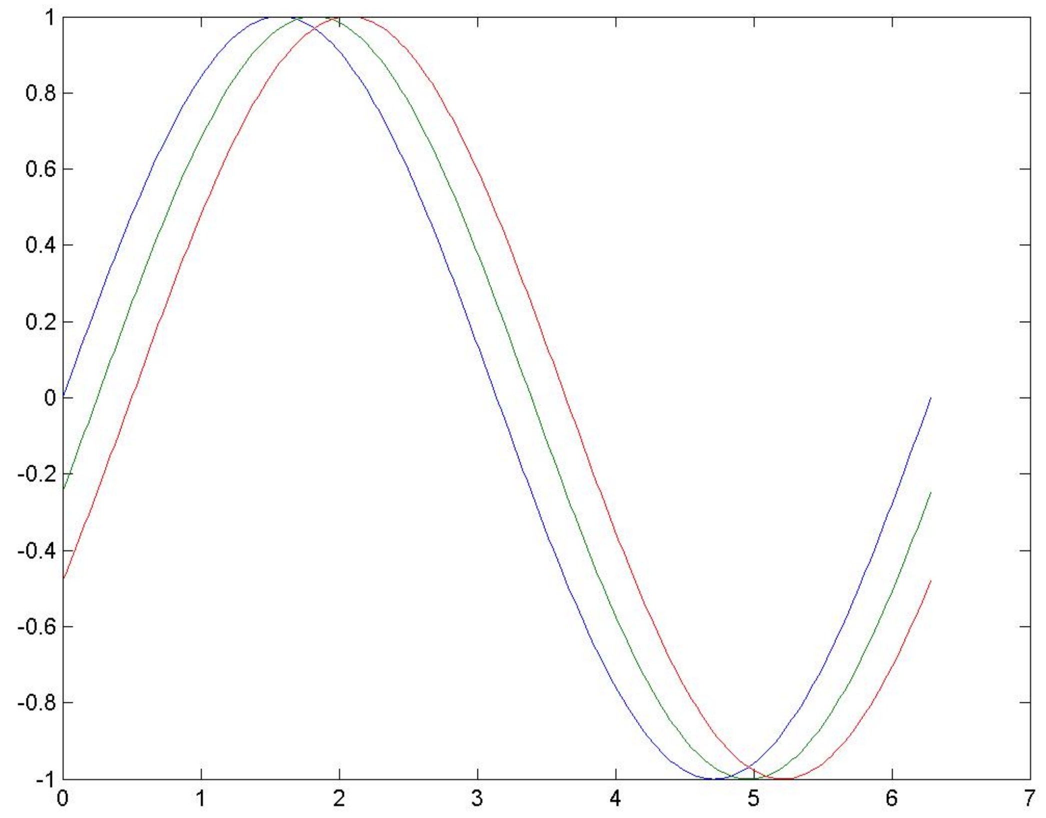

Figure 2: Multiple graphs in one plot

Specifying Line Style

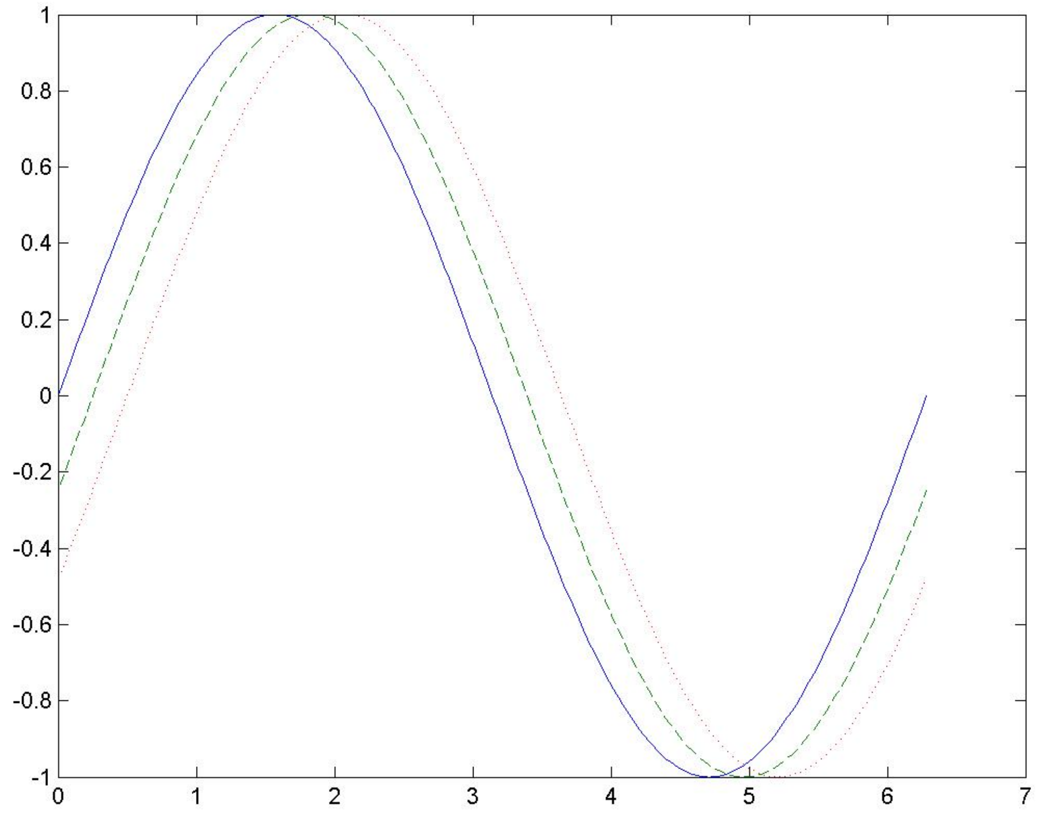

Figure 3: Different line styles

You can assign different line styles to each data set by passing line style identifier strings to `plot`. For example,

```
t = 0:pi/100:2*pi; y = sin(t); y2 = sin(t-0.25);
y3 = sin(t-0.5);
plot(t,y,'-',t,y2,'--',t,y3,':')
```

Specifying the Colour and Size of Lines

You can control a number of line style characteristics by specifying values for line properties:

☞ `LineWidth` — specifies the width of the line in units of points.

- ☞ `MarkerEdgeColor` — specifies the colour of the marker or the edge colour for filled markers (circle, square, diamond, pentagram, hexagram, and the four triangles).

- ☞ `MarkerFaceColor` — specifies the colour of the face of filled markers.

- ☞ `MarkerSize` — specifies the size of the marker in units of points.

For example, these statements,

```
x = -pi:pi/10:pi; y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',10)
```

produce a graph with:

- ☞ A red dashed line with square markers.

- ☞ A line width of two points.

- ☞ The edge of the marker coloured black.

- ☞ The face of the marker coloured green.

- ☞ The size of the marker set to 10 points.

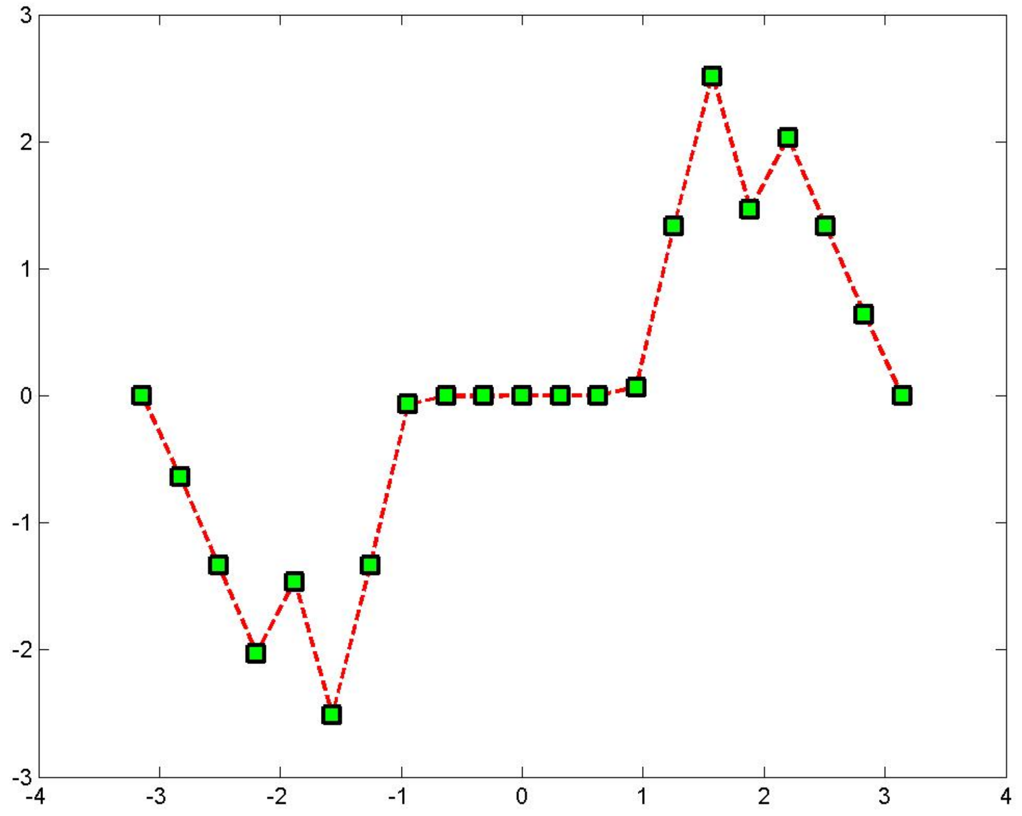

Figure 4: Lines of different styles

Adding Plots to an Existing Graph

You can add plots to an existing graph using the `hold` command. When you set `hold` to `on`, MATLAB does not remove the existing graph; it adds the new data to the current graph, rescaling if the new data falls outside the range of the previous axis limits.

For example, these statements first create a semilogarithmic plot, then add a linear plot.

```
semilogx(1:100,'+')
hold on
plot(1:3:300,1:100,'--')
hold off
```

While MATLAB resets the x-axis limits to accommodate the new data, it does not change the scaling from logarithmic to linear.

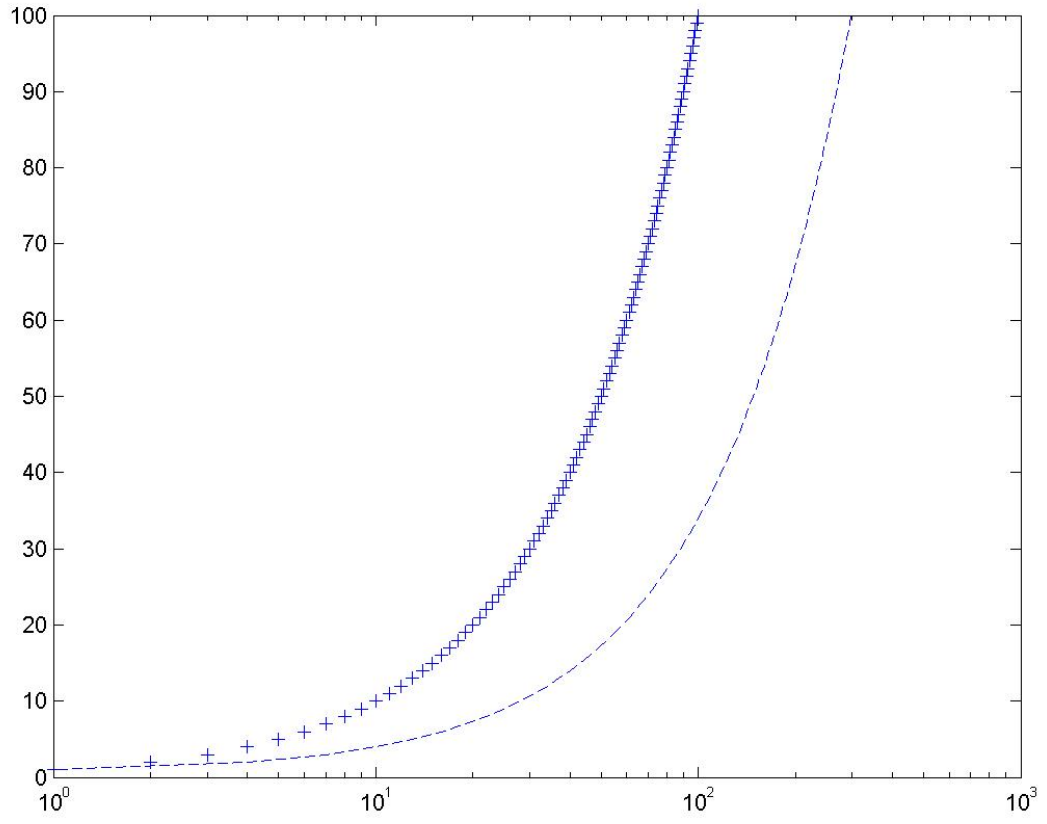

Figure 5: Adding plots to an existing graph

Line Plots of Matrix Data

When you call the `plot` function with a single matrix argument

```
plot(Y)
```

MATLAB draws one line for each column of the matrix. The x-axis is labeled with the row index vector, `1:m`, where `m` is the number of rows in `Y`. For example,

```
Z=peaks;
```

returns a 49-by-49 matrix obtained by evaluating a function of two variables. Plotting this matrix

```
plot(Z)
```

produces a graph with 49 lines.

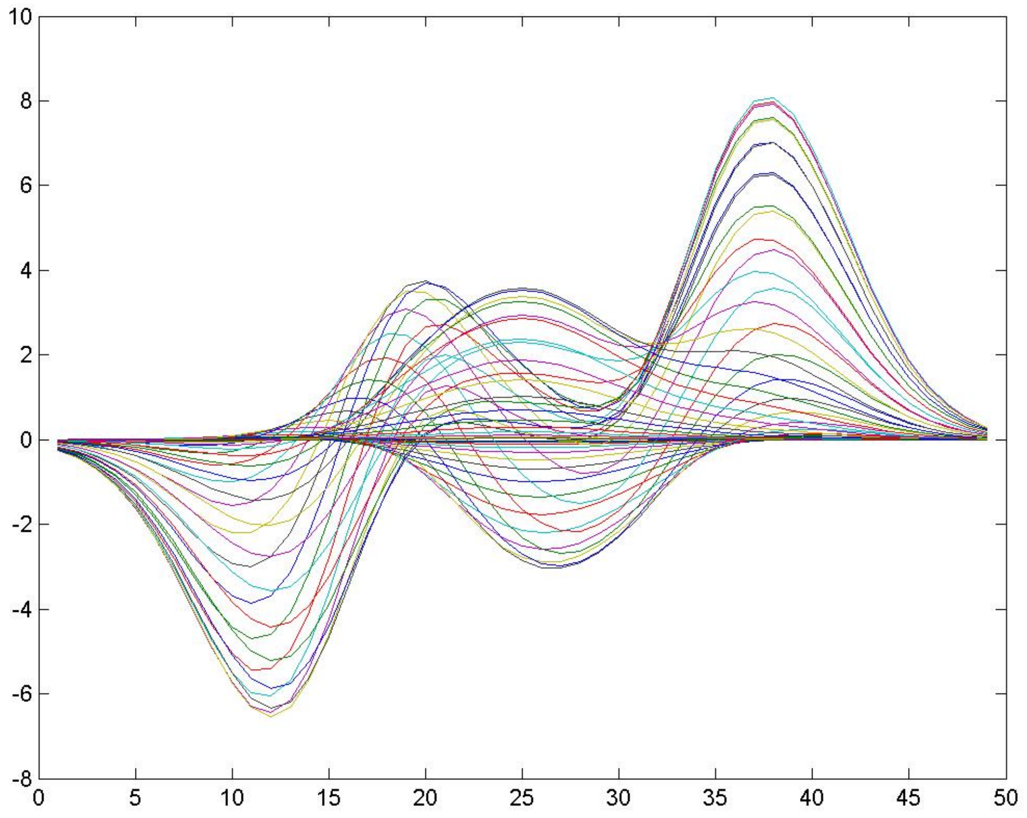

Figure 6: Line plot of matrix data

Formatting graphs

When creating presentation graphics, you may want to add labels and annotations to your graph to help explain your data. For example

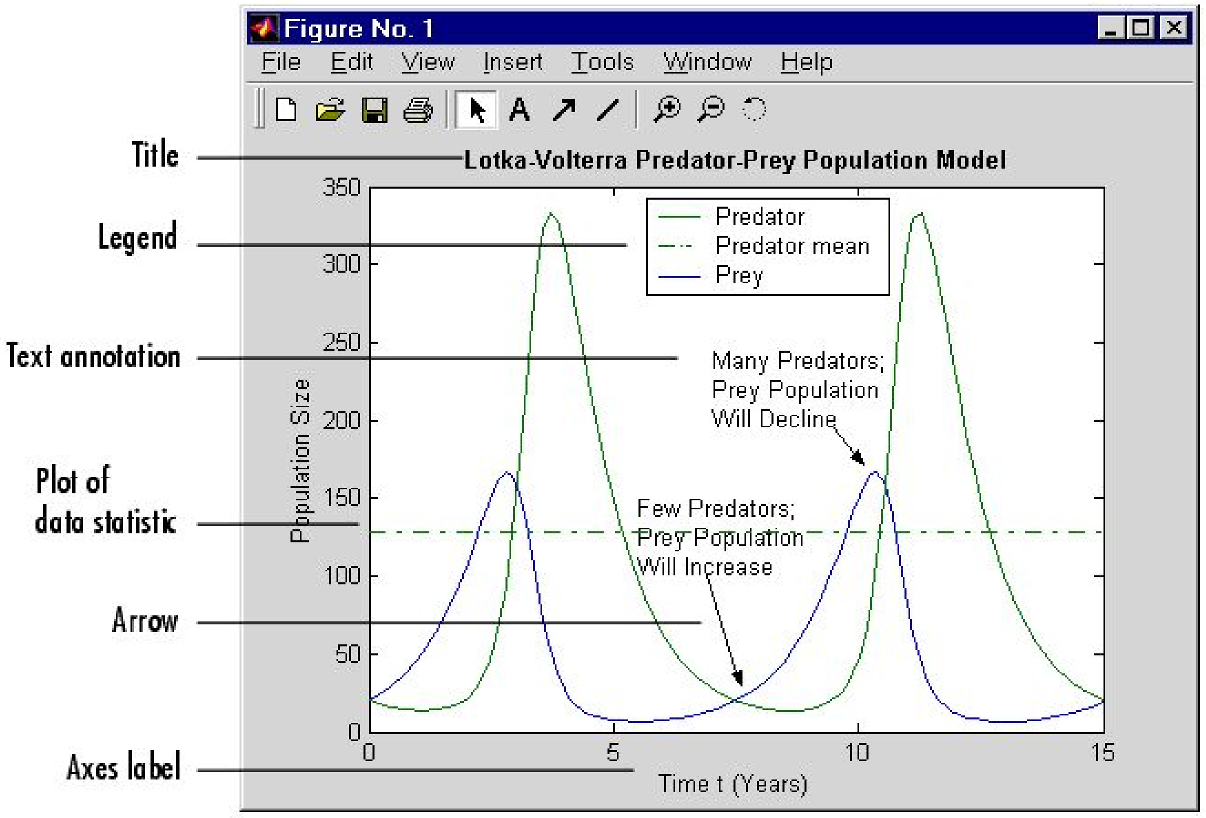
Figure No. 1
File Edit View Insert Tools Window Help
Title
Legend
Text annotation
Plot of data statistic
Arrow
Axes label
Lotka-Volterra Predator-Prey Population Model
Predator
Predator mean
Prey
Many Predators; Prey Population Will Decline
Few Predators; Prey Population Will Increase
Population Size
Time t (Years)
0
50
100
150
200
250
300
350
0
5
10
15

Figure 7: Presentation graphics

The `title` function

To add a title to a graph at the MATLAB command prompt or from an M-file, use the title function. The title function lets you specify the value of title properties at the time you create it.

For example, the following code adds a title to the current axes and sets the value of the `FontWeight` property to bold.

```
title('Lotka-Volterra Predator-Prey Population Model',...
  'FontWeight','bold')
```

The `legend` function

To add a legend to a graph at the MATLAB command prompt or from an M-file, use the `legend` function. You must specify the text labels when you create a legend using the `legend` function.

For example, the following code adds a legend to the current axes.

```
legend('Y1 Predator','Y2 Prey')
```

The `legend` function lets you specify many other aspects of the legend, such as its position. For more information, see the `legend` function reference information.

Using Axis-Label Commands

You can add x-, y-, and z-axis labels using the `xlabel`, `ylabel`, and `zlabel` commands. For example, these statements label the axes and add a title.

```
xlabel('t = 0 to 2\pi','FontSize',16)
ylabel('sin(t)','FontSize',16)
%
title('\it{Value of the Sine from 0 to 2\pi}',...
'FontSize',16)
```

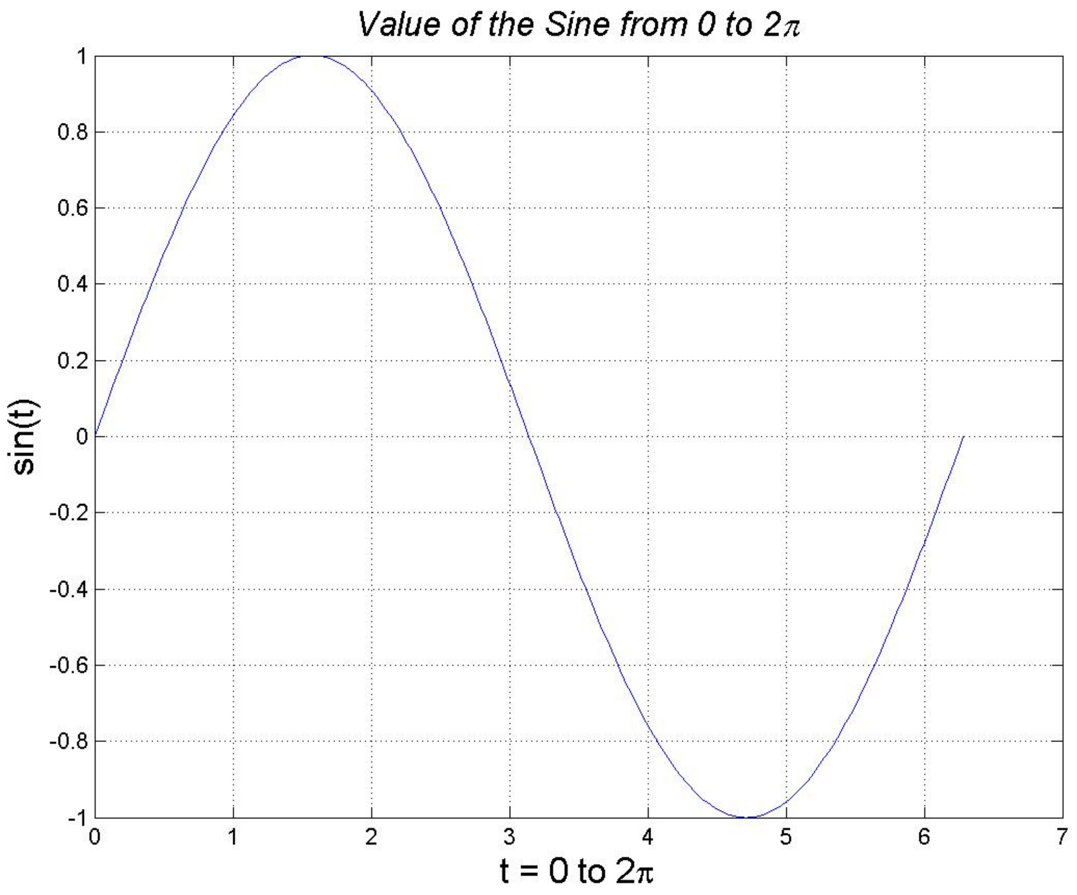

Figure 8: Axis-label commands

The labeling commands automatically position the text string appropriately. MATLAB interprets the characters immediately following the backslash \ as TEX commands. These commands draw symbols such as Greek letters and arrows.

Creating Text Annotations with the `text` Command

To create a text annotation using the text function, you must specify the the text and its location in the graph, using *x*- and *y*-coordinates. You specify the coordinates in the units of the graph.

For example, the following code creates text annotations at specific points in the Lotka–Volterra Predator–Prey Population Model graph.

```
str1(1) = {'Many Predators;'};
str1(2) = {'Prey Population'};
str1(3) = {'Will Decline'};
text(7,220,str1)

str2(1) = {'Few Predators;'};
str2(2) = {'Prey Population'};
str2(3) = {'Will Increase'};
text(5.5,125,str2)
```

Line Plots of 3-D Data

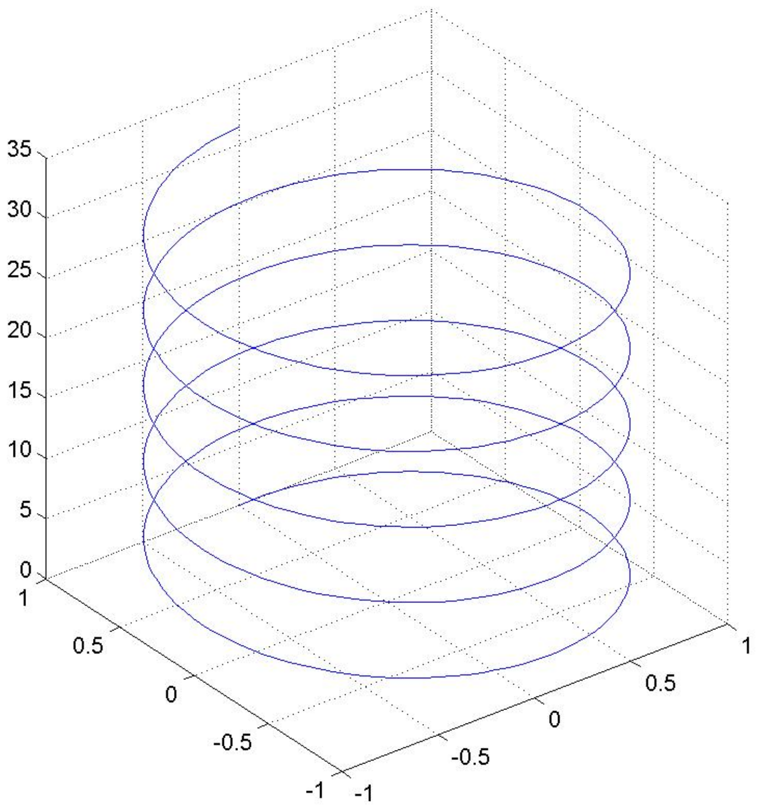

Figure 9: The helix

The 3-D analog of the `plot` function is `plot3`. If `x`, `y`, and `z` are three vectors of the same length,

```
plot3(x,y,z)
```

generates a line in 3-D through the points whose coordinates are the elements of `x`, `y`, and `z` and then produces a 2-D projection of that line on the screen. For example, these statements produce a helix.

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
axis square;
grid on
```

Representing a Matrix as a Surface

Function	Used to create
`mesh`, `surf`	Surface plot
`meshc`, `surfc`	Surface plot with contour plot beneath it
`meshz`	Surface plot with curtain plot (reference plane)
`pcolor`	Flat surface plot (value is proportional only to colour)
`surfl`	Surface plot illuminated from specified direction
`surface`	Low-level function (on which high-level functions are based) for creating surface graphics objects

MATLAB defines a surface by the z-coordinates of points above a rectangular grid in the x-y plane. The plot is formed by joining adjacent points with straight lines. Surface plots are useful for visualising matrices that are too large to display in numerical form and for graphing functions of two variables.

MATLAB can create different forms of surface plots. Mesh plots are wire-frame surfaces that colour only the lines connecting the defining points. Surface plots display both the connecting lines and the faces of the surface in colour. This table lists the various forms.

Visualising Functions of Two Variables

The first step in displaying a function of two variables, $z = f(x, y)$, is to generate `X` and `Y` matrices consisting of repeated rows and columns, respectively, over the domain of the function. Then use these matrices to evaluate and graph the function.

The `meshgrid` function transforms the domain specified by two vectors, `x` and `y`, into matrices, `X` and `Y`. You then use these matrices to evaluate 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`.

To illustrate the use of `meshgrid`, consider the function

$$z = \frac{\sin\sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2}}.$$

To evaluate this function between -8 and 8 in both x and y, you need pass only one vector argument to `meshgrid`, which is then used in both directions.

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
```

The matrix `R` contains the distance from the center of the matrix, which is the origin. Adding `eps` prevents the divide by zero (in the next step) that produces `Inf` values in the data.

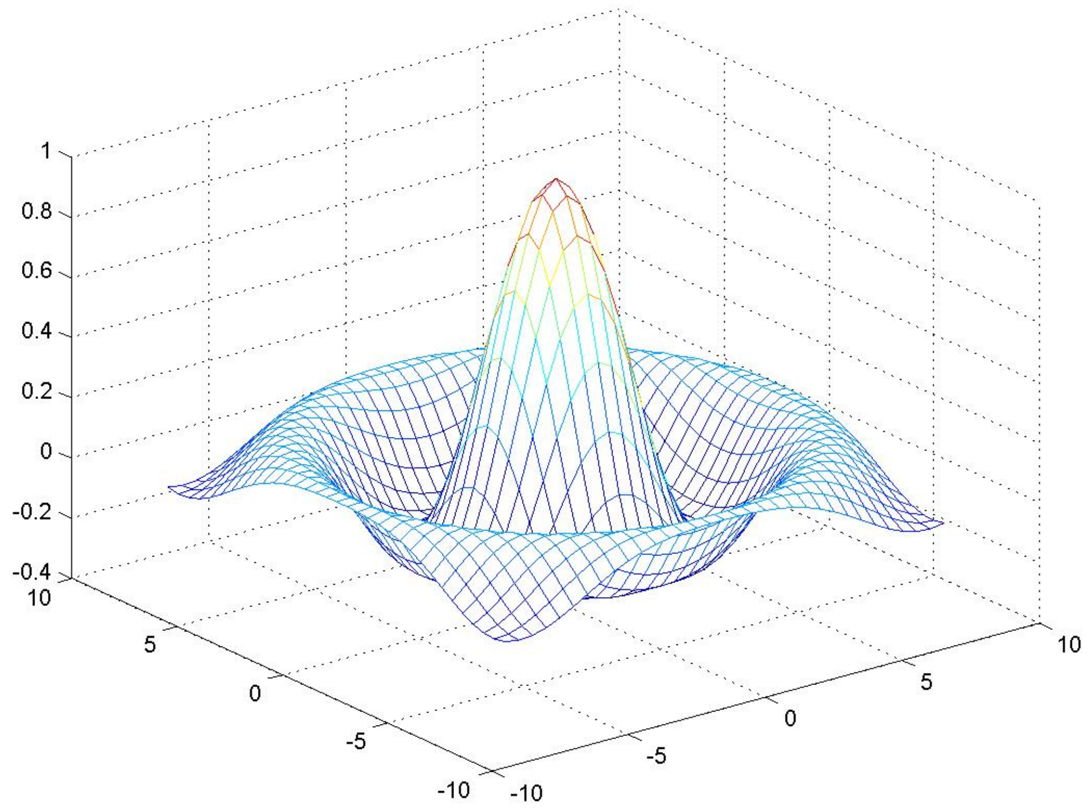

Figure 10: The first three-dimensional graph

Forming the function and plotting `Z` with mesh results in the 3-D surface.

```
Z = sin(R)./R; mesh(X,Y,Z)
```

Emphasising Surface Shape

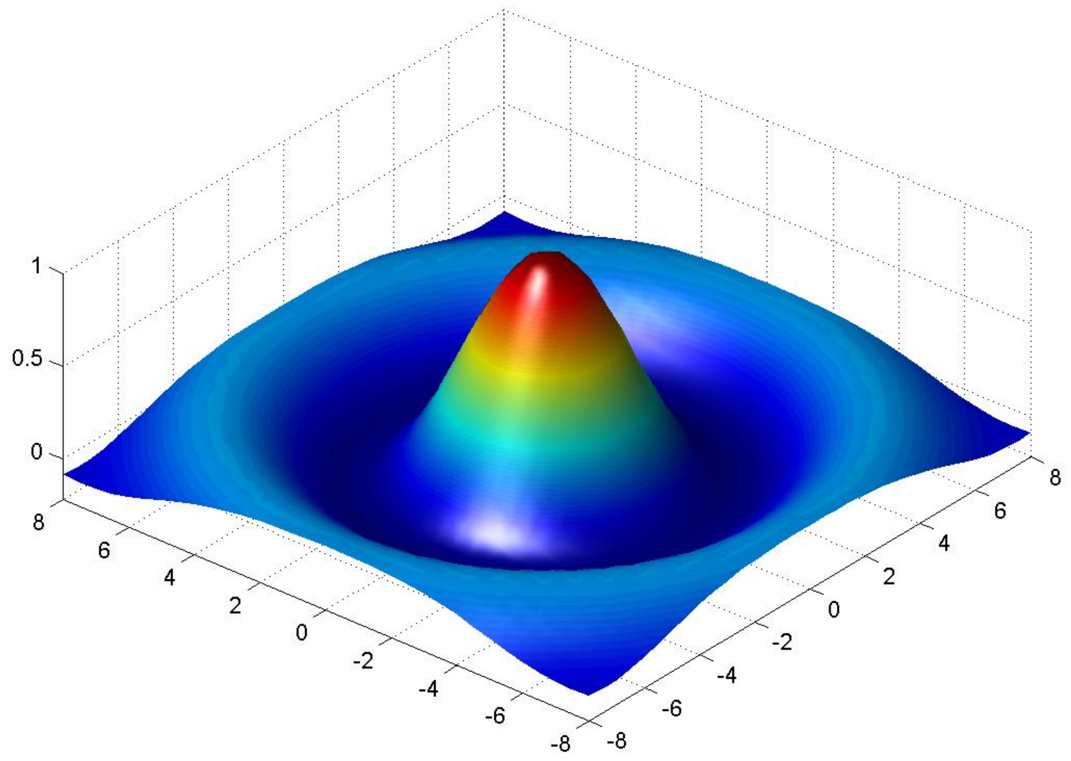

Figure 11: Emphasising surface shape

MATLAB provides a number of techniques that can enhance the information content of your graphs. For example, this graph of the next function uses the same data as the previous graph, but employs lighting and view adjustment to emphasise the shape of the graphed function (`daspect`, `axis`, `camlight`, `view`).

```
surf(X,Y,Z,'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
daspect([5 5 1])
axis tight
view(-50,30)
camlight left
```

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 data sets, 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.

Function	**Description**
`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

Grouped Bar Graph

By default, a bar graph represents each element in a matrix as one bar. Bars in a 2-D bar graph, created by the bar function, are distributed along the x-axis with each element in a column drawn at a different location. All elements in a row are clustered around the same location on the x-axis.

For example, define `Y` as a simple matrix and issue the `bar` statement in its simplest form.

```
Y = [5 2 1
     8 7 3
     9 8 6
     5 5 5
     4 3 2];
bar(Y)
```

The bars are clustered together by rows and evenly distributed along the x-axis.

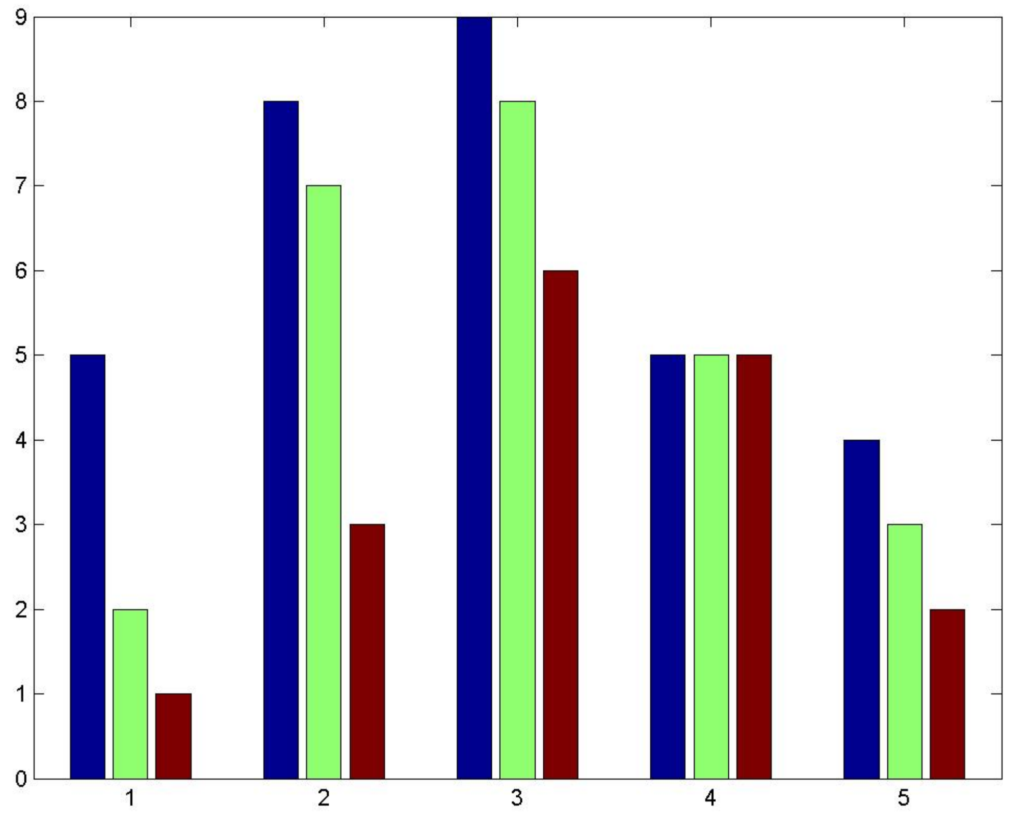

Figure 12: Grouped bar graph

Detached 3-D Bars

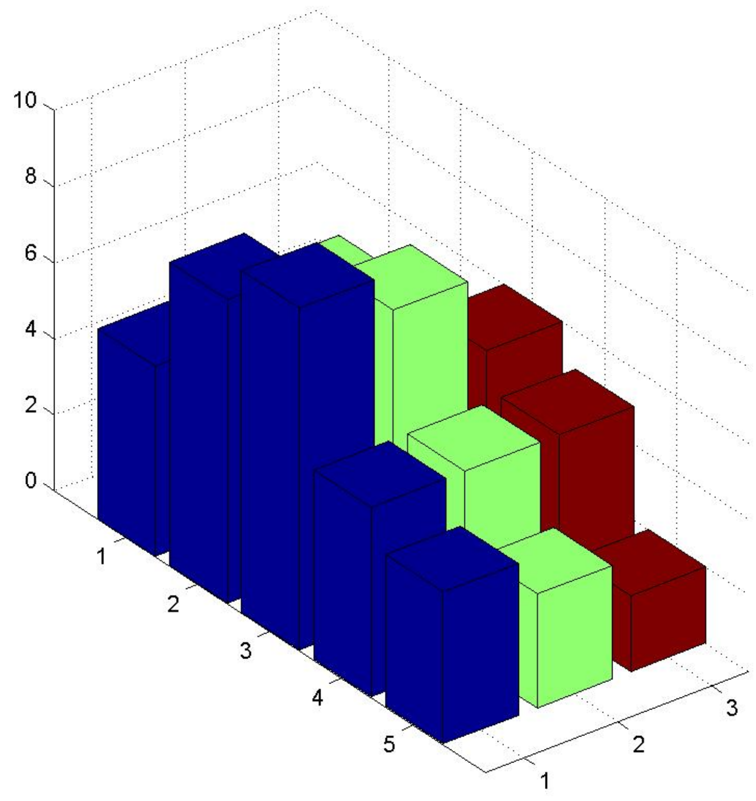

Figure 13: Detached 3-D bars

The `bar3` function, in its simplest form, draws each element as a separate 3-D block, with the elements of each column distributed along the y-axis. Bars that represent elements in the first column of the matrix are centred at 1 along the x-axis. Bars that represent elements in the last column of the matrix are centred at size `(Y,2)` along the x-axis. For example,

```
bar3(Y)
```

displays five groups of three bars along the y-axis. Notice that larger bars obscure `Y(1,2)` and `Y(1,3)`.

Grouped 3-D Bars

Cluster the bars from each row beside each other by specifying the argument '`group`'. For example,

```
bar3(Y,'group')
```

groups the bars according to row and distributes the clusters evenly along the y-axis.

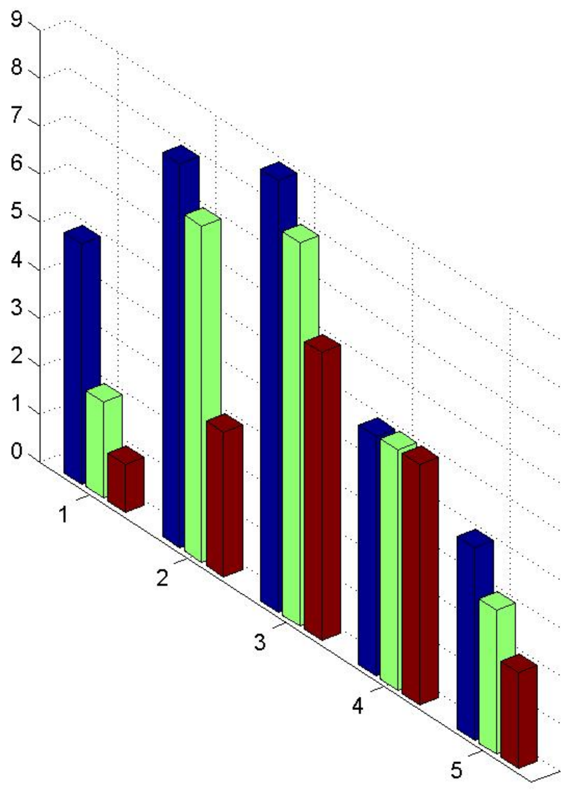

Figure 14: Grouped 3-D bars

Stacked Bar Graphs to Show Contributing Amounts

Bar graphs can show how elements in the same row of a matrix contribute to the sum of all elements in the row. These types of bar graphs are referred to as stacked bar graphs.

Stacked bar graphs display one bar per row of a matrix. The bars are divided into n segments, where n is the number of columns in the matrix. For vertical bar graphs, the height of each bar equals the sum of the elements in the row. Each segment is equal to the value of its respective element.

For example,

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

bar(Y,'stack')
grid on
set(gca,'Layer','top')
% display gridlines on top of graph
```

creates a 2-D stacked bar graph, where all elements in a row correspond to the same x location.

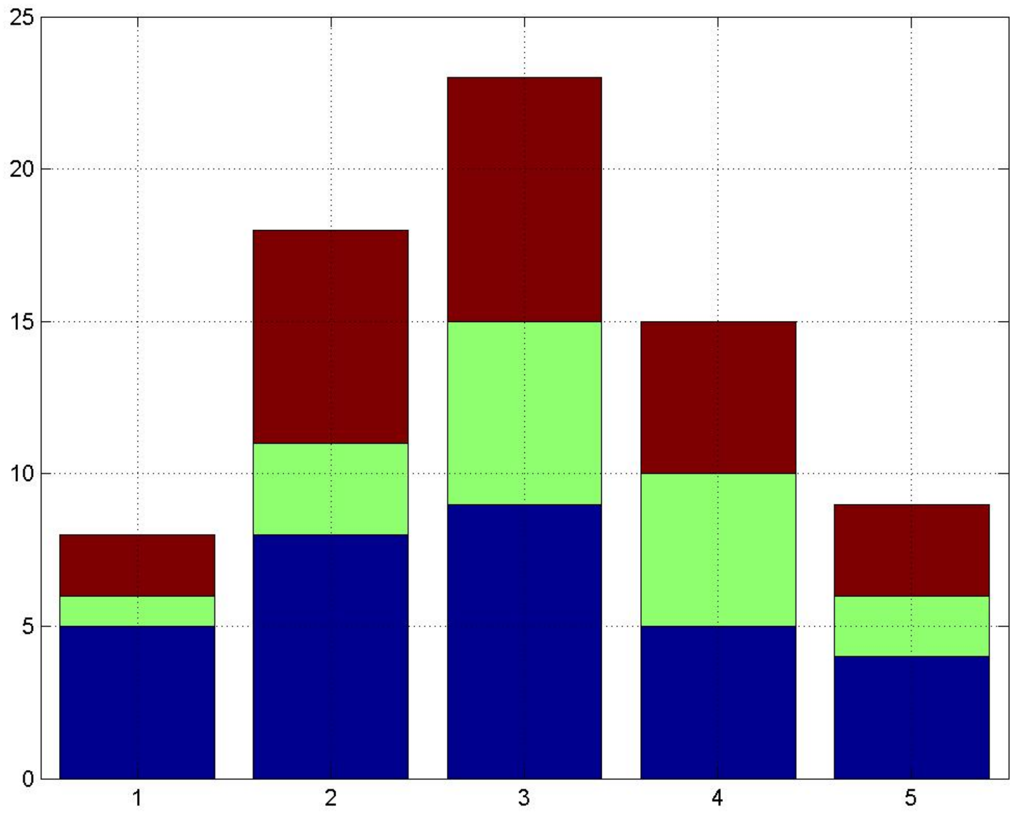

Figure 15: Stacked bar graphs to show contributing amounts

Area Graphs

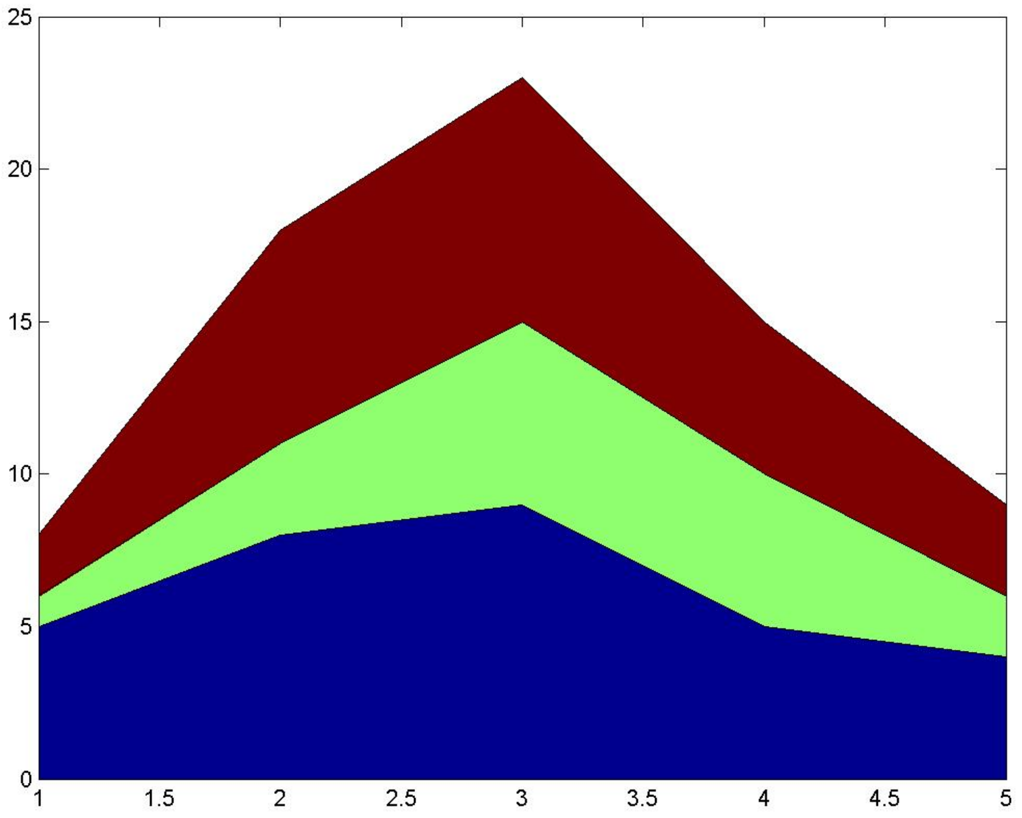

Figure 16: Area graph

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 the matrix `Y`, 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.

Example — Pie Chart

Here is an example using the `pie` function to visualise 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);
```

You can offset the slice of the pie that makes the greatest contribution using the `explode` input argument. This argument is a vector of zero and nonzero values. Nonzero values offset the respective slice from the chart.

First, create a vector containing zeros.

```
explode = zeros(size(x));
```

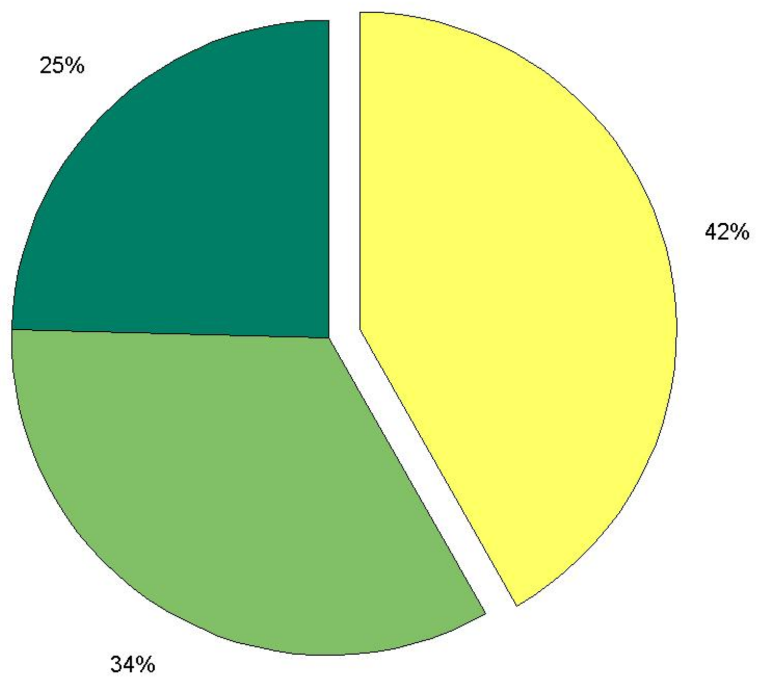

Figure 17: Pie chart

Then find the slice that contributes the most and set the corresponding explode element to 1.

```
[c,offset] = max(x);
explode(offset) = 1;
```

To create the exploded pie chart, use the statement

```
h = pie(x,explode); colormap summer
```

Histograms

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

Function	**Description**
`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.

Histograms in Cartesian Coordinate Systems

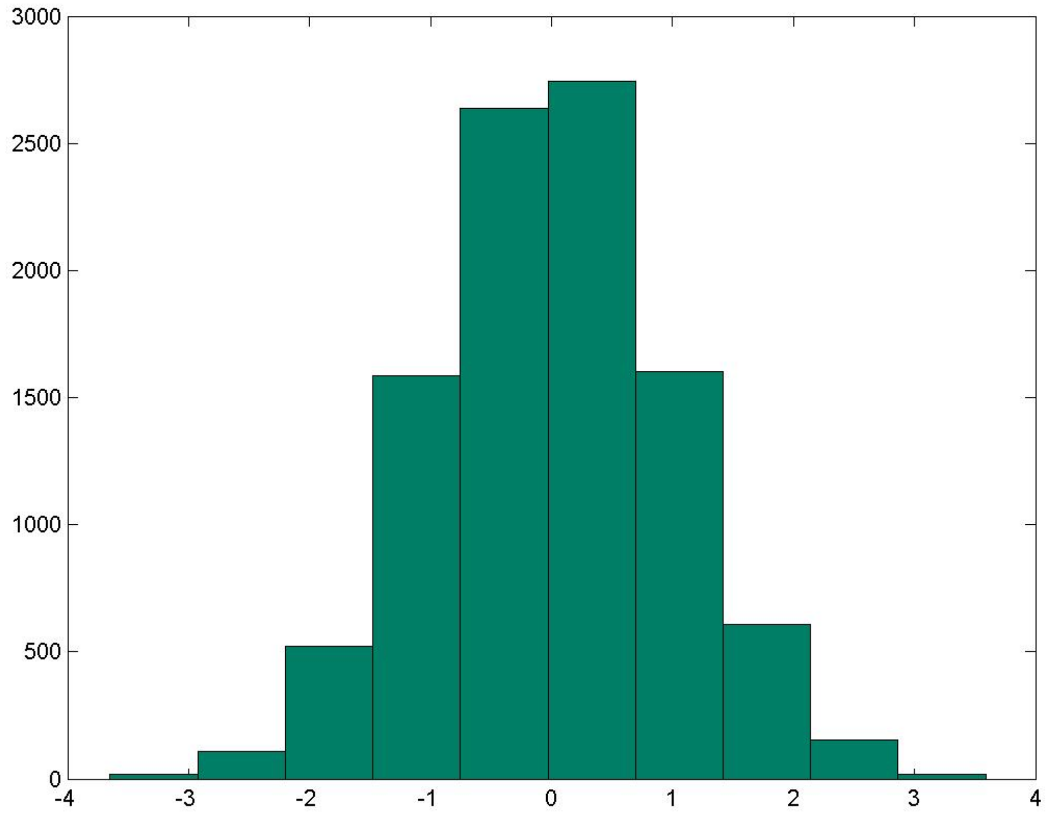

Figure 18: The histogram of a vector

The `hist` function shows the distribution of the elements in `Y` as a histogram with equally spaced bins between the minimum and maximum values in `Y`. If `Y` is a vector and is the only argument, hist creates up to 10 bins. For example,

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

generates 10000 random numbers and creates a histogram with 10 bins distributed along the x-axis between the minimum and maximum values of `yn`.

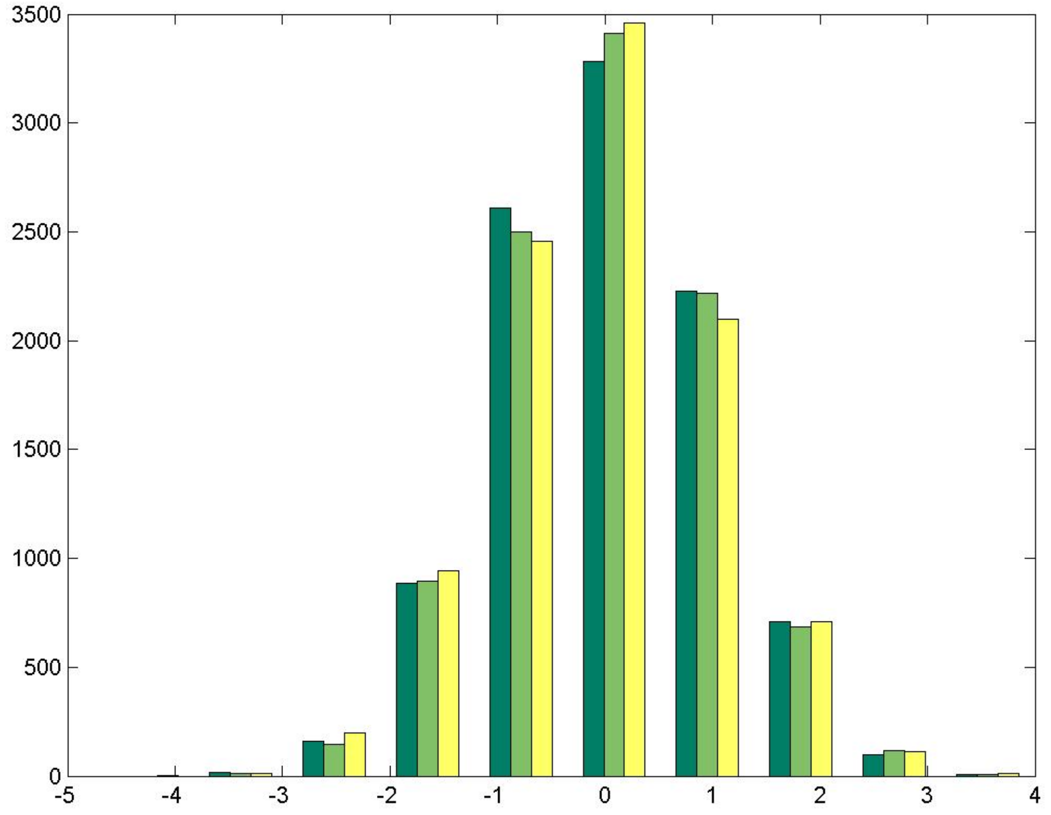

Figure 19: The histogram of a matrix

When `Y` is a matrix, `hist` creates a set of bins for each column, displaying each set in a separate colour. The statements

```
Y = randn(10000,3);
hist(Y)
```

create a histogram showing 10 bins for each column in `Y`.

Two-Dimensional Stem Plots

A stem plot displays data as lines (stems) terminated with a marker symbol at each data value. In a 2-D graph, stems extend from the x-axis.

The stem function displays two-dimensional discrete sequence data. For example, evaluating the function $y = e^{-\alpha t}\cos(\beta t)$ with the values

```
alpha = .02; beta = .5; t = 0:4:200;

y = exp(-alpha*t).*sin(beta*t);
```

yields a vector of discrete values for `y` at given values of `t`. A stem plot of the function plots only discrete points on the curve.

```
stem(t,y)
```

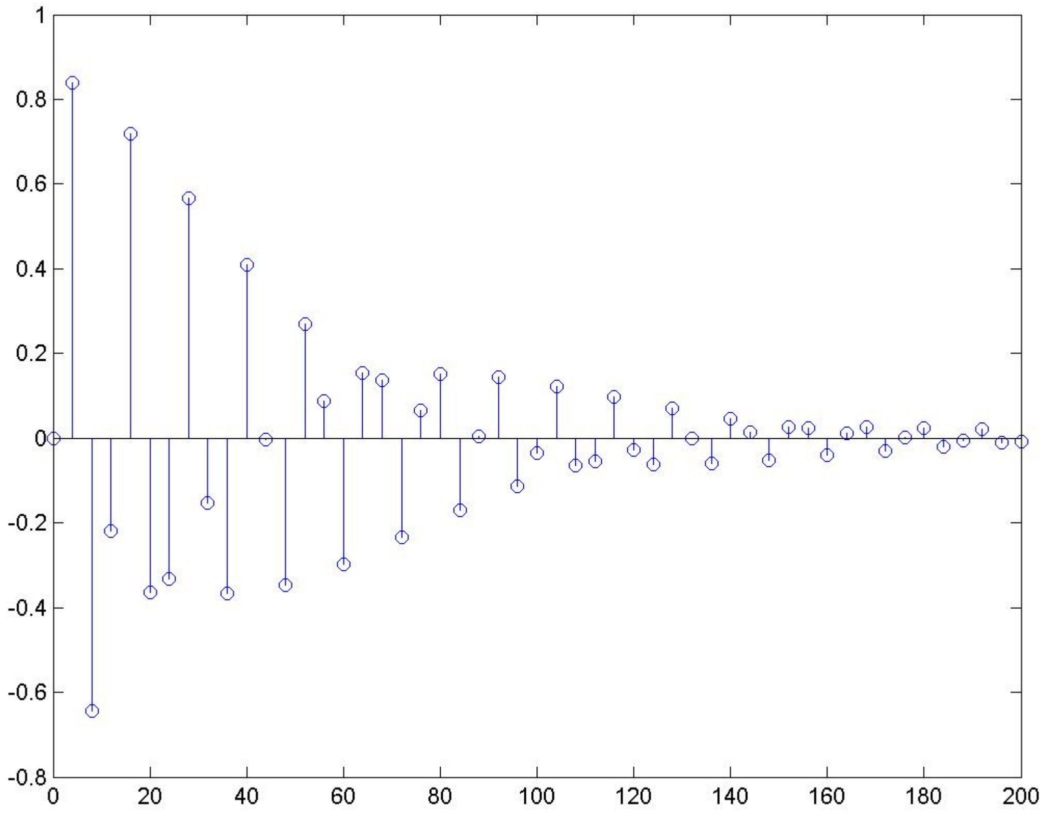

Figure 20: The stem plot

Contour Plots

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

Function	**Description**
`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 colour.
`contourc`	Low-level function to calculate the contour matrix used by the other contour functions.
`meshc`	Creates a mesh plot with a corresponding 2-D contour plot.
`surfc`	Creates a surface plot with a corresponding 2-D contour plot.

Creating Simple Contour Plots

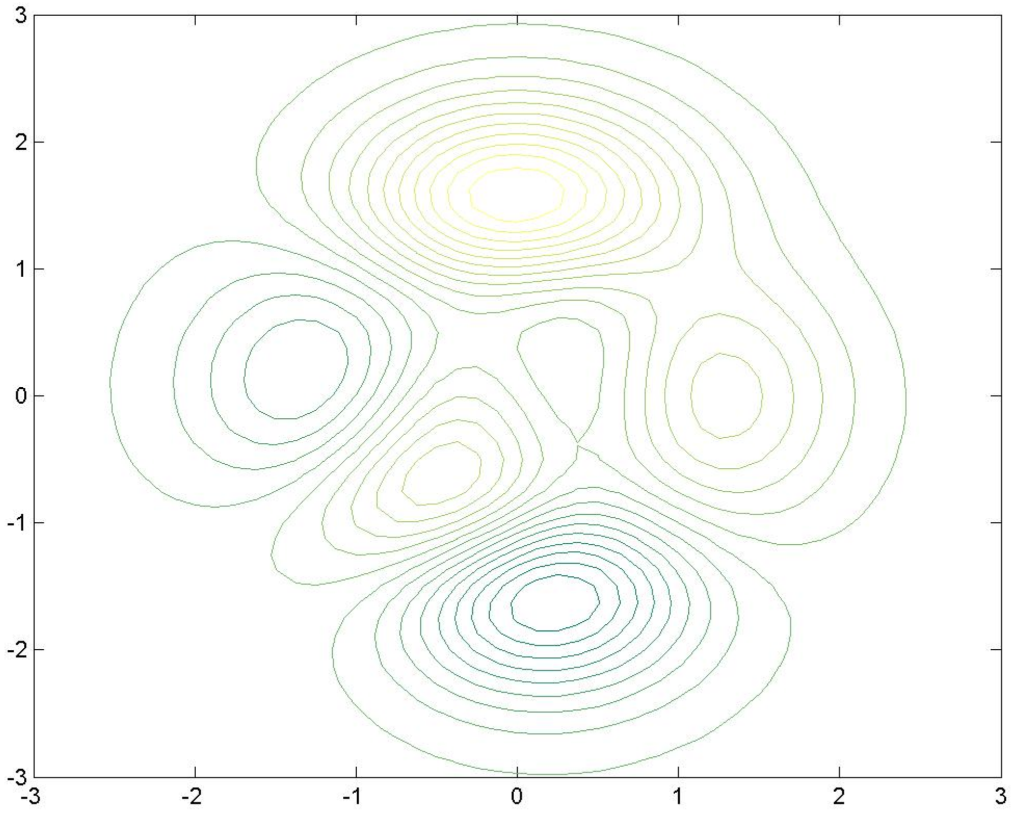

Figure 21: The contours of the `peaks` function

`contour` and `contour3` display 2- and 3-D contours, respectively. They require only one input argument — a matrix interpreted as heights with respect to a plane. In this case, the contour functions determine the number of contours to display based on the minimum and maximum data values.

To explicitly set the number of contour levels displayed by the functions, you specify a second optional argument.

For example, the statements

```
[X,Y,Z] = peaks;
contour(X,Y,Z,20)
```

display 20 contours of the `peaks` function in a 2-D view.

The statements

```
[X,Y,Z] = peaks;
contour3(X,Y,Z,20)
h = findobj('Type','patch');
set(h,'LineWidth',2)
%
title('Twenty Contours of the peaks Function')
```

display 20 contours of the `peaks` function in a 3-D view and increase the line width to 2 points.

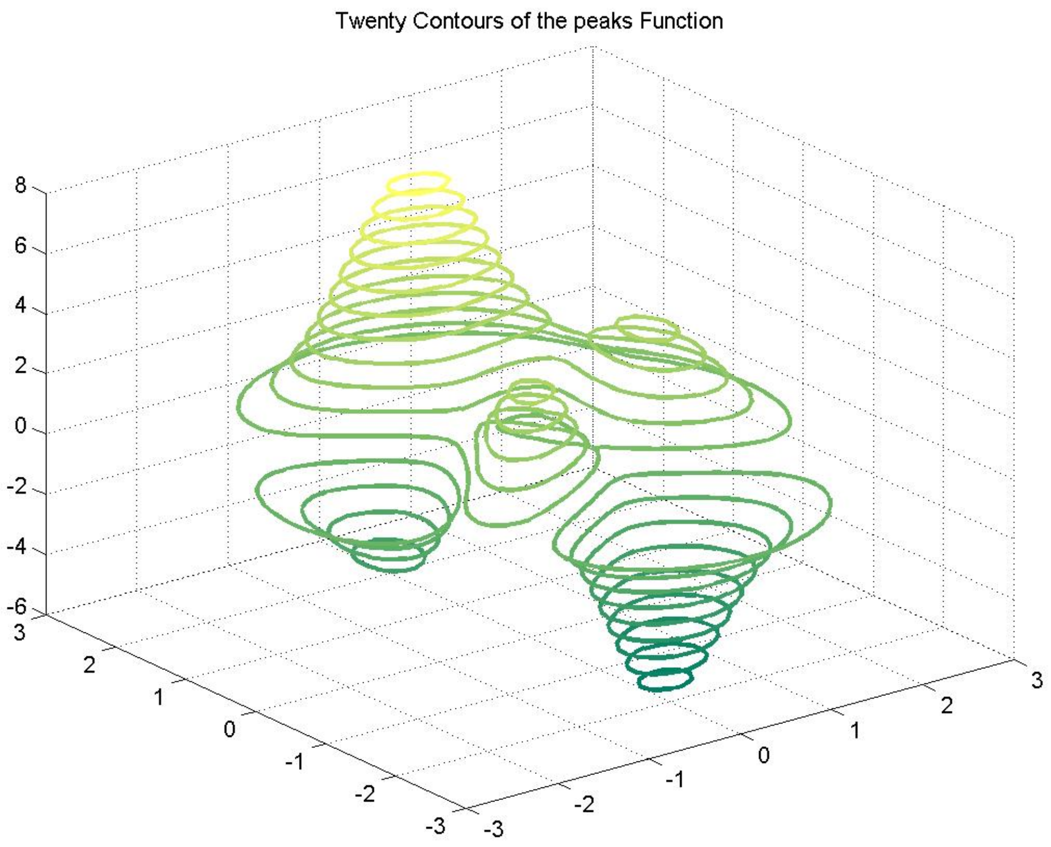

Figure 22: The contours of the `peaks` function in a 3-D view

Filled Contours

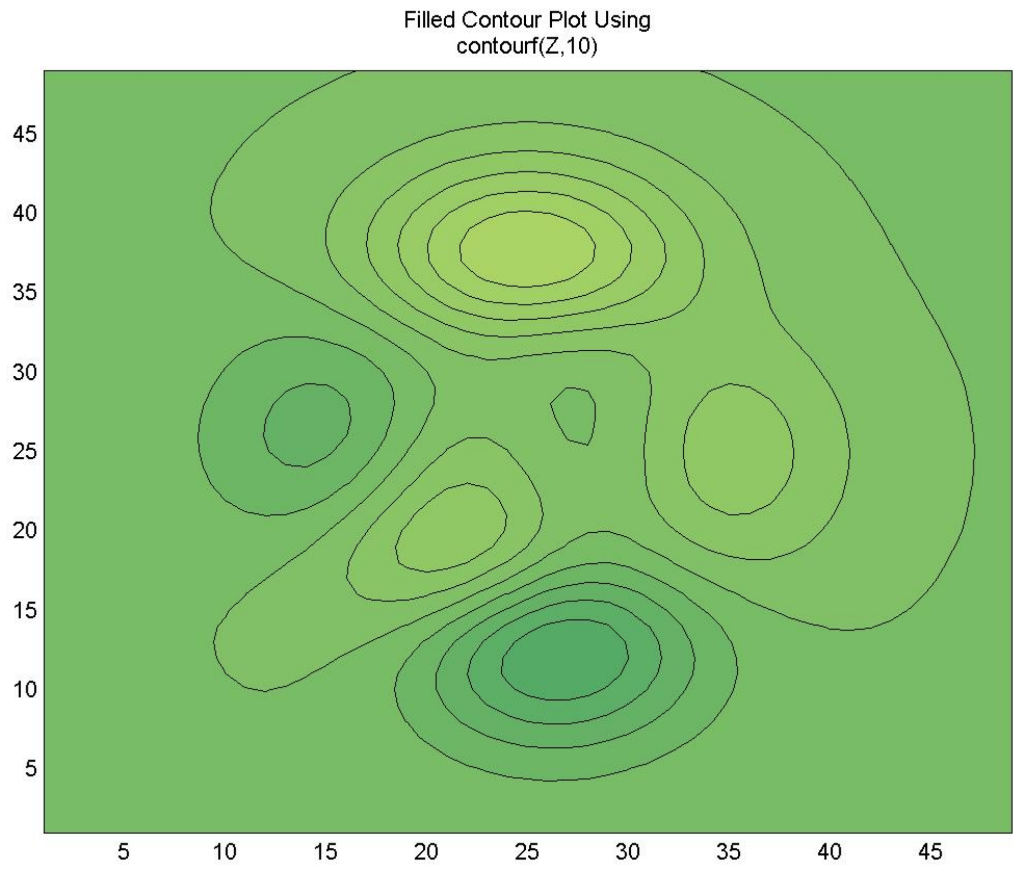

Figure 23: Filled contours

`contourf` displays a two-dimensional contour plot and fills the areas between contour lines. Use `caxis` to control the mapping of contour to colour. For example, this filled contour plot of the peaks data uses `caxis` to map the fill colours into the center of the colormap.

```
Z = peaks;
[C,h] = contourf(Z,10);
caxis([-20 20])
%
title({'Filled Contour Plot Using',...
'contourf(Z,10)'})
```

Problems

1. The Taylor approximation for $\log(1+x)$ is given by

$$T_n(x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + (-1)^{n+1}\frac{x^n}{n}.$$

   - Write a MATLAB function that sums the series while the absolute value of the current term is greater than or equal to the variable *tol*.
   - Write a MATLAB script that calculates the above approximation for $x = 0.5 : 0.01 : 1.0$ and $tol = 0.005$. Check the result by using the MATLAB function `log`. Plot the results on the same graph. The title should indicate the value of *tol*. Label the axes and use legend.

2. Draw a three-dimensional plot of the function

$$z = x\exp(-x^2 - y^2)$$

   for $x = -2 : 0.2 : 2$ and $y = -2 : 0.2 : 2$.

3. (For pass with distinction). Create and display the three-dimensional graph of the function

$$f = |\sqrt[4]{z^4 - 1}|$$

   of the complex variable z for $|z| < 1$. *Hint*: use MATLAB function `pol2cart`.